UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BRUNO DUARTE

IMPLEMENTAÇÃO DE META-HEURÍSTICAS PARA O PROBLEMA DA COLORAÇÃO DE VÉRTICES E DA ALOCAÇÃO DE REGISTRADORES

PATO BRANCO

2022

BRUNO DUARTE

IMPLEMENTAÇÃO DE META-HEURÍSTICAS PARA O PROBLEMA DA COLORAÇÃO DE VÉRTICES E DA ALOCAÇÃO DE REGISTRADORES

Implementation of Meta-heuristic for the Graph Coloring Problem and of Register Allocation

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Marco Antonio de Castro Barbosa

PATO BRANCO

2022

4.0 Internacional

Esta licença permite remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es) e que licenciem as novas criações sob termos idênticos. Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

BRUNO DUARTE

IMPLEMENTAÇÃO DE META-HEURÍSTICAS PARA O PROBLEMA DA COLORAÇÃO DE VÉRTICES E DA ALOCAÇÃO DE REGISTRADORES

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de Aprovação: 07 de abril de 2022.

___

Prof. Dr. Marco Antonio de Castro Barbosa
Doutorado em Informática
Universidade Tecnológica Federal do Paraná (UTFPR) – Pato Branco

___

Prof. Me. Silvio Luis Bragatto Boss
Mestrado em Informática
Universidade Tecnológica Federal do Paraná (UTFPR) – Pato Branco

___

Prof. Dr. Gustavo Weber Denardin
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná (UTFPR) – Pato Branco

PATO BRANCO

2022

Dedico este trabalho à minha mãe, a
maior incentivadora da minha busca pela
realização dos meus sonhos e objetivos.

AGRADECIMENTOS

Agradeço primeiramente a mim, pois sem toda minha dedicação e teimosia, este trabalho não poderia ter sido realizado. Em segundo lugar, agradeço minha família, por entender as vezes em que precisei me ausentar dos compromissos familiares "porque estava rodando o TCC". Agradeço também, a minha namorada, Fernanda, que deve ter sido a pessoa que mais me ouviu reclamar das dificuldades deste trabalho, sempre me incentivando e fazendo enxergar o lado bom de tudo.

Agradeço ao meu orientador, Professor Marco Antonio Barbosa, por ter aceitado me guiar nesta jornada, e também por ter feito parte do início do meu relacionamento com a pesquisa e o mundo das publicações acadêmicas. Agradeço também ao Professor Marcelo Teixeira, por ter feito parte da minha jornada acadêmica, sendo meu orientador em diversas empreitadas.

Preciso agradecer também, aos demais integrantes do "quarteto"do TCC, meus amigos Fábio Henrique Kurpel, Lucas Caldeira de Oliveira e Lucas Volkmer Hendges, que leram meu texto quase tantas vezes quanto eu mesmo, e me ajudaram a organizar meus pensamentos na forma de palavras.

Quero aproveitar também, para prestar meus agradecimentos à Professora Salete, da antiga 1ª série do fundamental, que me incentivou a gostar das ciências exatas, partindo da matemática básica, o que foi fundamental para me interessar pela computação.

Agradeço também as várias outras pessoas, que de uma forma ou de outra, contribuíram na minha jornada acadêmica até aqui. Não mencionarei nomes, para não esquecer de ninguém, mas todos sabem a contribuição que tiveram.

A todos, meus mais sinceros agradecimentos.

RESUMO

O Problema da Coloração de Grafos é um dos principais desafios da computação, tendo origem no século XIX, e sendo um dos grandes alicerces da teoria dos grafos. Este consiste em colorir regiões de um grafo, de modo que áreas vizinhas não recebam a mesma cor. Diversos outros problemas podem ser encarados através de uma abordagem via Coloração de Grafos, como a Alocação de Registradores no processo de compilação de programas. Neste, variáveis cujas faixas vivas – tempo entre declaração e último uso – interferem entre si, não podem ser alocadas ao mesmo registrador, e ao menos uma delas deve ser armazenada na memória. Ambos os problemas mencionados pertencem a classe dos NP-Completos, e lidam com gestão de poucos recursos, cores e registradores, sendo necessário trabalhar com inteligência em sua utilização. A resolução exata deste tipo de problema demanda tempo de computação exponencial, sendo as meta-heurísticas alternativas extremamente viáveis para encontrar soluções de qualidade em tempo polinomial. Este trabalho propôs a implementação das meta-heurísticas Busca Tabu e *Simulated Annealing*, para o Problema da Coloração de Grafos, e para o problema derivado deste, a Alocação de Registradores, com o objetivo de investigar o comportamento destes métodos e compará-los entre si. Após uma série de experimentos com casos de teste do DIMACS, foi possível verificar que a Busca Tabu implementada é capaz de produzir soluções de melhor qualidade para a Coloração de Grafos, enquanto o *Simulated Annealing* teve menores tempos de execução. Para a Alocação de Registradores, a Busca Tabu demonstrou-se capaz de alcançar melhores soluções, com menor tempo de execução.

Palavras-chave: coloração de grafos; alocação de registradores; meta-heurística; busca tabu; simulated annealing.

ABSTRACT

The Graph Coloring Problem is one of the major challenges in computing, having its origins in the 19th century, and being one of the great foundations of graph theory. It consists in coloring regions of a graph, such as vertices or edges, so that neighboring areas do not receive the same color. Several other problems can be faced through an approach via Graph Coloring, such as Register Allocation in the compilation process. Here, variables that intersect their life span – time between declaration and last use – cannot be allocated to the same register, and at least one of them must be stored in memory. Here, variables that intersect their live-ranges – time between declaration and last use – cannot be allocated to the same register, and at least one of them must be stored in memory. Both problems mentioned belong to the NP-Complete class, and deal with the management of few resources, colors and registers, being necessary to work intelligently in their utilization. Finding the exact solution of this type of problem demands exponential computation time, being the meta-heuristics extremely suitable alternatives to find solutions of quality in polynomial time. This work proposed the implementation of Tabu Search and Simulated Annealing meta-heuristics for the Graph Coloring Problem, and for the problem derived from it, Register Allocation, with the objective of investigating the behavior of these methods and comparing them with each other. After a series of experiments with DIMACS test cases, it was possible to verify that the implemented Tabu Search is capable of producing better quality solutions for Graph Coloring, while Simulated Annealing had lower execution times. For Register Allocation, Tabu Search proved to be able to achieve better solutions, with less execution time.

Keywords: graph coloring; register allocation; meta-heuristic; tabu search; simulated awnnealing.

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo para Busca Tabu Simples 32
Algoritmo 2 – Algoritmo do Simulated Annealing 34
Algoritmo 3 – Geração de solução aleatória 39
Algoritmo 4 – Método Guloso 40
Algoritmo 5 – Mecanismo de movimentação 41

LISTA DE ILUSTRAÇÕES

Figura 1 – Grafo com 5 vértices e sugestão de arranjo de cores para resolver o GCP com $k = 3$. 19
Figura 2 – Estrutura interna de um compilador. 23
Figura 3 – Etapas internas do *front end* de um compilador. 24
Figura 4 – Etapas internas do *back end* de um compilador. 26
Figura 5 – Diagrama de blocos do algoritmo de Chaitin para alocação de registradores por coloração de grafos. 30
Figura 6 – Fluxograma da adaptação realizada ao algoritmo de Chaitin 43
Figura 7 – Resultados da Busca Tabu para o GCP 47
Figura 8 – Média do tempo de execução da Busca Tabu para a instância $lei450_15c$ . . 49
Figura 9 – Resultados do SA para o GCP 51
Figura 10 – Média do tempo de execução do *Simulated Annealing* para a instância $lei450_15c$ 53
Figura 11 – Número de *spillings* com 8 registradores 53
Figura 12 – Gráfico comparativo dos tempos de execução do processo de alocação com 8 registradores para a instância *zeroin.i.1* 54
Figura 13 – Número de *spillings* com 12 registradores 55
Figura 14 – Gráfico comparativo dos tempos de execução do processo de alocação com 12 registradores para a instância *zeroin.i.1* 56

LISTA DE TABELAS

Tabela 1 – Apresentação dos parâmetros dos grafos utilizados no GCP 38
Tabela 2 – Apresentação dos parâmetros dos grafos utilizados no problema da alocação de registradores 38
Tabela 3 – Comparativo do menor número médio de conflitos obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória. 49
Tabela 3 – Comparativo do menor número médio de conflitos obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória. 50
Tabela 4 – Comparativo do menor número médio de spillings obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória. 55
Tabela 5 – Comparativo do menor número médio de spillings obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória. 56

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

ACO	*Ant Colony Optimization*
AG	Algoritmo Genético
AM	*Algoritmo Memético*
BL	Busca Local
BT	Busca Tabu
GCP	*Graph Coloring Problem*
PSA	*Parallel Simulated Annealing*
PSO	*Particle Swarm Optimization*
RI	Representação Intermediária
RLF	*Recursively Large First*
SA	*Simulated Annealing*
SDMA	*Solution Driven Multilevel Algorithms*
TLBO	*Teaching Learning-Based Optimization*
UTFPR	Universidade Tecnológica Federal do Paraná
VRP	*Vehicle Routing Problem*

ACRÔNIMOS

DSATUR	Degree of Saturation
SAT	Satisfiability Problem

SUMÁRIO

1 INTRODUÇÃO . . . 12
1.1 OBJETIVOS . . . 14
1.1.1 OBJETIVOS ESPECÍFICOS . . . 14
1.2 ORGANIZAÇÃO DO TRABALHO . . . 14

2 REVISÃO DE LITERATURA . . . 15
2.1 CLASSES DE PROBLEMAS . . . 15
2.1.1 TÉCNICAS PARA RESOLUÇÃO DE PROBLEMAS NP-COMPLETOS . . . 16
2.2 O PROBLEMA DA COLORAÇÃO DE GRAFOS . . . 18
2.2.1 APLICAÇÕES DO GCP . . . 22
2.3 COMPILADORES . . . 23
2.3.1 *FRONT END* . . . 24
2.3.2 *BACK END* . . . 25
2.4 ALOCAÇÃO DE REGISTRADORES . . . 26
2.5 META-HEURÍSTICAS . . . 31
2.5.1 BUSCA TABU . . . 31
2.5.2 *SIMULATED ANNEALING* . . . 33

3 DETALHES DE IMPLEMENTAÇÃO E ORGANIZAÇÃO DOS EXPERIMENTOS . . . 36
3.1 INSTÂNCIAS DE TESTE . . . 36
3.2 DETALHES DE IMPLEMENTAÇÃO . . . 38
3.2.1 IMPLEMENTAÇÃO DA BUSCA TABU . . . 41
3.2.2 IMPLEMENTAÇÃO DO *SIMULATED ANNEALING* . . . 42
3.3 DIFERENÇAS ENTRE A IMPLEMENTAÇÃO DO GCP E DA ALOCAÇÃO DE REGISTRADORES . . . 43
3.4 DEFINIÇÃO DOS EXPERIMENTOS E PARÂMETROS UTILIZADOS . . . 44

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS . . . 46
4.1 RESULTADOS DA COLORAÇÃO DE GRAFOS . . . 46
4.2 RESULTADOS DA ALOCAÇÃO DE REGISTRADORES . . . 51

5 CONCLUSÃO . . . 58
5.1 TRABALHOS FUTUROS . . . 59

REFERÊNCIAS . . . 61

1 INTRODUÇÃO

O Problema da Coloração de Grafos consiste em atribuir cores aos elementos de um grafo, de modo que elementos adjacentes não sejam coloridos com a mesma cor. Existem diferentes versões do problema, como a Coloração de Vértices, em que dois vértices adjacentes, isto é, com aresta entre si, devem ser coloridos com cores diferentes; e a Coloração de Arestas, que busca colorir um grafo de modo que arestas adjacentes recebam cores diferentes (CHARTRAND; ZHANG, 2019). No contexto desse trabalho, ao mencionar a Coloração de Grafos, deve-se remeter à Coloração de Vértices.

Este problema pertence à classe dos problemas NP-Completos (KARP, 1972), não havendo portanto, um algoritmo capaz de resolvê-lo de maneira ótima em tempo polinomial. Entretanto, se algum algoritmo puder resolver este – ou qualquer outro problema desta classe – de maneira exata e eficiente, toda a classe NP poderá ser resolvida em tempo polinomial, o que renderia um prêmio em dinheiro ao responsável por encontrar o algoritmo (CARLSON; JAFFE; WILES, 2006).

Na literatura existem diversas abordagens ao Problema da Coloração de Grafos, que vão desde heurísticas a meta-heurísticas. Dentre as primeiras tentativas de solucionar a Coloração de Grafos, tem-se os métodos gulosos, que operam de maneira sequencial, como as heurísticas DSATUR (BRÉLAZ, 1979) e RLF (LEIGHTON, 1979). Esses métodos, apesar de chegarem a soluções de maneira rápida, não apresentam tanta qualidade, sendo atualmente mais utilizados na geração de soluções iniciais para meta-heurísticas (LÜ; HAO, 2010).

Meta-heurísticas também foram aplicadas a Coloração de Grafos, como o *Simulated Annealing* (CHAMS; HERTZ; DE WERRA, 1987; JOHNSON; ARAGON *et al.*, 1991) e a Busca Tabu (HERTZ; WERRA, 1987; DORNE; HAO, 1999). Além destas, meta-heurísticas baseadas em algoritmos populacionais foram aplicadas ao problema, como os Algoritmos Genéticos (FLEURENT; FERLAND, 1996), Meméticos (LÜ; HAO, 2010), Otimização por Enxame de Partículas (AOKI; ARANHA; KANOH, 2015) e Otimização por Colônia de Formigas (SALARI; ESHGHI, 2005). Algoritmos populacionais correspondem ao estado da arte da coloração de grafos (SUN *et al.*, 2021), pois compreendem a maior parte das técnicas utilizadas em trabalhos atuais, além de apresentarem os melhores resultados. Entretanto, algoritmos populacionais apresentam um maior custo de memória computacional, pois, diferente da Busca Tabu, por exemplo, precisam armazenar um conjunto maior de soluções a cada iteração.

É possível reduzir o Problema da Coloração de Grafos a diversos outros problemas,

como o agendamento de aulas, distribuição de frequências etc. (PARDALOS; MAVRIDOU; XUE, 1998). A alocação de registradores é, também, um problema que pode ser encarado como um problema de coloração.

De modo a possibilitar a execução de programas desenvolvidos por um programador, é necessária a geração de um código-objeto (também chamado de código alvo) que possa ser compreendido a nível de máquina. Essa conversão pode ser realizada por um compilador. Em sua forma mais complexa, um compilador traduz um programa em uma linguagem de programação, em um programa executável em linguagem de máquina para uma arquitetura de destino (SANTOS; LANGLOIS, 2018).

Durante a compilação, o alocador de registradores é o responsável por garantir o melhor uso possível dos registradores disponíveis, ao definir quais variáveis devem ser armazenadas diretamente em registradores, e quais devem ser armazenadas inicialmente na memória (COOPER, K.; TORCZON, 2011), para depois serem carregadas aos registradores e ser utilizadas. Além disso, é também o alocador que determina em qual registrador cada variável deve ser armazenada (COOPER, K.; TORCZON, 2011). Para armazenar variáveis na memória, é preciso inserir operações *load* e *store* no código-objeto, o que reduz o desempenho de execução.

O problema da alocação é NP-Completo (SETHI, 1975), e pode ser reduzido a um problema de coloração de grafos, com cada cor sendo equivalente a um registrador. Um dos primeiros trabalhos com essa abordagem foi realizado por Chaitin *et al.* (1981), que se tornou uma das bases fundamentais desse segmento de pesquisa (PEREIRA, 2008).

Outras abordagens possíveis para sua resolução são as meta-heurísticas, como os trabalhos de Kri e Feeley (2004) e Lintzmayer, Mulati e Anderson Faustino da Silva (2011), com Algoritmos Genéticos e Otimização por Colônia de Formigas, respectivamente. Entretanto, o uso de meta-heurísticas aplicadas ao processo de compilação é escasso (KRI; GÓMEZ; CARO, 2005), o que torna a aplicação de outras meta-heurísticas à alocação de registradores um objeto de estudo relevante. Ademais, os trabalhos citados tratam-se de algoritmos populacionais, não havendo implementações de meta-heurísticas de busca local.

Esse trabalho propõe a implementação das meta-heurísticas *Simulated Annealing* e Busca Tabu para resolução do Problema da Coloração de Grafos. Estas meta-heurísticas já se demonstraram eficazes a problemas de grafos, sendo de interesse avaliar o desempenho de ambas, e compará-las uma com a outra e com a literatura. Além disso, propõe-se também aplicar estas meta-heurísticas ao Problema da Alocação de Registradores, para estudar o comportamento de meta-heurísticas nesta etapa do processo de compilação, e comparar o desempenho de ambas as

técnicas.

1.1 OBJETIVOS

Implementar meta-heurísticas para resolver o Problema da Coloração de Grafos, abordando também um problema derivado, a Alocação de Registradores em compiladores.

1.1.1 OBJETIVOS ESPECÍFICOS

1. Implementar as meta-heurísticas Busca Tabu e *Simulated Annealing* para a Coloração de Grafos;
2. Aplicar os métodos implementados na Alocação de Registradores;
3. Comparar os resultados de cada método implementado, entre si.

1.2 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 apresentam-se as bases teóricas essenciais para entendimento e realização deste trabalho. A metodologia utilizada nesse trabalho pode ser vista no Capítulo 3, onde abordam-se questões técnicas a respeito do método escolhido para abordar o problema de alocação de registradores, das instâncias de teste utilizadas, e da organização dos experimentos. A apresentação e discussão dos resultados obtidos é realizada no Capítulo 4. Finalmente, as principais conclusões do trabalho, bem como perspectivas de trabalhos futuros, são expostas no Capítulo 5.

2 REVISÃO DE LITERATURA

Nesse capítulo serão introduzidos os conceitos teóricos necessários para execução do trabalho. Na Seção 2.1 serão abordadas as classes de problemas computacionais, como P e NP, além dos problemas NP-Completos. Além disso, serão apresentadas algumas técnicas de resolução que costumam ser utilizadas nos diferentes problemas que as compõem.

Na Seção 2.2 o Problema da Coloração de Grafos é apresentado, bem como alguns dos principais trabalhos e abordagens para sua solução.

A contextualização a respeito de compiladores de programas computacionais será realizada na Seção 2.3. Serão abordadas as etapas internas que compõem o processo de compilação. O problema da alocação de registradores será discutido com detalhes na Seção 2.4. Serão abordados aspectos da alocação em blocos locais, alocação global, técnicas de abordagens. Um foco maior será dado à representação da alocação como o Problema da Coloração de Grafos, que é a abordagem a ser seguida no trabalho. Serão discutidos também os principais trabalhos relacionados ao problema.

Por fim, a Seção 2.5 apresenta as meta-heurísticas a serem implementadas neste trabalho.

2.1 CLASSES DE PROBLEMAS

Um problema consiste em qualquer questão que demande de uma resposta. Garey e Johnson (1990) definem que um problema é composto por uma especificação de seus parâmetros e dos requisitos que uma possível solução devem atender.

Os problemas podem ser classificados em três categorias: problemas de decisão, localização e otimização. Enquanto os problemas de decisão consistem em determinar se há ou não uma solução, os de localização visam encontrar uma solução viável. Por sua vez, os problemas de otimização buscam encontrar a melhor solução possível.

Também é possível classificar problemas de acordo com a complexidade de tratamento, dentro das classes ***P*** e ***NP***. Cook (2006) define a classe P como a classe de problemas de decisão que podem ser resolvidos por algoritmos com número de passos limitados a um polinômio fixo em função do tamanho da entrada. A classe NP consiste nos problemas que podem ser verificados, mas não resolvidos, por algoritmos determinísticos em tempo polinomial (GOODRICH; TAMASSIA, 2009). Estes têm complexidade de ordem exponencial em função da entrada, sendo considerados intratáveis por métodos determinísticos.

Existe um problema em aberto que visa responder se $P = NP$. Cook (2006) afirma que é trivial mostrar que $P \subseteq NP$. Entretanto, provar a igualdade (ou a diferença) entre as classes é uma tarefa considerada extremamente difícil, visto que até mesmo a prova de teoremas é um problema da classe NP (DASGUPTA; PAPADIMITRIOU; VAZIRANI, 2008). Caso $P \neq NP$, então a distinção entre P e NP-P (lê-se *NP menos P* é de extrema importância pois implica que os problemas em P podem ser resolvidos em tempo polinomial, enquanto os problemas em NP-P são intratáveis (GAREY; JOHNSON, 1990).

A classe **NP-difícil** pode ser definida como segue: considerando um problema M, este é NP-difícil se todo problema $L \in NP$ pode ser reduzido a M em tempo polinomial (GOODRICH; TAMASSIA, 2009). Caso um problema seja NP-difícil e esteja em NP, é dito que ele é **NP-Completo** (GAREY; JOHNSON, 1990). Resolver um problema NP-Completo com um algoritmo determinístico em tempo polinomial implicaria que todos os demais problemas em NP também poderiam ser resolvidos nesse tempo, provando que $P = NP$ (GAREY; JOHNSON, 1990). Na classe dos problemas NP-Completos é possível citar como exemplos:

- O Problema da Satisfatibilidade (SAT) é um dos núcleos das pesquisas de problemas NP-Completos. Tem como objetivo encontrar uma combinação de valores verdade para variáveis, de modo que uma fórmula lógica escrita na Forma Normal Conjuntiva seja satisfatível (GU *et al.*, 1996);
- O Problema do Caixeiro Viajante (*The Traveling Salesman Problem* –), cujo objetivo é encontrar a menor rota entre um conjunto de cidades, passando uma única vez por cada uma e voltando ao ponto de partida (GOLDBARG, 2005);
- O Problema de Roteamento de Veículos (*Vehicle Routing Problem* – VRP), que consiste em encontrar a melhor rota para uma frota de veículos a serviço de um conjunto de clientes (KUMAR, S. N.; PANNEERSELVAM, 2012).

Além destes, tem-se também o Problema da Coloração de Grafos (*Graph Coloring Problem* - GCP), a ser abordado na Seção 2.2.

Existem diferentes técnicas para resolução de problemas, as quais serão abordadas a seguir.

2.1.1 TÉCNICAS PARA RESOLUÇÃO DE PROBLEMAS NP-COMPLETOS

Ao projetar um algoritmo para resolver um problema, é preciso levar em consideração sua complexidade, o que está intimamente relacionado com a classe do problema. Para problemas

NP-Completos, a complexidade para encontrar a melhor solução é não polinomial (exceto se $P = NP$). Talbi (2009) classifica os métodos de solução em quatro categorias: exatos, aproximados, gulosos e meta-heurísticas.

Os **métodos exatos** são capazes de encontrar a melhor solução possível, varrendo todo o espaço de busca das soluções. Entretanto, em problemas de complexidade exponencial, isto é, cujo espaço de busca varia exponencialmente de acordo com o tamanho da entrada, os métodos exatos podem ser aplicados somente em casos de teste pequenos (TALBI, 2009). Para o Problema da Coloração de Grafos, por exemplo, Talbi (2009) afirmava que, até então, os métodos exatos do estado da arte poderiam resolver, de maneira ótima, instâncias com número de vértices cuja ordem de grandeza era 100 – o valor exato varia de acordo com a estrutura interna do grafo. Como exemplos de métodos exatos podem ser citados os algoritmos *branch and bound*, *branch and cut*, algoritmo A^* e a programação dinâmica.

Os **métodos aproximados** têm como objetivo encontrar uma solução aproximada, pois nem sempre é possível encontrar uma solução exata para um problema. Na classe dos métodos aproximados, Talbi (2009) distingue duas famílias de algoritmos: os aproximados e os heurísticos. Enquanto os heurísticos permitem encontrar soluções consideradas boas em tempo razoável, os aproximados garantem qualidade e tempo de execução limitados por alguns fatores específicos. Um algoritmo de ϵ-aproximação gera soluções próximas da solução ótima por um fator ϵ que pode ser uma constante ou uma função do tamanho da instância de entrada (TALBI, 2009). Os algoritmos aproximados podem ser aplicados a problemas com entradas consideravelmente maiores que as de métodos exatos.

Talbi (2009) cita os **métodos gulosos** como um método construtivo de solução. Inicia-se de uma solução vazia, construindo-a ao atribuir valores, uma variável de decisão por vez, até que a solução seja completa. Cada decisão é a melhor possível no momento, como, no caso de problemas com grafos, selecionar uma aresta com menor peso, ou ainda, um vértice com grau mais alto (GONZALEZ, 2018).

As **meta-heurísticas** por sua vez permitem abordar problemas com instâncias grandes entregando soluções satisfatórias com tempo polinomial, ao abrir mão da garantia de otimalidade (TALBI, 2009). Enquanto métodos aproximados correm o risco de ficar presos em soluções de ótimo local, as meta-heurísticas incluem procedimentos que permitem fugir dessas regiões (GENDREAU; POTVIN *et al.*, 2010). Dentre as principais aplicações de meta-heurísticas, é possível citar o aprendizado de máquinas, a mineração de dados em bioinformática, biologia computacional, finanças, processamento de imagens, simulações, problemas de roteamento, entre

outras (TALBI, 2009).

2.2 O PROBLEMA DA COLORAÇÃO DE GRAFOS

O problema da coloração de grafos - do inglês, *Graph Coloring Problem* (GCP) - consiste em colorir um grafo de modo que não haja vértices adjacentes compartilhando a mesma cor.

Proposto inicialmente em 1852, o GCP deriva de estudos a respeito do Problema das Quatro Cores, cuja conjectura inicial afirmava que todas as regiões de todos os mapas poderiam ser coloridas com 4 ou menos cores, de modo que duas regiões fronteiriças fossem coloridas com cores diferentes (CHARTRAND; ZHANG, 2019).

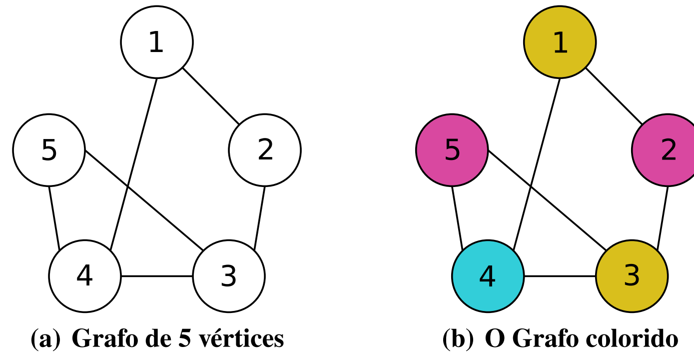
O problema em questão pode ser definido como segue: Dado um grafo arbitrário, não direcionado, $G = (V,E)$, em que $V = \{v_1,v_2,...,v_n\}$ é o conjunto de vértices e $E = \{e_1, e_2,...,e_m\} \subseteq (E \times E)$ é o conjunto de arestas. Define-se adjacência entre vértices quando existe uma aresta $e = \left(v_i, v_j\right) \in E$, ou seja, dois vértices v_i, v_j possuem uma aresta entre si (PARDALOS; MAVRIDOU; XUE, 1998). O problema da coloração de grafos consiste em colorir um grafo de modo que não seja atribuída uma mesma cor a dois vértices adjacentes (SALARI; ESHGHI, 2005). As cores podem ser rótulos numéricos, caracteres, ou, no escopo desse trabalho, registradores.

Um grafo é dito k-colorável se existe uma coloração de G a partir de um conjunto de k cores, e o menor número k de cores que permite essa coloração é chamado de número cromático, e é denotado por $\chi(G)$ (CHARTRAND; ZHANG, 2019). O problema pode ser abordado sob a ótica de um problema de decisão, que verifica se é possível colorir o grafo G com k cores, ou como um problema de otimização, que busca encontrar $\chi(G)$.

A formalização do problema de decisão pode ser dada como segue: dado um grafo G e um inteiro k, o GCP consiste em determinar se $\chi(G) \leq k$ (PARDALOS; MAVRIDOU; XUE, 1998).

Um exemplo do GCP pode ser visto na Figura 1. Nela, tem-se um grafo não direcionado $G = (V,E)$, com o conjunto de vértices $V = \{1,2,3,4,5\}$ e de arestas $E = \{(1,2), (1,4), (2,3), (3,4), (3,5), (4,5)\}$. Na Figura 1(a) é possível observar o grafo não colorido. Dado um número de cores $k = 3$, uma possível coloração para o grafo é apresentada na Figura 1(b). Nota-se que dois vértices adjacentes não foram coloridos com a mesma cor, respeitando as restrições do GCP.

Figura 1 – Grafo com 5 vértices e sugestão de arranjo de cores para resolver o GCP com $k = 3$.



(a) Grafo de 5 vértices **(b) O Grafo colorido**

Fonte: Autoria própria.

A coloração de grafos pertence a classe dos Problemas NP-Completos, conforme provado por Karp (1972) ao demonstrar que este poderia ser reduzido ao problema 3-SAT, já sabido ser NP-Completo. Este fato foi reforçado em Garey, Johnson e Stockmeyer (1974) e Stockmeyer (1973), ao mostrarem que o problema da k-coloração de grafos permanece NP-Completa para qualquer $k \geq 3$. Como consequência de ser um problema de complexidade exponencial, a utilização de heurísticas é necessária na busca de soluções para grafos maiores.

O problema é alvo frequente de pesquisas e discussões, sendo inúmeros os trabalhos dedicados a solucionar a ele e suas variações. A seguir, serão apresentados os principais trabalhos que abordaram o problema.

Duas abordagens ao problema são apresentada em Brélaz (1979). A primeira delas trata-se da heurística DSATUR (do inglês, *Degree of Saturation*), que inicia a coloração pelo vértice de maior grau, e em seguida seleciona os próximos vértices a serem coloridos por meio do grau de saturação de cada um. Esta heurística é considerada exata para grafos bipartidos. O segundo método proposto por Brélaz (1979) trata-se do *Randall-Brown's Modified Algoritm*, uma modificação do método exato proposto em Brown (1972), incorporando conceitos do algoritmo DSATUR, ao colorir os vértices de acordo com o maior grau de saturação, e interrompendo sua execução quando o número de cores tem a mesma dimensão do clique maximal conhecido do grafo.

No mesmo ano Leighton (1979) apresenta uma heurística de coloração de grafos chamada *Recursively Large First* (RLF), que busca completar a atribuição de uma determinada cor i antes de começar a atribuir a cor $i + 1$. Como o próprio nome sugere, os vértices são coloridos de maneira recursiva, e a cada iteração a cor i é atribuída a todos os nós não adjacentes entre si que atendam ao critério de seleção do algoritmo. Este critério, por sua vez, consiste em selecionar os vértices com o maior grau.

Qu e Potkonjak (1998) apresentam uma análise acerca da viabilidade de duas técnicas de marca d'água para o problema, quando comparadas com a coloração do grafo original. Enquanto a primeira força pares de vértices escolhidos a receber cores diferentes, adicionando novas arestas entre eles, a segunda seleciona um (ou mais) conjuntos independentes do grafo original, marcando cada conjunto com exatamente uma cor.

Uma das primeiras meta-heurísticas a ser aplicada ao GCP foi o *Simulated Annealing* (SA), que é baseado na analogia entre a simulação de recozimento de sólidos e a solução de problemas de otimização combinatória (VAN LAARHOVEN; AARTS, 1987). O trabalho de Chams, Hertz e De Werra (1987) apresenta a primeira tentativa de aplicação do método, tanto em uma versão pura quanto híbrida com o RLF, demonstrando sua eficácia em relação aos métodos apresentados anteriormente na literatura. Johnson, Aragon *et al.* (1991) aplicaram três versões do SA, que diferenciam-se entre si da seguinte maneira: a primeira aceita colorações com conflitos, mas busca minimizá-los enquanto tenta minimizar o número de cores utilizadas; a segunda restringe as soluções aceitas a colorações sem nenhum conflito; e a terceira limita o número de cores disponíveis, sendo o custo da solução, o número de arestas entre vértices da mesma cor. As três versões se mostraram superiores às técnicas, até então tradicionais, para determinados tipos de grafos quando o tempo computacional disponível não é tão restrito. Uma versão paralelizada do SA foi aplicada ao GCP por Łukasik, Kokosiński e Świętoń (2007), denominada *Parallel Simulated Annealing* (PSA), que consiste na utilização de múltiplos processadores, os quais trabalham em cadeias individuais de solução, de maneira concorrente. Após um intervalo fixo de iterações, as soluções correntes são compartilhadas entre os processadores.

Após os primeiros trabalhos aplicando SA ao GCP, Hertz e Werra (1987) trouxeram uma implementação da Busca Tabu. O objetivo desta meta-heurística é, a partir de uma solução inicial, avançar passo a passo em direção a melhores soluções através de pequenos movimentos. Entretanto, a principal característica da Busca Tabu é a implementação de uma lista tabu, que contém os movimentos não permitidos na iteração corrente, evitando voltar a soluções anteriormente visitadas. No trabalho de Hertz e Werra (1987) foi apresentada uma implementação de Busca Tabu para o GCP que foi denominada TABUCOL. No mesmo trabalho, os autores apresentam uma combinação do TABUCOL com outra etapa da Busca Tabu para construir conjuntos independentes de vértices, que seriam posteriormente coloridos via TABUCOL. Dentre essas duas implementações apresentadas em Hertz e Werra (1987), os melhores resultados obtidos foram provenientes da segunda versão. Em Dorne e Hao (1999) a Busca Tabu é aplicada ao GCP, e a outros dois problemas relacionados, demonstrando-se competitiva.

Outra classe de algoritmos já aplicados ao GCP corresponde aos algoritmos populacionais, que operam sobre um conjunto de soluções, denominado população. Dentro desta classe de métodos é possível citar os Algoritmos Evolutivos, que são técnicas de otimização inspiradas na natureza. Como exemplo de meta-heurística pertencente aos Algoritmos Evolutivos, temos os Algoritmos Genéticos (AG) (HOLLAND, 1975), os quais se baseiam nos princípios da seleção natural e da recombinação genética, para construir soluções e melhorá-las com o passar de gerações de cruzamento e de mutação. O trabalho de Fleurent e Ferland (1996) apresenta uma implementação de AG, enfatizando que o método puro não atinge resultados de qualidade tão boa quanto outras heurísticas de busca, enquanto a hibridização com outras heurísticas apresenta resultados ótimos para boa parte das instâncias testadas. Dorne e Hao (1998) trouxeram uma implementação híbrida de AG cujo operador de mutação é realizada por um procedimento de Busca Tabu alcançando os melhores resultados conhecidos para os casos testados. No trabalho proposto em Hindi e Yampolskiy (2012), apresentou-se a implementação de um AG com dois diferentes métodos de mutação, escolhidos de acordo com o estado corrente da solução, resultando em melhor desempenho do algoritmo ao encontrar ótimos globais. Em Lü e Hao (2010), uma implementação de Algoritmos Meméticos (AM) é proposta para resolver o GCP. O AM proposto implementava um operador de cruzamento adaptativo com multi-parentagem, além de um critério de atualização de gerações baseado em distância e qualidade.

Outras abordagens baseadas em algoritmos populacionais bioinspirados foram aplicadas ao problema. A meta-heurística Otimização por Colônia de Formigas (sigla em inglês ACO) baseia-se no comportamento de formigas em busca de alimento, que depositam quantidades de feromônios no caminho a ser seguido. Salari e Eshghi (2005) apresenta uma implementação de ACO chamada *Max-Min Ant System*, que consiste em limitar os índices de depósito de feromônios nas rotas de solução. Já em Aoki, Aranha e Kanoh (2015), a meta-heurística Otimização por Enxame de Partículas (PSO) foi implementada com uma transição probabilística baseada na distância de Hamming. O trabalho de Dokeroglu e Sevinc (2021) traz a meta-heurística Otimização Baseada em Ensino-Aprendizagem (TLBO) combinada com uma etapa de Busca Tabu, além de uma versão paralelizada do método.

O trabalho de Sun *et al.* (2021) é um dos mais recentes no problema, e apresenta o Algoritmo de Otimização Multinível Dirigida por Soluções, do inglês *Solution-Driven Multilevel Algorithm* (SDMA), cujos resultados mostraram-se equivalentes ao estado da arte do problema. A execução do método SDMA consiste em duas etapas: inicialmente reduz-se o grafo original a grafos menores, colorindo-os utilizando um mecanismo de refinamento baseado em Busca

Tabu. Em seguida, realiza-se uma projeção da solução para os grafos maiores. Os algoritmos populacionais representam o estado da arte para o GCP, normalmente com técnicas híbridas que utilizam procedimentos de busca local para otimização das soluções(SUN *et al.*, 2021). A Busca Tabu é frequentemente utilizada como mecanismo de refinamento. Entretanto, a implementação de meta-heurísticas de algoritmos populacionais apresenta uma desvantagem em relação a meta-heurísticas que operam em uma única solução, pois, por lidar com muitas soluções a cada iteração, demandam de mais memória.

Apesar de ser um problema extramente relevante, o GCP é um problema fundamentalmente teórico. Entretanto, diversas são as aplicações que podem ser encaradas como derivadas do problema. A seguir, serão apresentadas algumas das principais aplicações práticas do GCP.

2.2.1 APLICAÇÕES DO GCP

Na literatura existem diversos problemas que podem ser modelados como GCP. Nesta Seção são apresentadas as principais delas.

O Problema da Coloração de Mapas é, provavelmente, o problema mais diretamente relacionado ao GCP, visto que a origem deste remonta aquele (CHARTRAND; ZHANG, 2019). A coloração de mapas consiste em colorir um mapa político, de modo que duas regiões fronteiriças não compartilhem a mesma cor.

O Problema de Agendamento consiste em agendar atividades que não possam ser realizadas simultaneamente, como disciplinas diferentes ministradas pelo mesmo professor (PARDALOS; MAVRIDOU; XUE, 1998). A primeira menção a modelar esse problema como um GCP foi apresentada em Werra (1985). No exemplo citado, as disciplinas seriam os vértices do grafo, e caso um par fosse ministrado pelo mesmo professor, uma aresta seria inserida entre elas. Assim, os períodos de cada aula seriam análogos às cores, e o objetivo do problema é minimizar o número de períodos, e/ou encontrar o $\chi(G)$ do grafo em questão (PARDALOS; MAVRIDOU; XUE, 1998).

O Problema da Atribuição de Frequências consiste em atribuir diferentes frequências aos meios de telecomunicações (PARDALOS; MAVRIDOU; XUE, 1998). Um exemplo consiste em um conjunto de estações de rádio que devem receber uma frequência para transmissão de seus sinais, de modo que estações próximas umas as outras não estejam sujeitas à interferências entre si (AHMED, 2012). Neste caso, as estações são análogas aos vértices, a proximidade entre elas implica na existência ou não de uma aresta, e as cores são correspondentes as frequências.

Outros problemas modeláveis como um GCP podem ser agendamento de vôos, solução de Sudoku, número de câmeras necessárias para monitorar um edifício etc. (AHMED, 2012).

Neste trabalho, o Problema da Alocação de Registradores em Compiladores foi abordado como um GCP. Nas seções seguintes serão abordados conceitos referentes ao processo de compilação e de alocação.

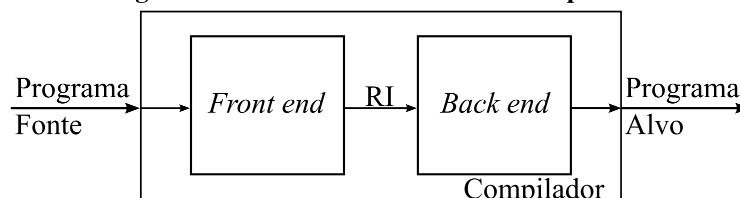
2.3 COMPILADORES

Antes que um programa possa ser executado, deve ser traduzido em um conjunto de operações que estão definidas no computador alvo (COOPER, K.; TORCZON, 2011). Essa tradução pode ser realizada por um *Compilador*.

De acordo com Louden e Flávio Soares Corrêa Silva (2004), compiladores são programas de computador que traduzem de uma linguagem para outra. O compilador é capaz de ler programas em uma linguagem-fonte e traduzi-lo para um programa equivalente em uma linguagem-alvo (AHO *et al.*, 2007), capaz de ser compreendida e executada pelo *hardware*. É comum que a linguagem-fonte seja de alto nível, como Java e C, e a linguagem-alvo seja um código-objeto ou código de máquina para a máquina-alvo (LOUDEN; SILVA, F. S. C., 2004).

O processo de tradução entre linguagens é dividido em duas etapas principais que ocorrem internamente em um compilador: análise e síntese (SANTOS; LANGLOIS, 2018), ou ainda, *front end* e *back end* (COOPER, K.; TORCZON, 2011). Esse processo é ilustrado na Figura 2. Em linhas gerais, o *front end* foca em entender o programa em linguagem-fonte, enquanto o *back end* busca mapear programas para a máquina alvo (COOPER, K.; TORCZON, 2011).

Figura 2 – Estrutura interna de um compilador.



Fonte: Adaptado de (COOPER, K.; TORCZON, 2011).

A análise é a etapa responsável por processar a sequência de entrada e verificar se o programa foi escrito em conformidade com a linguagem-fonte para a qual o compilador foi projetado (SANTOS; LANGLOIS, 2018). Esse processo é realizado aplicando uma estrutura gramatical na entrada, e cria uma representação intermediária (RI) do programa fonte, detectando

possíveis inconformidades sintáticas ou semânticas, alertando ao usuário para que este possa efetuar as correções necessárias (AHO *et al.*, 2007). Além disso, a análise coleta e armazena informações sobre o programa de entrada, em uma estrutura de dados chamada tabela de símbolos, que é passada em conjunto com a RI para a etapa de síntese (AHO *et al.*, 2007). O processo de análise é subdividido em quatro etapas internas: análise léxica, sintática, semântica e geração de código intermediário. Na Figura 2 é possível observar a representação da RI como produto da etapa de *front end*.

A etapa de síntese processa a RI e a tabela de símbolos para produzir um formato de saída, considerando que a etapa de análise ocorreu corretamente (SANTOS; LANGLOIS, 2018). O *back end* deve mapear a RI para o conjunto de instruções e os recursos finitos disponíveis na máquina alvo com cada etapa do processo de síntese levando o programa mais próximo do conjunto de instruções da máquina alvo (COOPER, K.; TORCZON, 2011). Segundo Santos e Langlois (2018), o processo de síntese apresenta três etapas internas: seleção de instruções, escalonamento e reserva de registadores.

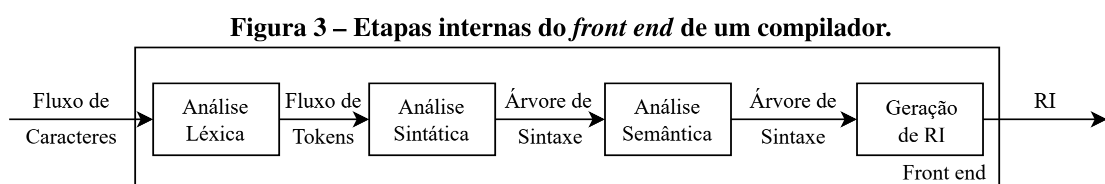
Aho *et al.* (2007) e Keith Cooper e Torczon (2011) consideram ainda uma fase intermediária entre as etapas de *front end* e *back end*: a etapa de otimização. Esta etapa toma como entrada um código em RI e apresenta como saída outro código em RI equivalente, e tem como objetivo melhorar o programa de alguma maneira (COOPER, K.; TORCZON, 2011), aplicando transformações na RI (AHO *et al.*, 2007).

Nas seções a seguir serão abordadas, com maiores detalhes, as etapas intermediárias dos processos de análise e de síntese.

2.3.1 *FRONT END*

Nesta Seção, serão abordadas as etapas internas ao *front end*. A Figura 3 apresenta um diagrama das fases de análise.

Figura 3 – Etapas internas do *front end* de um compilador.



Fonte: Adaptado de (AHO *et al.*, 2007) e (COOPER, K.; TORCZON, 2011).

O **Analisador Léxico** recebe como entrada a sequência de caracteres individuais. que compõe o programa fonte, dividindo-a em uma sequência de elementos léxicos, chamados de

tokens (MOGENSEN, 2017). Aho *et al.* (2007) propõe que cada *token* é da forma <*nomeToken, valor*>, em que *nomeToken* corresponde a um símbolo que identifica o tipo de *token*, enquanto *valor* representa a entrada na tabela de símbolos correspondente ao *token*. É dever do analisador léxico classificar cada palavra entre palavras-chave, identificadores e símbolos especiais suportados pela linguagem (LOUDEN; SILVA, F. S. C., 2004). Ao identificar um erro, deve reportar ao usuário e passar para o próximo *token*. O fluxo de *tokens* é então enviado ao análisador sintático.

O **Analisador Sintático** recebe os *tokens* e utiliza uma gramática para verificar se o programa está sintaticamente correto. Enquanto a análise léxica divide o programa em *tokens*, a sintática os recombina, geralmente na forma de uma árvore de sintaxe, em que as folhas são os *tokens*. Ao ler as folhas da esquerda para a direita, se obtém a mesma sequência da entrada (MOGENSEN, 2017). É responsabilidade desta etapa determinar se o programa que está sendo compilado é sintaticamente válido para a linguagem de programação analisada (LOUDEN; SILVA, F. S. C., 2004). Ao encontrar um erro de sintaxe, deve reportar ao usuário.

O **Analisador Semântico** corresponde a terceira e última etapa do processo de análise. Tem como entrada a árvore de sintaxe e as informações da tabela de símbolos, as utilizando para verificar a consistência entre o programa-fonte e linguagem. É nesta etapa em que os tipos de dados são levados em consideração, pois são relacionados aos *tokens* correspondentes na árvore sintática e na tabela de símbolos (AHO *et al.*, 2007). A partir dessa etapa, é possível proceder com a geração de código intermediário.

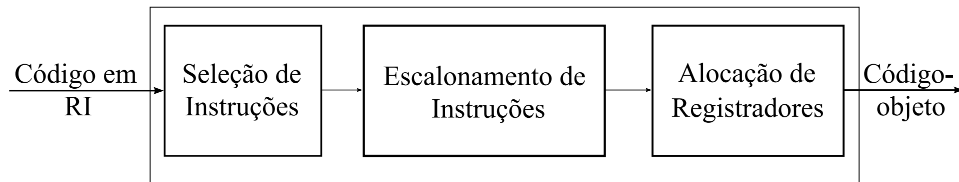
A última etapa do *front end* é a **Geração de Código Intermediário**. No processo de tradução de linguagem-fonte para código-alvo, o compilador pode construir uma sequência de RIs. Esta consiste de uma representação de nível mais baixo que a linguagem-fonte, sendo adequada para tarefas dependentes de máquina, como seleção de instruções e alocação de registradores (AHO *et al.*, 2007). O código em RI pode passar por uma etapa de otimização, a depender do compilador, e em seguida serve de entrada para o *back end*.

2.3.2 *BACK END*

Nesta Seção serão abordadas duas das etapas internas do *back end*: seleção e escalonamento de instruções. Por se tratar de uma das aplicações deste trabalho, a etapa de alocação de registradores será abordada na Seção 2.4. A Figura 4 apresenta um diagrama das etapas internas ao *back end*, para visualização da sequência de etapas.

A primeira etapa é a **Seleção de Instruções** e consiste no processo de escolha de

Figura 4 – Etapas internas do *back end* de um compilador.



Fonte: Adaptado de (AHO *et al.*, 2007) e (COOPER, K.; TORCZON, 2011).

operações de código de máquina que implemente as operações em RI definidas na geração de código intermediário (COOPER, K.; TORCZON, 2011). Para Santos e Langlois (2018), a seleção de instruções é um passo complexo do processo de compilação, pois é necessário conhecer as instruções do processador, e tem como resultado um código *assembly* específico da arquitetura desejada, sendo difícil de verificar e corrigir. O processo de seleção de instruções pode ser uma tarefa muito complexa, a depender do número de instruções disponíveis na arquitetura do processador (AHO *et al.*, 2007). Dentre as principais estratégias utilizadas são a seleção de instruções por casamento de padrões de árvore e por meio da otimização *peephole* (COOPER, K.; TORCZON, 2011).

O **Escalonamento de Instruções** é o processo em que um compilador reordena as operações no código compilado em uma tentativa de reduzir o tempo de execução (COOPER, K.; TORCZON, 2011). Esta etapa, em alguns processadores, pode ser realizada com operações em paralelo, e só pode ser realizado após a seleção de instruções, para saber quais comandos podem ser paralelizados (SANTOS; LANGLOIS, 2018).

A seguir, apresenta-se o problema da alocação de registradores, o qual será um dos tópicos de pesquisa propostos neste trabalho.

2.4 ALOCAÇÃO DE REGISTRADORES

O problema da alocação de registradores refere-se ao mapeamento das variáveis do programa para registradores ou para endereços de memória (PEREIRA, 2008). Na hierarquia de memória, os registradores ocupam o posto de memória de acesso mais rápido, e normalmente são os únicos locais que as operações podem acessar diretamente (COOPER, K.; TORCZON, 2011). O propósito da alocação é mapear um grande número de variáveis em um (tanto quanto) pequeno número de registradores (MOGENSEN, 2017). Variáveis que não podem ser armazenadas em registradores, devem ser temporariamente armazenadas na memória principal, que demanda um maior tempo de acesso, em um processo chamado de derramamento, do inglês *spilling*

(MOGENSEN, 2017). Assim, um uso eficiente dos registradores disponíveis é fundamental ao desempenho do programa a ser compilado.

O processo de *spilling* introduz operações de *load* e *store* no código-alvo para gerenciar o carregamento de variáveis de e para a memória. A alocação de registradores é realizada pelo alocador de registradores, cujo objetivo é reduzir o número de operações de *load* e *store* no código-alvo. Este recebe como entrada o programa, com instruções selecionadas e possivelmente escalonado, e o número de registradores disponíveis, produzindo como saída um programa equivalente que cabe no conjunto de registradores da máquina-alvo (COOPER, K.; TORCZON, 2011).

Sethi (1975) provou que alocar variáveis a registradores de maneira ótima trata-se de um problema NP-Completo. Dessa maneira, muitas técnicas diferentes podem ser utilizadas para abordar o problema, como: algoritmos *greedy*, *top-down*, ordenação de subárvores, próximo uso, pesquisa linear ou coloração de grafos (SANTOS; LANGLOIS, 2018).

Um alocador lida com dois sub-problemas distintos, mas relacionados: a alocação e a atribuição. A alocação é responsável por determinar quais variáveis serão armazenadas nos registradores disponíveis na máquina álvo, enquanto a atribuição mapeia quais variáveis serão destinadas a cada registrador (COOPER, K.; TORCZON, 2011).

É possível confrontar a alocação de registradores nos escopos de alocação local e global. Enquanto a alocação local de registradores atribui registradores a variáveis em blocos básicos de código, a alocação global atribui registradores para variáveis através do programa todo (FARACH-COLTON; LIBERATORE, 2000).

A **alocação local** de registradores lida com conceitos de blocos básicos de código. Considera-se que o programa inicia e termina com o bloco de código, sem herdar valores de blocos executados anteriormente ou passar parâmetros para blocos seguintes (COOPER, K.; TORCZON, 2011). Assume-se que, na fase de alocação, pseudo-registradores são usados para armazenar valores temporários, variáveis locais e constantes frequentemente usadas, e, na etapa de atribuição o alocador de registradores mapeia os pseudo-registradores para registradores reais (LIBERATORE; FARACH-COLTON; KREMER, 1999). Duas abordagens para lidar com a alocação local são apresentadas em Keith Cooper e Torczon (2011): *top-down* e *bottom-up*. Enquanto a primeira realiza-se através da contagem do número de usos de cada variável, varrendo-o do início ao fim, a segunda demanda maior conhecimento do programa e das relações entre as variáveis, pois avança sobre cada bloco e determina, a cada operação se é ou não necessário um *spill*.

Uma métrica importante a ser considerada pelo alocador é a *faixa viva*, ou *live range*. Essa métrica permite saber se duas variáveis podem ou não ocupar o mesmo registrador, em diferentes momentos. Keith Cooper e Torczon (2011) diz que uma variável v está viva em um instante p se foi definida ao longo de um caminho desde o início do procedimento até p, e existe um caminho a partir de p até um uso de v, sem que este seja redefinido. A qualquer momento em que v esteja viva, seu valor deve ser preservado pois pode ser usado na sequência. O intervalo de instantes de tempo em que v está viva é a *faixa viva* (CHAITIN *et al.*, 1981). Caso duas ou mais variáveis estejam vivas simultaneamente, ou seja, se há interseção entre suas faixas vivas, é dito que há *interferência* entre elas (GEORGE; APPEL, 1996).

A **alocação global** torna-se mais complexa que a local, visto que uma variável pode ser usada em diferentes contextos, o que pode gerar diferentes faixas vivas para ela. Além disso, diferentes referências a uma variável podem ser executadas várias vezes, o que torna difícil determinar o custo de *spilling* da variável (COOPER, K.; TORCZON, 2011). Para lidar com o problema das faixas vivas, o alocador pode gerar nomes diferentes para cada intervalo de uso da variável, e para o número de execuções de uma variável é possível que cada referência (ou bloco básico) seja rotulado com uma estimativa de frequência de execução (COOPER, K.; TORCZON, 2011).

Existem diversas abordagens na literatura para lidar com a alocação de registradores. Em (GOODWIN; WILKEN, 1996) tem-se uma abordagem via programação linear inteira. O problema é formulado pelos autores como um *0-1 Integer Linear Programming*, modelando todas as restrições como um conjunto de equações lineares. O algoritmo proposto demonstrou-se um método exato, com uma complexidade prática de $O(n^3)$. Em Kong e Wilken (1998) outra abordagem via *0-1 Integer Linear Programming* foi proposta, para arquiteturas irregulares de processadores. O algoritmo foi testado em processadores x86, apresentando menor sobrecarga que métodos via GCP.

Em Kurdahi e Parker (1987) é apresentado o REAL (sigla para REgister ALlocation), que toma como base algoritmo *Left edge*. As variáveis são ordenadas de acordo com o instante de início de suas faixas vivas. A variável – digamos x cujo intervalo inicia primeiro é então atribuída ao primeiro registrador, seguida da primeira variável cujo início seja imediatamente após o término da faixa viva de x. Esse processo é repetido até que todas as variáveis sejam atribuídas, utilizando um novo registrador quando necessário. Os autores clamam que o resultado é ótimo quando não há *pipeline* nem desvios condicionais.

Poletto e Sarkar (1999) trazem uma abordagem não baseada em GCP, utilizando

heurísticas e *linear scan*. Essa abordagem parte das faixas-vivas das variáveis, analisando-os e alocando-os com um algoritmo guloso.

Koes e Goldstein (2005) apresentaram um alocador de registradores progressivo que usa MCNF (*Multi-Commodity Network Flow*). O algoritmo é dito progressivo pois, gera uma solução inicial como um alocador convencional, mas, se for permitido o uso de tempo computacional suficiente, pode ser melhorada até o resultado ótimo.

O problema de alocação de registradores também pode ser reduzido ao GCP, sendo esta, na verdade, a abordagem mais utilizada na literatura (PEREIRA, 2008), pois se um número adequado de registradores for disponibilizado, uma alocação por coloração de grafos apresenta uma abordagem extremamente direta, reduzindo o peso de algumas restrições do processo de alocação (BRIGGS, 1992). Para isso, algumas considerações são necessárias. A partir do conceito de interferência citado anteriormente, é possível construir um **grafo de interferência**. Este é um grafo não direcionado $G = (V, E)$, em que o conjunto de vértices $V = v_1, v_2, ..., v_n$ representa o conjunto de variáveis do programa, e o conjunto de arestas $E = (e_1, e_2,...,e_m)$ em que uma aresta $e = (v_i, v_j) \in E$ se, e somente se, as variáveis v_i e v_j interferem entre si (CHAITIN *et al.*, 1981). Dessa maneira, as cores atribuídas a cada vértice representam os registradores, e variáveis adjacentes não podem receber a mesma cor.

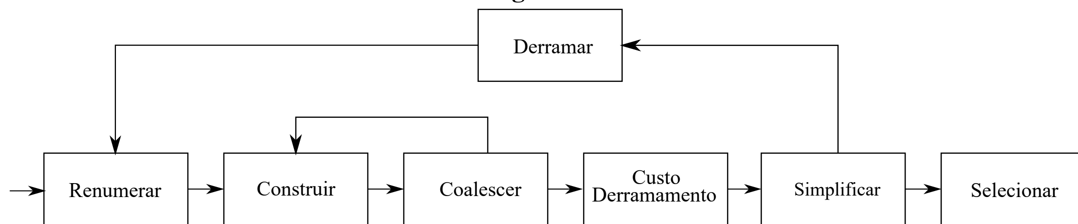
A primeira abordagem de alocação de registradores via GCP foi implementada por Chaitin *et al.* (1981) em uma estrutura de sete fases:

1. Renumerar as faixas vivas, calculando-as para todas as variáveis.
2. Construir o grafo de interferência.
3. Coalescer, ou aglutinar variáveis que compartilhem instruções do tipo $a := b$.
4. Custo de *spilling*, ou seja, uma estimativa do custo para *spill* de cada faixa viva
5. Simplificar o grafo, colorindo-o segundo a heurística da Kempe (1879), em que, ao retirar um vértice com K vizinhos do grafo (sendo K o número de cores, ou registradores), o grafo resultante é colorável com $K - 1$ cores, então o grafo original pode ser colorido com K cores. Nesta etapa, os vértices são removidos, um a um, e armazenados numa pilha.
6. *Spilling*, realizado quando não é possível realizar a simplificação
7. Selecionar, atribuindo cores aos vértices na ordem definida na pilha gerada pela simplificação.

Estas fases estão organizadas, como um fluxograma, na Figura 5.

A partir do trabalho de Chaitin *et al.* (1981), abordagens derivadas foram propostas.

Figura 5 – Diagrama de blocos do algoritmo de Chaitin para alocação de registradores por coloração de grafos.



Fonte: Adaptado de Briggs, Keith D Cooper e Torczon (1994).

Em Briggs, Keith D Cooper, Kennedy *et al.* (1989) a decisão de realizar ou não o processo de *spilling* é realizado após a etapa de seleção, e não de simplificação, o que permitiu reduzir o código de derramamento inserido. Já em Chaitin (1982), a decisão de efetuar *spilling* é baseada no resultado da operação $custo(v)/grau(v)$, em que $custo(v)$ é o custo estimado da operação de derramamento da variável v, e $grau(v)$ é o grau do vértice correspondente no grafo de interferência, escolhendo a variável com menor resultado para a referida operação. Bernstein *et al.* (1989) apresentam uma melhoria nos algoritmos existentes até então, introduzindo a utilização de múltiplas heurísticas para reduzir a ocorrência de *spillings*, em conjunto com uma heurística gulosa para determinar a ordem de coloração.

O trabalho de Shimosaka (2019) sugere contabilizar a quantidade de pontos do programa em que duas variáveis u e v estão vivas, dando ao grafo de interferência informação relativa o peso das arestas. Assim, o peso das arestas é levado em consideração no cálculo das métricas de decisão a respeito do derramamento.

Uma abordagem para a alocação de registradores como um GCP, utilizando *Deep Learning* foi proposta em Das, Ahmad e Venkataramanan Kumar (2020). De modo a poder validar sua abordagem, aplicaram o método a grafos com até 100 vértices, para os quais era possível encontrar uma coloração ótima, de maneiraa exata. Entretanto, uma fase de correção mais tradicional mostrou-se necessária, tornando a abordagem híbrida.

Quando se trata da aplicação de meta-heurísticas ao problema, alguns trabalhos podem ser citados. Em Kri e Feeley (2004) uma abordagem com algoritmos genéticos é proposta, mostrando-se um método interessante para acelerar a geração de código. Já em Lintzmayer, Mulati e Anderson Faustino da Silva (2011), é proposta uma implementação de ACO, denominada CA-RT-RA. O método mostrou-se capaz de reduzir o número de *spills*, além de gerar um código de maior qualidade.

2.5 META-HEURÍSTICAS

Nessa Seção serão abordadas as Meta-heurísticas Busca Tabu e *Simulated Annealing*. O funcionamento de cada meta-heurística e seus respectivos algoritmos básicos serão apresentados com detalhes.

2.5.1 BUSCA TABU

Com origem nos trabalhos de Glover (1986) e Hansen (1986), a Busca Tabu BT é uma meta-heurística amplamente utilizada em problemas de otimização combinatória, sendo uma das técnicas mais utilizadas em hibridizações. A BT trata-se de uma estratégia para resolução de uma grande variedade de problemas, com aplicações que vão desde a teoria de grafos até problemas gerais puros e mistos de programação inteira (GLOVER, 1989).

O princípio básico da BT é realizar Buscas Locais (BL) sempre que encontrar um ótimo local, permitindo movimentos em direção a soluções que não necessariamente sejam melhores que a atual, varrendo todo o **espaço de busca de soluções**. O espaço de busca é o conjunto de todas as soluções que podem ser visitadas durante a busca (GENDREAU; POTVIN, 2005). A ciclagem para soluções previamente visitadas é evitada através da utilização de memórias, chamadas listas tabu, que registram o histórico recente da busca (GENDREAU; POTVIN *et al.*, 2010).

Um dos principais pontos de destaque da BT são as estruturas de memória, classificadas em dois tipos (GLOVER; TAILLARD, 1993): uma memória de curto prazo, chamada de lista tabu, que registra os últimos movimentos que efetivamente melhoraram a solução, os restringindo por T iterações; e uma memória de longo prazo, denominada lista de frequência, cujo objetivo é direcionar a busca para determinadas regiões da vizinhança com base na análise da frequência dos movimentos.

Os movimentos capazes de reverter o efeito de movimentações recentes são denominados **movimentos tabu** (GENDREAU; POTVIN *et al.*, 2010), sendo armazenados na lista tabu. Alguns dos principais parâmetros da BT relacionam-se com os movimentos tabu, como o tamanho da lista tabu, o número de iterações que cada movimento pode ser restringido, ou qual informação será registrada na lista – se a solução toda, ou se apenas o movimento (GENDREAU; POTVIN *et al.*, 2010). A definição desses parâmetros é crucial no desempenho do algoritmo, cabendo ao projetista determinar seus valores com cuidado.

A Busca Local realizada pela BT é também uma etapa crucial no método, sendo efetivamente responsável pela descoberta de novas soluções ao explorar a vizinhança. Sua execução é controlada a partir dos critérios de **restrição** e de **aspiração**, que permitem ou não a escolha de uma solução vizinha. A restrição é implementada através da já citada lista tabu. O critério de aspiração é responsável por permitir que alguns movimentos da lista tabu possam ser realizados, quando não apresentarem o perigo de ciclagem, ou quando houver risco de estagnação do processo de busca (GENDREAU; POTVIN, 2005), revogando o status de movimento tabu.

A seguir, apresenta-se o algoritmo básico da BT, adaptado do trabalho de Glover (1989). Para tal, é necessário introduzir algumas notações.

- S, a solução corrente,
- S^*, a melhor solução conhecida,
- f^*, o valor de avaliação de S^*,
- $N(S)$,a vizinhança de S,
- $\tilde{N}(S)$ o subconjunto de $N(S)$, que representa as soluções admissíveis (i.e., não-tabu ou permitidas por aspiração)
- T, a lista tabu.

O Algoritmo 1 apresenta o funcionamento básico da BT.

Algoritmo 1 – Algoritmo para Busca Tabu Simples

Requer: Uma solução inicial S_0
 $\{Inicialização\}$
1 $S \leftarrow S_0$
2 $f^* \leftarrow f(S_0)$
3 $S^* \leftarrow S_0$
4 $T \leftarrow \emptyset$ $\{Busca\}$
5 **repete**
6 Escolha S em $Ótimo(\tilde{N}(S))$
7 **se** $f(S) < f^*$ **então**
8   $f^* \leftarrow f(S)$, $S^* \leftarrow S$
9 **finaliza se**
10 Registre o tabu do movimento corrente em T
11 Se necessario, remova de T a entrada mais antiga
12 **até** condição de parada seja satisfeita
13 **retorna** A melhor solução encontrada S^*

Fonte: Adaptado de Glover (1989).

A linha 1 apresenta as inicializações, em que a solução corrente, e a melhor conhecida são inicializadas com a solução inicial S_0, que pode ser obtida com uma heurística de construção como o algoritmo guloso. O *loop while* representa a etapa de busca por uma solução na vizinhança. A função *Ótimo()* na linha 3, implementa a busca local, realizada em $\tilde{N}(S)$, considerando a lista tabu. A depender do critério de aspiração implementado, alguns movimentos que levem a

soluções $S \notin \tilde{N}(S)$ podem ser permitidos. Nas linhas 4 e 5 é realizada a atualização da melhor solução, caso S se mostre melhor que S^*. A atualização da lista tabu é realizada nas listas 6 e 7.

Conforme exposto na Seção 2.2, a BT é frequentemente empregada em estratégias de solução para o GCP, normalmente em técnicas híbridas, sendo utilizada como um tipo de método de refinamento de outras meta-heurísticas. Nesse trabalho, a Busca Tabu clássica foi implementada para o GCP, bem como foi aplicada ao problema da alocação de registradores.

2.5.2 *SIMULATED ANNEALING*

Proposto inicialmente em Kirkpatrick, Gelatt e Vecchi (1983), o *Simulated Annealing* (SA) trata-se de uma técnica de otimização combinatória cujo funcionamento está pautado em uma analogia com o processo de recozimento de materiais – sendo também conhecido como Têmpera Simulada. De acordo com Gendreau, Potvin *et al.* (2010), o processo de recozimento consiste em aquecer, até uma temperatura inicial T, um sólido cristalino e depois resfriá-lo lentamente para que atinja a configuração de partículas mais regular possível, resultando em um cristal com integridade estrutural superior. Uma analogia a esse processo pode ser aplicado na geração de soluções para problemas de otimização (VAN LAARHOVEN; AARTS, 1987). Por sua simplicidade de implementação e eficiência, é uma das meta-heurísticas mais utilizadas em problemas de otimização (TALBI, 2009).

De acordo com Talbi (2009) a qualidade da solução depende da temperatura inicial T, pois temperaturas não suficientemente altas, podem levar a soluções não tão eficientes. Em analogia ao processo de recozimento, temperaturas iniciais baixas deixam os cristais fragilizados.

A cada iteração do SA aplicado a um problema de otimização, os valores de duas soluções (a atual e uma nova solução selecionada com os critérios de busca) são comparadas. A melhora de soluções é sempre aceita, enquanto soluções piores são aceitas de acordo com uma taxa probabilística, com o objetivo de escapar de ótimos locais (GENDREAU; POTVIN *et al.*, 2010).

A probabilidade do critério de aceitação segue a distribuição de Boltzmann (VAN LAARHOVEN; AARTS, 1987), dada por:

$$P(\Delta E, T) = \mathrm{e}^{\frac{-\Delta E}{T}} \tag{1}$$

em que ΔE representa a diferença em energia (valor ou qualidade) entre a solução corrente, e a solução gerada na iteração, e T é a temperatura atual do sistema.

O funcionamento geral do método será explicado a partir do Algoritmo 2, adaptado de (TALBI, 2009). Antes de apresentar o algoritmo, algumas notações são necessárias:

- T_{max}, a temperatura inicial do sistema,
- T, temperatura atual do sistema,
- f, o valor da energia de cada solução,
- ΔE, a diferença de energia entre as soluções,
- S, solução atual,
- S', solução gerada na iteração atual,
- S^*, melhor solução encontrada,
- g, função de atualização (redução) da temperatura.

Algoritmo 2 – Algoritmo do Simulated Annealing

Requer: Uma solução inicial S_0, Uma temperatura inicial T_{max}
{Inicialização}
1 $S \leftarrow S_0$
2 $T \leftarrow T_{max}$
3 **repete**
4   **repete**
5   Gerar uma solução S' segundo critério estabelecido
6     $\Delta E = f(S') - f(S)$
7   **se** $\Delta E \leq 0$ **então**
8       $S \leftarrow S'$ {Aceita a solução gerada}
9   **senão**
10   Aceita S' com probabilidade $e^{\frac{-\Delta E}{T}}$
11   **finaliza se**
12   **se** $f(S) < f(S^*)$ **então**
13       $S^* \leftarrow S$
14   **finaliza se**
15   **até** Condição de equilíbrio alcançada {e.g um dado número de iterações executadas a cada temperatura T}
16   $T = g(T)$ {Atualização de temperatura}
17 **até** Critério de parada seja satisfeito /*e.g. $T < T_{min}$*/
18 **retorna** A melhor solução encontrada S^*

Fonte: Adaptado de Talbi (2009).

A entrada do algoritmo é composta por uma solução inicial S_0, que pode ser construída com heurísticas de construção – como o método guloso–, e a temperatura máxima do sistema T_{max}. A cada iteração do SA, uma nova solução é gerada de acordo com algum critério, como a aleatoriedade, conforme a linha 4. A energia ΔE é calculada na linha 5 como a diferença entre o valor da solução corrente $f(S)$ e a nova solução gerada $f(S')$.

A estrutura condicional das linhas 6 a 8 determinam se a solução gerada é ou não aceita. Caso haja melhora, a solução gerada é aceita. Caso a nova solução seja pior que a seleção corrente, a solução é aceita com probabilidade seguindo a distribuição de Boltzmann (Equação 1). Além disso, se a solução gerada tiver energia menor que a melhor conhecida, atualiza-se a melhor

solução conhecida (linhas 10 e 11).

Na linha 12 é realizada a atualização do valor da temperatura, o que ocorre de acordo com o esquema de resfriamento definido na função $g(T)$, que é reduzida até que se atinja o critério de parada do algoritmo, que pode ser uma temperatura T_{min} definida a *priori*.

O SA é frequentemente empregado em problemas de otimização, sendo uma das primeiras técnicas utilizadas para resolução do GCP, conforme exposto na Seção 2.2. Por apresentar familiaridade com problemas de grafos, o SA será implementado neste trabalho, para abordar o GCP, bem como o problema da alocação de registradores.

3 DETALHES DE IMPLEMENTAÇÃO E ORGANIZAÇÃO DOS EXPERIMENTOS

Neste capítulo são discutidos os aspectos práticos que mostraram-se relevantes no desenvolvimento do trabalho. Na Seção 3.1 são apresentadas as instâncias utilizadas nos experimentos computacionais. Na Seção 3.2 são discutidos os detalhes relativos à implementação realizada, desde aspectos como representação dos grafos e de soluções, funções auxiliares e comuns a ambas as meta-heurísticas, até especificações das meta-heurísticas implementadas. Na Seção 3.3 se discute as especificações que diferenciam a implementação das abordagens ao GCP e à Alocação de Registradores. Por fim, a Seção 3.4 apresenta as definições dos parâmetros utilizados, e a organização dos experimentos realizados.

3.1 INSTÂNCIAS DE TESTE

No escopo deste trabalho, aplicaram-se as meta-heurísticas Busca Tabu e *Simulated Annealing* ao GCP em duas frentes de trabalho. Inicialmente, o objeto de pesquisa foi o GCP na forma de um problema de decisão, com o objetivo de verificar se um grafo é k-colorável para k cores. Em seguida, para verificar a viabilidade dos métodos na alocação de registradores, avaliaram-se grafos relativos a interferência entre variáveis em programas reais. Assim, o problema foi encarado com o objetivo de otimização, visando reduzir o número de *spill*s das variáveis para a memória.

O processo real da alocação de registradores, conforme descrito na Seção 2.4, é complexo e passa por várias etapas internas, além de todo o processo de compilação que ocorre previamente à alocação (ver Seção 2.3). Como o foco do trabalho é estudar o uso de meta-heurísticas no GCP, definiu-se que o processo de alocação fosse abordado assumindo-se que o grafo de interferência do programa estivesse construído, ou seja, partindo da segunda fase proposta por Chaitin *et al.* (1981).

De modo a testar e validar os métodos a serem implementados, é necessária a utilização de uma base de testes já consolidada na literatura. Assim, foram utilizados os casos de teste disponíveis na base DIMACS (JOHNSON; TRICK, 1996). Esta base consiste em diversas instâncias de grafos utilizados em muitos trabalhos da literatura, dentre os quais – mas não exclusivamente, Johnson, Aragon *et al.* (1991), Qu e Potkonjak (1998) e Galinier e Hao (1999).

Os grupos de instâncias escolhidas para este trabalho podem ser classificadas em:

- DSJ: grafos aleatórios abordados em Johnson, Aragon *et al.* (1991);

- LEI: grafos de Leighton com tamanho garantido de coloração (LEIGHTON, 1979);
- MYC: grafos baseados na transformação de Mycielski (FAN, 2004);
- CAR: grafos de k-Inserção (*k-Insertion*) e de Inserção Completa (*Full Insertion*), que tratam-se de uma generalização dos grafos do grupo MYC, com vértices inseridos para aumentar o tamanho dos graphos, mas manter sua densidade.
- SGB: grafos da Base de Grafos de Stanford, de Donald Knuth (*Stanford GraphBase*) (KNUTH, 1993);
- HOS: grafos obtido de uma abordagem via de particionamento de matrizes em colunas segmentadas ao problema da determinação de matrizes Jacobianas esparsas;
- REG: grafos baseados em códigos reais de um problema de alocação de registradores (LEWANDOWSKI; CONDON, 1996);

Na abordagem do GCP, foram considerados os grafos dos grupos DSJ, LEI, SGB, MYC e CAR, para os quais foi possível encontrar, na literatura, um número cromático χ, ou um número mínimo de cores conhecido k^*. Estes dados, bem como o número de vértices de cada grafo utilizado, podem ser conferidos na Tabela 1.

Para o problema da alocação de registradores, abordou-se o conjunto de instâncias REG. Este trata-se de grafos relativos a programas reais. Os grafos propostos em Lewandowski e Condon (1996) para a alocação de registradores variam em tamanho de 100 a 850 vértices, para quatro diferentes programas base, e foram construídos para compiladores com 32 registradores. No contexto deste trabalho, serão avaliados os grafos dos sub-grupos mulsol e zeroin, cujo número de vértices varia de 100 a 200. Na Tabela 2 são apresentadas as instâncias utilizadas nos experimentos da alocação de registradores. Nela, é possível observar o número de vértices dos grafos, simbolizado por $|V|$, e o menor número de cores (registradores) conhecido na literatura que torna possível colorir os grafos de maneira ótima, simbolizado por k^*.

O número de vértices foi limitado a 200, por questões práticas, como o *hardware* disponível. Houveram tentativas de utilizar os métodos implementados para instâncias com mais de 300 vértices, mas o tempo de execução mostrou-se inviável para condução dos experimentos.

No contexto deste trabalho, estes grafos foram avaliados no contexto de processadores com 8 ou 12 registradores. Estes números de registradores são consideravelmente menores que os k^* necessários para alocar os programas de maneira ótima, o que potencializa a necessidade de realizar *spillings* para que não hajam conflitos entre as variáveis do programa. Apesar de existirem processadores com maior número de registradores, limitou-se os experimentos a 12, pois, para os grafos utilizados, números maiores reduziriam (ou tornariam desnecessário) o

Tabela 1 – Apresentação dos parâmetros dos grafos utilizados no GCP

Instância	|V|	χ ou k^*	Grupo	Instância	|V|	χ ou $k*$	Grupo
DSJC1000.1	1000	20	DSJ	2-FullIns_5	852	7	CAR
DSJC250.1	250	8	DSJ	2-Insertions_5	597	6	CAR
DSJC250.5	250	28	DSJ	3-FullIns_4	405	7	CAR
DSJC500.1	500	12	DSJ	4-FullIns_4	690	8	CAR
DSJC500.5	500	48	DSJ	4-Insertions_4	475	5	CAR
DSJR500.1	500	12	DSJ	anna	138	11	SGB
DSJR500.1c	500	85	DSJ	david	87	11	SGB
DSJR500.5	500	122	DSJ	homer	561	13	SGB
le450_5a	450	5	LEI	huck	74	11	SGB
le450_5b	450	5	LEI	jean	80	10	SGB
le450_5c	450	5	LEI	miles250	128	8	SGB
le450_5d	450	5	LEI	miles500	128	20	SGB
le450_15a	450	15	LEI	miles750	128	31	SGB
le450_15b	450	15	LEI	miles1000	128	42	SGB
le450_15c	450	15	LEI	miles1500	128	73	SGB
le450_15d	450	15	LEI	queen5_5	25	5	SGB
le450_25a	450	25	LEI	queen6_6	36	7	SGB
le450_25b	450	25	LEI	queen7_7	49	7	SGB
le450_25c	450	25	LEI	queen8_8	64	9	SGB
le450_25d	450	25	LEI	queen8_12	96	12	SGB
myciel3	11	4	MYC	queen9_9	81	10	SGB
myciel4	23	5	MYC	queen11_11	121	11	SGB
myciel5	47	6	MYC	queen13_13	169	13	SGB
myciel6	95	7	MYC	ash331GPIA	662	4	HOS
myciel7	191	8	MYC	ash608GPIA	1216	4	HOS
1-Insertions_6	607	7	CAR				

Tabela 2 – Apresentação dos parâmetros dos grafos utilizados no problema da alocação de registradores

Instância	|V|	χ ou $k*$
mulsol.i.1	197	49
mulsol.i.2	188	31
mulsol.i.3	184	31
mulsol.i.4	185	31
mulsol.i.5	186	31
zeroin.i.1	211	49
zeroin.i.2	211	30
zeroin.i.3	206	30

número de *spillings*.

3.2 DETALHES DE IMPLEMENTAÇÃO

Nesta seção, são discutidos os detalhes de implementação. Inicialmente se discutem os detalhes gerais, como representação de grafos e soluções, métodos de geração de população inicial e mecanismos de exploração de vizinhança. Em seguida, a implementação de cada

meta-heurística é discutida com mais detalhes.

De modo a tornar possível a aplicação das meta-heurísticas, é preciso definir uma estrutura de representação dos grafos, além de uma representação das soluções. Neste trabalho, o grafo $G = (V,E)$, conforme definido na Seção 2.2, foi implementado na forma de uma matriz e inteiros $D_{n \times n} = (d_{ij})$, em que $n = |V|$, ou seja, representa o número de vértices do grafo, e d_{ij} representa a presença ($d_{ij} = 1$) ou ausência ($d_{ij} = 0$) de uma aresta entre os vértices i e j. Escolheu-se a representação matricial, devido a facilidade de acessar as posições, além de não ser necessário alocar e realocar memória o tempo todo. Escolheu-se a forma de um vetor de inteiros $S = [s_0, s_1, ..., s_n]$ para representar uma solução. Neste vetor, cada posição $s_i, 0 < i \leq n$ é relativa a um vértice do grafo G, com cada um recebendo um valor no intervalo $[0, k - 1]$, em que k representa o número de cores disponíveis para coloração.

A qualidade de uma solução, ou seja, sua *fitness*, é medida de acordo com o problema abordado. No caso do GCP, a *fitness* é dada pela contagem do número de conflitos q, que corresponde ao número de arestas que liga vértices coloridos com a mesma cor. É importante ressaltar que, como os grafos abordados são todos não direcionados, tem-se que $e_k = (v_i, v_j) = (v_j, v_i)$. O objetivo final no GCP é obter $q(G) = 0$, para um grafo G.

Para o caso do problema da alocação de registradores, a qualidade dos métodos foi avaliada por meio do número de *spillings* necessário para que o número de conflitos seja zerado, visando minimizar este número.

Conforme exposto na Seção 2.5, ambos os métodos implementados requerem uma solução inicial S_0, a partir da qual são aplicadas as técnicas de minimização de conflitos e/ou *spillings*. Neste trabalho, tanto para o GCP quanto para a alocação, foram implementados dois mecanismos de geração de solução inicial: aleatória (*random*) e gulosa (*greedy*), que serão apresentados na sequência.

O Algoritmo 3 apresenta o funcionamento básico da geração de solução inicial aleatória.

Algoritmo 3 – Geração de solução aleatória

Requer: Um grafo $G = (V,E)$, Um número de cores disponíveis k
  {*Inicialização*}
1 $S_0 \leftarrow \{\}$
  {*Construção*}
2 **para** i até |V| **faz**
3   $S_0[s_i] \leftarrow$ cor_aleatoria $\in [0, k - 1]$
4 **finaliza para**
5 calcula_fitness(S_0, G)
6 **retorna** Uma solução S_0

Fonte: Autoria Própria.

Na linha 1 uma solução S_0 é inicializada, com a alocação de memória para um vetor com $n = |V|$ posições. Em seguida, as posições correspondentes aos vértices são preenchidas por sorteio dentre o intervalo numérico, relativo às cores disponíveis. Por fim, a *fitness* da solução é calculada, contando o número de conflitos entre as cores dos vértices adjacentes no grafo G. O objetivo de gerar uma solução inicial de maneira aleatória, é obter um ponto de partida em um período reduzido de tempo, quando confrontado com o método guloso.

O Algoritmo 4 apresenta o funcionamento básico o método guloso implementado. Este tem por objetivo construir uma solução inicial, de maneira iterativa, escolhendo a opção que minimize o número de conflitos a cada rodada.

Algoritmo 4 – Método Guloso

Requer: Um grafo $G = (V,E)$, Um número de cores disponíveis k
{*Inicialização*}
1 $S_0 \leftarrow \{\}$
2 $S_0[0] = 1$
3 **para** s_i, $i \in [1, |V| - 1]$ **faz**
4 **para** cor_j, $j \in [1, k]$ **faz**
5 array_conflitos[j] = contar_conflitos_vertices_coloridos(G, S_0[0:i])
6 **finaliza para**
7 s_i = cor_menor_conflitos(array_conflitos)
8 $S_0[0] = s_i$
9 **finaliza para**
10 calcula_fitness(S_0, G)
11 **retorna** Uma solução S_0

Fonte: Autoria Própria.

Após a linha 1 inicializar uma solução S_0, reservando memória para um vetor com $n = |V|$ posições, a primeira posição da solução S_0, correspondente ao vértice 0, é colorida com a primeira cor disponível. A partir da linha 3, a construção da solução é realizada de maneira iterativa, com cada vértice i recebendo a cor s_i que gera menos conflitos com os vértices já coloridos. O laço iniciado na linha de repetição iniciado na linha 4 é responsável por contabilizar o número de conflitos gerados pela coloração com cada cor disponível. As linhas 6 e 7 são responsáveis por identificar a cor s_i que insere menos conflitos, e atribuí-la à posição correspondente ao vértice i. Por fim, a *fitness* da solução S_0 é calculada.

Além dos dois mecanismos de geração de solução inicial, implementou-se um método de procura na vizinhança das soluções, que foi compartilhado por ambas as meta-heurísticas. Este método, doravante denominado *movimento* (ver Algoritmo 5), consiste no sorteio de uma cor, dentre todas as disponíveis, e de um dos vértices do grafo G.

Na sequência, são discutidos os detalhes da implementação das meta-heurísticas Busca Tabu e *Simulated Annealing*.

Algoritmo 5 – Mecanismo de movimentação

Requer: Uma solução S, um grafo G, um número de cores k
1 $num_index \leftarrow sorteio_indices(G)$
2 $S_m \leftarrow S$
3 **repete**
4 $\quad S_m[num_index] \leftarrow sortear_cor(k)$
5 **até** $S_m! = S$
6 calcular_fitness(S_m, G)
7 **retorna** Uma solução advinda de uma movimentação S_m

Fonte: Autoria Própria.

3.2.1 IMPLEMENTAÇÃO DA BUSCA TABU

A implementação da Busca Tabu seguiu como base o Algoritmo 1, apresentado na Seção 2.5.1.

A estrutura de dados utilizada para representação da Lista Tabu, consiste em uma lista encadeada, com cada elemento $node_i$ da lista, formado pela seguinte estrutura:

$$node_i = \{ptr_node_{i+1}, count_iter, index, undo\} \quad (2)$$

em que ptr_node_{i+1} representa um ponteiro para o próximo nó da lista, $count_iter$ é o número de iterações restantes pelo qual o movimento em questão está restrito, $index$ é o índice do vértice que foi modificado no processo de movimento (Algoritmo 5) e, por fim, $undo$ é a cor que, ao ser atribuída ao vértice $index$, reverte a movimentação realizada.

Como entrada do método utilizou-se um parâmetro T, relativo ao número de iterações de permanência na lista tabu, e uma solução inicial – gerada ou de maneira aleatória, conforme o Algoritmo 3, ou de maneira gulosa, como o Algoritmo 4.

Quanto ao procedimento de busca tabu, cada iteração do *loop* iniciado na linha 2 do Algoritmo 1 foi implementada como segue: A linha 3, a escolha da possível nova solução da iteração corrente, consiste na geração de um número $ns = 0.1 * |V|$ de soluções temporárias (ou $ns = 5$, caso $|V| < 100$). Dentre estas soluções, se escolhe como movimento a ser seguido a que apresenta o menor número de conflitos. Para atualizar a solução corrente, verifica-se se o movimento selecionado é, ou não, tabu. Se sim, aceita-se o movimento, e este é inserido na lista tabu, ficando restrito por T iterações. Caso o movimento seja tabu, mas apresente uma melhora em relação a solução corrente, o movimento é aceito e seu registro na lista tabu tem o valor T atribuído ao $count_iter$ (linha 6). É a partir da solução corrente que, na próxima iteração, são aplicados os processos de movimentação para explorar a vizinhaça. Em seguida, todos os demais elementos da lista tabu têm seus atributos $count_iter$ decrementados em uma unidade (linha 7).

Finalmente, verifica-se se a solução corrente apresenta menos conflitos que a melhor solução encontrada até o momento. Caso positivo, a solução corrente passa a ser a melhor (linhas 4 e 5).

Todo esse procedimento é repetido até que o número de conflitos chegue a zero, ou que um determinado número de iterações tenha ocorrido. Ao final da execução da busca tabu, retorna-se a melhor solução encontrada.

3.2.2 IMPLEMENTAÇÃO DO *SIMULATED ANNEALING*

O processo de implementação da meta-heurística *Simulated Annealing* foi baseado no Algoritmo 2 apresentado na Seção 2.5.2. Como entrada do método utilizou-se um parâmetro T_max, relativa a temperatura inicial do processo de têmpera, e uma solução inicial – gerada ou de maneira aleatória ou gulosa.

O SA é pautado em dois laços de repetição, um interno e outro mais externo. O laço interno, iniciado na linha 3 do Algoritmo 2, é responsável por gerar novas soluções a serem avaliadas, enquanto o laço externo define a taxa base de probabilidade para aceite de uma solução.

O processo de geração de uma solução S' (linha 4) é realizado por meio do sistema de movimentação apresentado na Seção 3.2. Normalmente, cada iteração do SA gera uma nova solução de maneira aleatória. Entretanto, neste trabalho, optou-se pela utilização do mesmo sistema de movimentação da busca tabu, para avaliar o mesmo sistema de vizinhança quando abordado por diferentes meta-heurísticas.

O restante da implementação do SA segue rigorosamente o Algoritmo 2, com os seguintes detalhes:

- Ao fim de cada iteração, de ambos os laços interno e externo, verifica-se a solução corrente, ou a melhor encontrada atingiram o número de conflitos igual a zero;
- A função de atualização de temperatura $g(T)$, da linha 12, foi implementada na forma $T = \alpha * T$, em que o parâmetro de decaimento da temperatura escolhido foi $\alpha = 0{,}9$.

Todo esse procedimento é repetido até que o número de conflitos chegue a zero, ou que a temperatura T fique abaixo de um valor mínimo. Ao final da execução do SA, retorna-se a melhor solução encontrada.

3.3 DIFERENÇAS ENTRE A IMPLEMENTAÇÃO DO GCP E DA ALOCAÇÃO DE REGISTRADORES

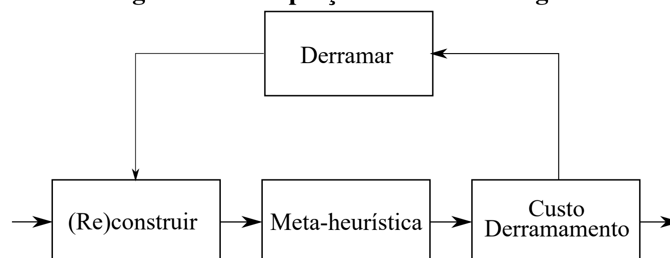
Conforme exposto na Seção 3.2, a métrica de avaliação da *fitness* de cada solução para o problema, é o número de *spillings*, ou seja, o número de variáveis que precisam ser armazenadas diretamente na memória.

As meta-heurísticas implementadas têm como objetivo direto minimizar o número de conflitos entre os vértices, e para o problema da alocação de registradores, é preciso adaptar a utilização das meta-heurísticas. Esta adaptação toma como base a abordagem de Chaitin *et al.* (1981), apresentada na Figura 5.

Como neste trabalho se propôs partir dos grafos já construídos, ou seja, da fase 2 do algoritmo, a etapa de coalescer variáveis não foi abordada, visto que não há um código-fonte para verificar sua viabilidade. As demais etapas foram adaptadas para inclusão das meta-heurísticas no processo de coloração.

Partindo de um grafo, aplicou-se a meta-heurística desejada, considerando-se o número de registradores disponíveis como o número k de cores. Após obter-se uma solução, é realizado o cálculo do número de conflitos por variável (vértice do grafo), o que compreende o cálculo do custo de *spilling*. A variável que tiver o maior número de conflitos com faixas-vivas de outras, é escolhida para ser derramada para a memória. Quando uma variável é marcada para o processo de *spilling*, inicia-se o processo de renumerar as faixas-vivas e de reconstrução do grafo de adjacências, removendo do grafo o vértice correspondente à variável, bem como todas as suas arestas. Com um novo grafo, todo o processo é repetido, até que o número de conflitos seja zerado. A cada variável derramada para a memória, o número de *spillings* é incrementado. Todo esse procedimento é ilustrado no fluxograma da Figura 6.

Figura 6 – Fluxograma da adaptação realizada ao algoritmo de Chaitin



Fonte: Autoria própria.

A cada iteração, uma nova solução inicial é necessária para dar partida no processo de busca das meta-heurísticas. Dessa maneira, ou o método aleatório (Algoritmo 3) ou o

método guloso (Algoritmo 4) são executados novamente, a depender da configuração inicial do experimento.

3.4 DEFINIÇÃO DOS EXPERIMENTOS E PARÂMETROS UTILIZADOS

Nesta seção são definidos os parâmetros utilizados em cada meta-heurística na execução dos experimentos, além da organização destes.

Conforme exposto nas Seções 2.5 e 3.2, as meta-heurísticas implementadas demandam de alguns parâmetros para serem executadas: a Busca Tabu precisa do número T de iterações em que um movimento permanece na lista tabu, e o SA precisa de um valor T_max referente a temperatura inicial do processo de têmpera. Para a Busca Tabu, foi adotado $T = \{10, 25, 50, 100\}$ pois, permitem observar diferentes períodos de tempo de restrição de movimentos, o que tem grande impacto na prevenção, ou permissão, de ciclos durante as buscas. Para o SA foi escolhido $T_max = \{100, 1000, 5000, 10000\}$, pois, apesar de o número de rodadas de decaimento permanecer fixo, essas faixas de valores apresentam diferentes graus de aceitabilidade de soluções ruins, a medida que a temperatura inicial aumenta (TALBI, 2009). Além disso, o número de cores utilizado para coloração, é o χ/k^* apresentado na Tabela 1, que corresponde ao número cromático, ou mínimo conhecido de cada grafo.

Cada instância do GCP foi submetida as duas meta-heurísticas. Cada meta-heurística foi executada de acordo com uma combinação dos fatores *solução inicial* e *parâmetro da meta-heurística*. Por exemplo, a Busca Tabu foi executada com a solução inicial aleatória com cada um dos valores de T, e em seguida foi executada com a solução inicial gulosa para cada um dos valores de T. O mesmo raciocínio é válido para o SA e T_max. Cada combinação de meta-heurística, solução inicial e parâmetros foi executada 20 vezes para cada instância de teste, de modo que seja possível estudar o comportamento de cada uma.

Essas combinações também foram aplicadas às instâncias de teste do problema da alocação de registradores, inserindo essas combinações no fluxo apresentado na Seção 3.3.

Quanto ao ambiente de execução dos experimentos, foi utilizada uma máquina com processador Intel® Core™ i5-7200U 2.5 GHz, 8 GB de RAM, sistema operacional linux Ubuntu 20.04 LTS. Para implementação das meta-heurísticas, escolheu-se a linguagem de programação C (Compilador GCC 9.3.0). Para análise dos resultados foi utilizada a linguagem Python 3.9.7, com auxílio das bibliotecas: pandas (TEAM, 2020; MCKINNEY, 2010) para lidar com os arquivos de saída dos experimentos; seaborn (WASKOM, 2021) e Matplotlib (HUNTER, 2007) para geração

e visualização de gráficos.

Na sequência, são apresentados os resultados e algumas análises a respeito deles.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Neste capítulo serão apresentados os resultados obtidos nos experimentos realizados conforme descritos no Capítulo 3. A exposição e análise dos resultados foi organizada da seguinte maneira: inicialmente, são apresentados os resultados das meta-heurísticas aplicadas ao GCP, com comparação do número de conflitos obtidos por cada método de construção de solução inicial, em cada meta-heurística, filtrados pelos parâmetros definidos na Seção 3.4. Em seguida, são apresentados os resultados obtidos na alocação de registradores, para ambas as meta-heurísticas, seguindo os mesmos critérios estabelecidos para o GCP, com a distinção de que os resultados avaliam o número de *spillings* e consideram um número fixo de registradores, conforme estabelecido na Seção 3.1.

4.1 RESULTADOS DA COLORAÇÃO DE GRAFOS

Nesta seção, são apresentados os resultados da aplicação das meta-heurísticas ao GCP. Os resultados de cada instância são categorizados de acordo com a solução inicial utilizada, com *greedy* correspondendo ao método guloso e *random* sendo referente a geração aleatória de solução.

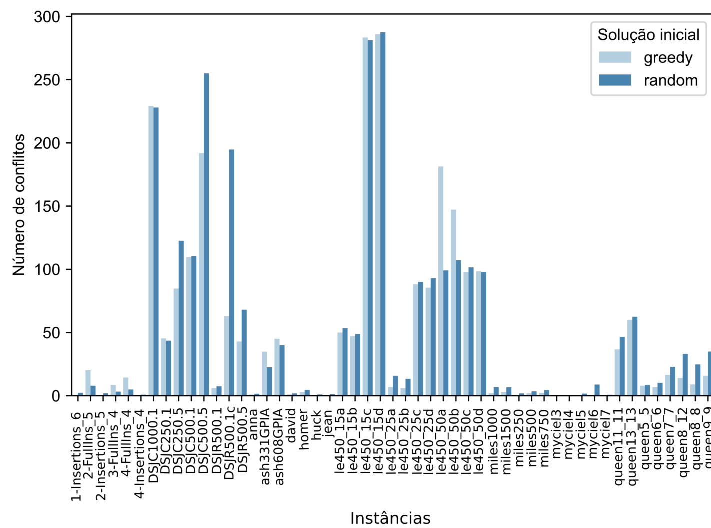
Inicialmente, são expostos os resultados da Busca Tabu, com cada valor de T sendo analisado individualmente. Em seguida, o SA também é apresentado, separando os resultados de acordo com o valor de T_max utilizado.

Além disso, se ressalta que no cenário ideal, não haveriam conflitos em nenhuma das instâncias, visto que o parâmetro k número de cores disponíveis para coloração, ou é o número cromático χ, ou o número mínimo conhecido para colorir o grafo sem conflitos.

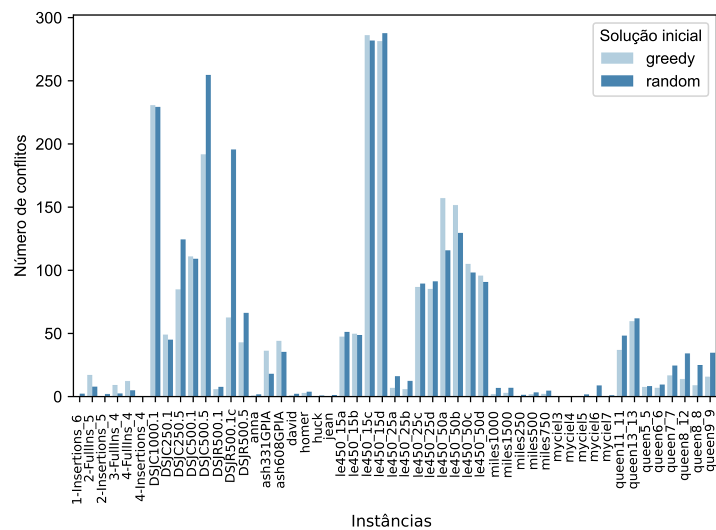
A Figura 7 apresenta o número de conflitos médio obtido pela Busca Tabu em cada instância, considerando o número de iterações de permanência na lista tabu.

É possível observar que para a maioria dos casos de teste, a versão com solução gulosa apresenta menor número de conflitos. Além disso, ao analisar os resultados de acordo com os grupos de instâncias expostos na Tabela 1 da Seção 3.1, é possível observar que as instâncias do grupo DSJ e LEI são as que, em números absolutos de conflitos, apresentam o maior erro, ou distância em relação a coloração ótima, enquanto os grafos dos grupos CAR, SGB e MYC apresentam os menores números de conflitos, com destaque para MYC que foi colorida de maneira ótima.

Figura 7 – Resultados da Busca Tabu para o GCP



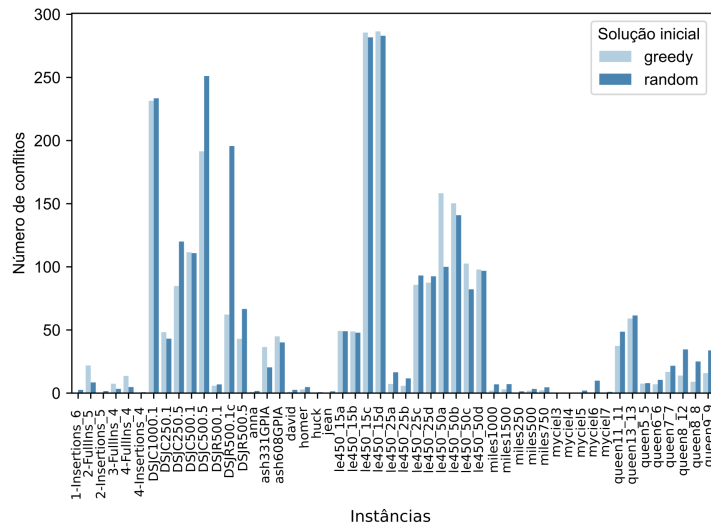
(a) T = 10 iterações



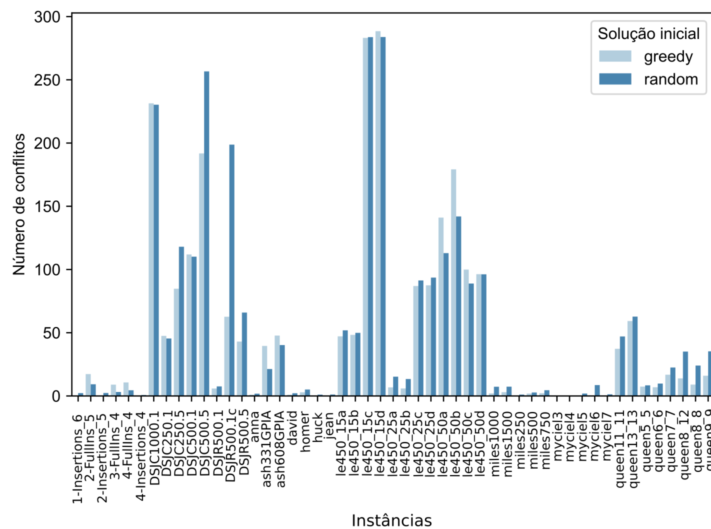
(b) T = 25 iterações

Outro ponto a considerar, é que o comportamento médio do número de conflitos permanece o mesmo para os diferentes valores de T.

Para analisar o tempo médio de execução da Busca Tabu, foi escolhido analisar os resultados para a instância *le450_15c*, por ser uma das que apresentou o maior número de conflitos. A Figura 8 apresenta o gráfico dos tempos médios da Busca Tabu para esta instância, para os diferentes valores de T. É possível observar que mesmo com o aumento do valor de T, não há uma variação considerável no tempo de execução, o que faz sentido, visto que este parâmetro não define o tempo de busca, e sim o período de restrição de cada movimento. Além disso, é possível observar que as execuções que receberam uma solução inicial aleatória levaram mais tempo para terem suas buscas encerradas, o que leva ao questionamento de a qualidade do



(a) T = 50 iterações



(b) T = 100 iterações

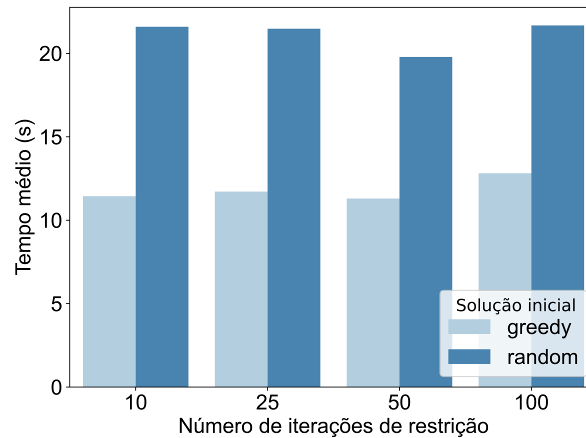
Fonte: Autoria própria.

ponto de partida ser ou não um fator determinante no tempo de convergência do método.

Os resultados do SA podem ser vistos na Figura 9. Esta Figura apresenta o número de conflitos médio obtido pelo SA em cada instância, considerando a temperatura máxima inicial do processo de têmpera. É possível observar que a versão com solução inicial gulosa se saiu melhor que a aleatória, em todos os casos de teste, gerando soluções com menor número médio de conflitos. Outro ponto a se observar é que para os conjuntos de instâncias do grupo SGB e MYC, o número absoluto de conflitos, foi o menor encontrado, e neste grupo, partir de uma solução gulosa gerou resultados com zero conflitos.

Além disso, o comportamento geral do método permanece o mesmo para todos os valores de T_max testados.

Figura 8 – Média do tempo de execução da Busca Tabu para a instância $lei450_15c$



Fonte: Autoria própria.

Uma análise a respeito do tempo de execução do SA pode ser realizada a partir da Figura 10. Para se ter uma comparação justa com a Busca Tabu, escolheu-se também a instância *le450_15c*. É possível observar que o tempo médio de execução do SA aumenta em conjunto com a temperatura inicial. Isso corrobora o fato de que o parâmetro T_max define o tempo de busca do método. Além disso, é possível observar que para o SA, diferente da Busca Tabu, o tempo de execução é praticamente o mesmo quando utilizadas as soluções iniciais gulosa e aleatória. Isso também decorre do fato de ser o parâmetro T_max que define o tempo máximo de execução da meta-heurística.

Por fim, a Tabela 3 apresenta um comparativo do menor número médio de conflitos obtidos por cada meta-heurística, quando aplicadas ao GCP.

Tabela 3 – Comparativo do menor número médio de conflitos obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória.

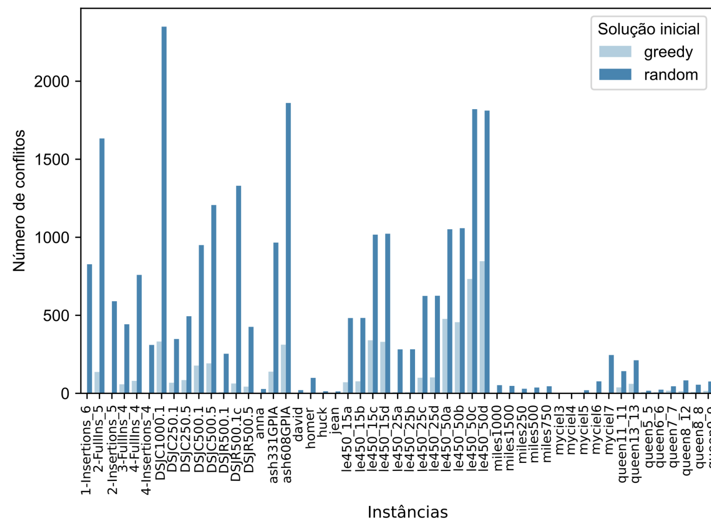
Instância	Busca Tabu				Simulated Annealing			
	Greedy		Random		Greedy		Random	
	S	T(s)	S	T(s)	S	T(s)	S	T(s)
1-Insertions_6	0,00	0,00	1,70	37,33	0,00	0,00	800,90	34,28
2-FullIns_5	15,20	157,68	6,20	126,14	136,90	13,67	1604,00	52,66
2-Insertions_5	0,00	0,00	0,90	32,82	0,00	0,00	572,10	37,64
3-FullIns_4	5,30	14,40	2,40	8,08	57,80	5,72	427,60	18,43
4-FullIns_4	5,80	91,45	3,50	46,01	79,80	3,10	739,60	46,66
4-Insertions_4	0,00	0,00	0,10	11,24	0,00	0,00	297,60	17,77
DSJC1000.1	208,50	238,00	210,60	390,99	332,00	5,26	2309,10	85,73
DSJC250.1	40,50	2,13	37,30	2,95	67,60	3,17	332,90	6,96
DSJC250.5	84,70	0,21	114,90	2,48	84,90	4,59	479,50	5,60
DSJC500.1	98,30	25,47	97,40	33,00	177,80	4,04	929,50	26,70
DSJC500.5	191,50	1,57	239,80	26,34	192,00	1,26	1180,80	21,53
DSJR500.1	5,50	6,38	5,30	22,64	7,00	1,38	241,20	22,90
DSJR500.1c	61,70	3,64	172,60	36,33	63,00	1,12	1309,60	16,94
DSJR500.5	42,80	1,45	61,90	21,43	43,00	1,06	409,40	17,54

Tabela 3 – Comparativo do menor número médio de conflitos obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória.

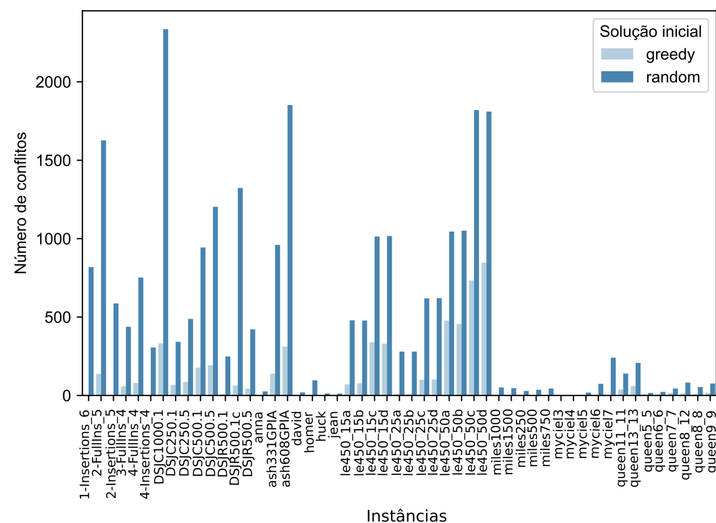
	Busca Tabu				Simulated Annealing			
anna	0,80	0,09	1,30	0,17	1,00	0,09	23,90	1,21
ash331GPIA	29,30	83,27	16,20	87,83	138,60	7,05	943,10	59,84
ash608GPIA	35,70	678,45	31,60	842,81	311,50	38,71	1828,10	99,75
david	0,90	0,02	1,40	0,04	1,00	0,04	17,30	0,50
homer	2,70	8,20	3,70	19,60	3,00	1,35	90,80	19,71
huck	0,00	0,00	0,40	0,02	0,00	0,00	11,20	0,41
jean	0,00	0,00	0,80	0,02	0,00	0,00	10,00	0,46
le450_15a	43,50	15,57	42,80	28,54	71,00	1,14	466,70	17,14
le450_15b	43,20	17,08	40,90	30,08	76,90	3,03	469,10	15,36
le450_15c	267,50	19,20	268,40	28,82	339,80	3,73	1000,10	16,93
le450_15d	271,50	16,97	273,00	26,48	329,80	6,45	1003,40	12,70
le450_25a	6,30	5,37	11,50	20,29	8,00	0,94	270,40	9,59
le450_25b	5,40	4,41	9,10	19,15	7,00	0,95	271,40	14,42
le450_25c	82,40	9,95	81,60	25,41	99,90	5,52	605,60	16,18
le450_25d	80,20	11,64	84,60	22,70	101,90	2,13	607,70	8,13
le450_5a	93,80	39,09	73,00	47,47	476,80	4,96	1028,40	25,03
le450_5b	91,50	37,64	69,30	52,86	455,70	3,30	1035,80	25,07
le450_5c	80,40	20,36	76,50	22,22	732,10	9,34	1791,80	24,41
le450_5d	80,30	16,82	75,50	23,40	846,40	1,82	1783,00	27,13
miles1000	2,00	0,03	5,80	0,17	2,00	0,07	47,50	1,13
miles1500	3,00	0,03	6,20	0,16	3,00	0,07	44,50	0,94
miles250	0,20	0,02	0,80	0,15	0,90	0,16	26,70	1,27
miles500	1,80	0,07	1,90	0,15	2,00	0,08	34,30	0,76
miles750	2,10	0,04	3,60	0,18	3,00	0,07	41,90	1,01
myciel3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
myciel4	0,00	0,00	0,00	0,00	0,00	0,00	2,10	0,09
myciel5	0,00	0,00	1,10	0,01	0,00	0,00	16,70	0,24
myciel6	0,00	0,00	7,10	0,05	0,00	0,00	68,70	0,79
myciel7	0,00	0,00	0,50	0,56	0,00	0,00	229,60	3,13
queen11_11	36,70	0,03	45,70	0,20	37,90	0,24	134,10	1,33
queen13_13	58,70	0,23	59,90	0,51	60,90	0,35	202,00	2,34
queen5_5	6,90	0,00	7,00	0,00	8,00	0,01	15,00	0,10
queen6_6	6,60	0,00	8,90	0,00	7,00	0,02	20,60	0,14
queen7_7	16,50	0,00	21,00	0,02	16,90	0,02	41,60	0,25
queen8_12	13,90	0,01	32,40	0,06	13,80	0,28	76,90	0,80
queen8_8	9,00	0,00	23,40	0,02	9,00	0,03	51,60	0,37
queen9_9	15,80	0,01	32,60	0,03	15,80	0,04	71,50	0,60

Observando a Tabela 3 é possível observar que num contexto geral, a Busca Tabu apresentou resultados de melhor qualidade que o SA, com a versão com solução inicial gulosa apresentando os menores índices de conflitos. Para algumas poucas instâncias, como *DSJR500.5, anna, homer* e *jean*, a versão do SA com solução inicial gulosa saiu-se melhor que a versão aleatória da Busca Tabu. Outro ponto a se considerar é o tempo de execução. O SA com método guloso mostrou-se a implementação mais rápida, em 34 das 51 instâncias de teste, para convergir a uma solução.

Figura 9 – Resultados do SA para o GCP



(a) T_max = 100



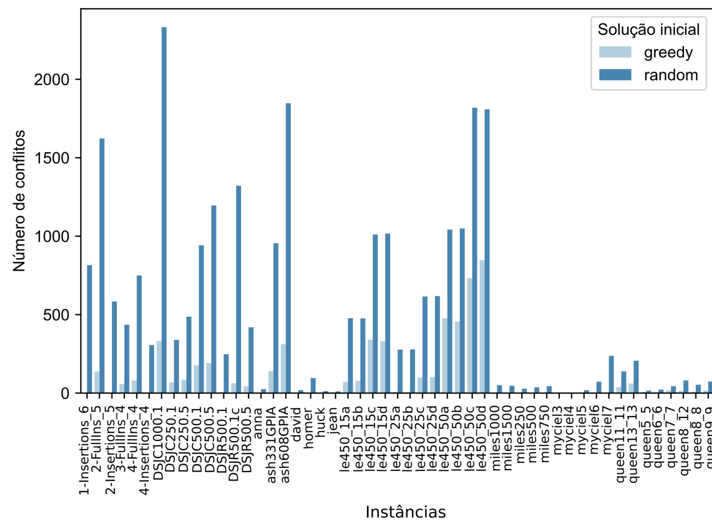
(b) T_max = 1000

4.2 RESULTADOS DA ALOCAÇÃO DE REGISTRADORES

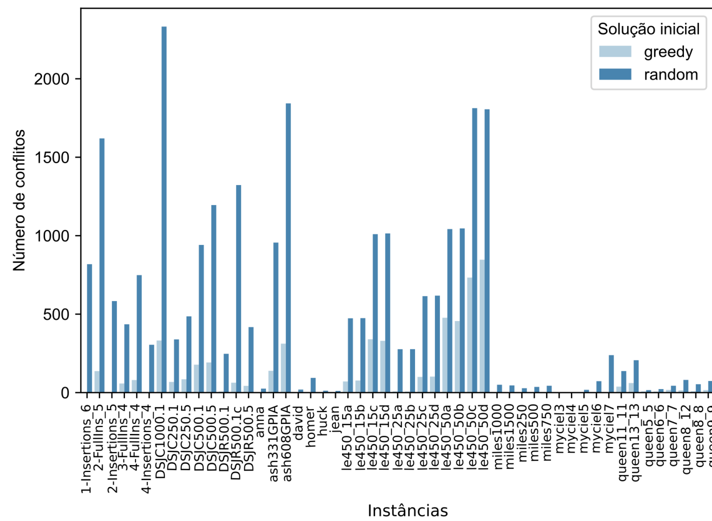
Nesta seção, são apresentados os resultados obtidos ao aplicar as meta-heurísticas ao problema da alocação de registradores. Os resultados são categorizados com base no número de registradores disponível para alocar as variáveis, conforme exposto na Seção 3.1. Além disso, os resultados de ambas as meta-heurísticas serão abordados em conjunto.

Ressalta-se que nesta seção, a qualidade das soluções é medida por meio do número de *spillings*, visto que em todos os casos o fluxograma da Figura 6 derrama variáveis para a memória até que o número de conflitos seja zerado.

Para a Busca Tabu, escolheu-se manter os mesmos valores de T que no GCP, enquanto



(a) T_max = 5000



(b) T_max = 10000

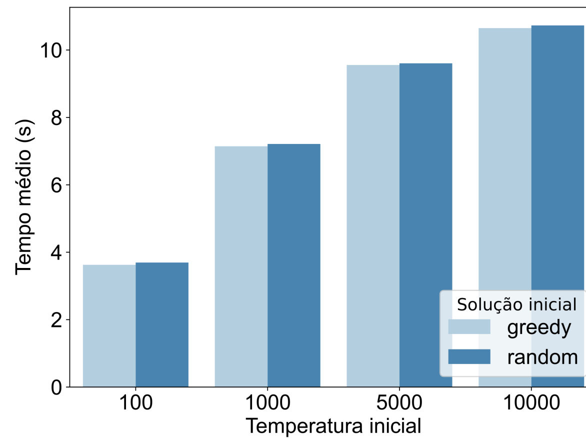
Fonte: Autoria própria.

que para o SA optou-se por utilizar $T_max = \{100, 1000, 10000\}$ para avaliar o desempenho em três ordens de grandeza diferentes.

Na Figura 11 são apresentados os resultados quando o número de registradores disponíveis é oito. Ao analisar individualmente a Busca Tabu, é possível observar que a versão com método guloso apresenta maior variação no número de *spilings* ao variar o parâmetro T. Além disso, nota-se que o método aleatório, num contexto geral, apresenta um desempenho igual ou pior que o guloso. Outro ponto a ser considerado, é que variar o parâmetro T não leva a diferenças expressivas no número de *spillings*.

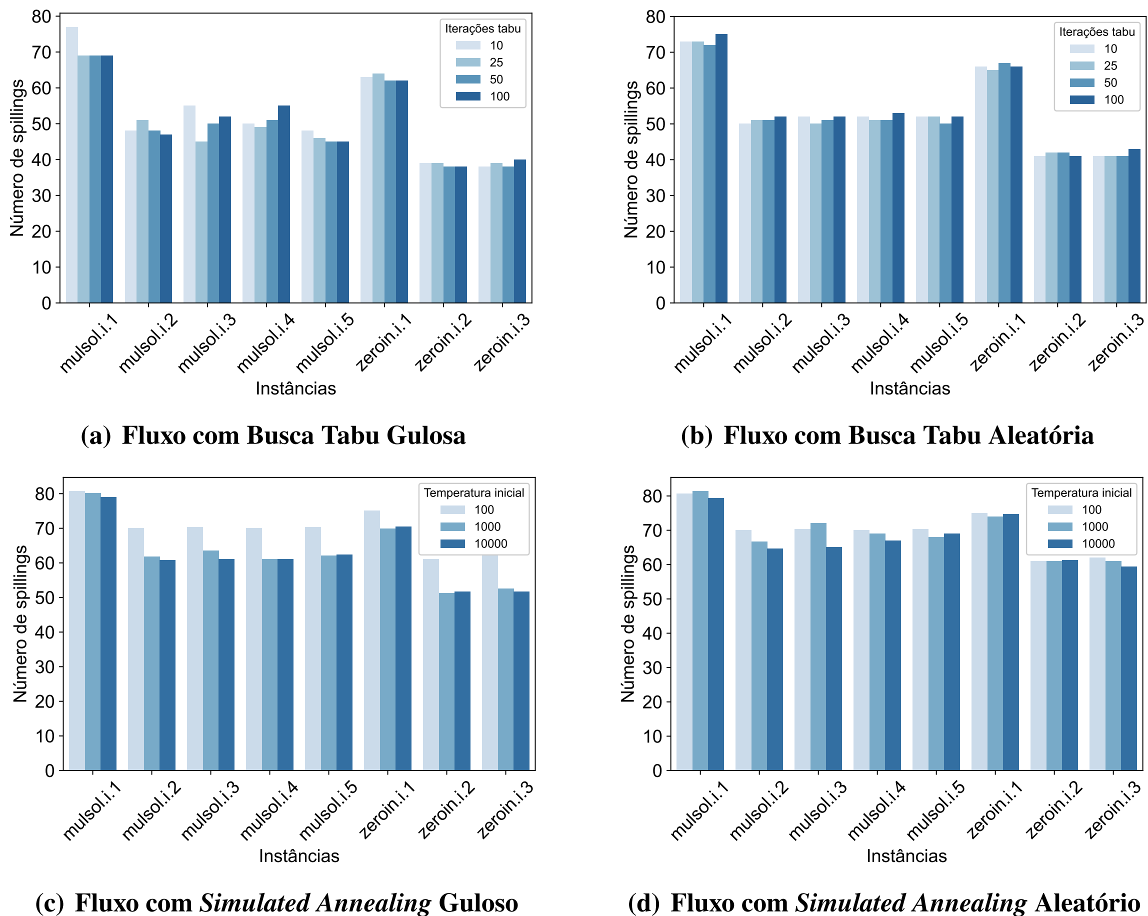
Ao comparar os gráficos da Figura 11(c) e Figura 11(d) é notável que o SA com solução inicial gulosa gerou soluções de melhor qualidade. O aumento da Temperatura inicial também

Figura 10 – Média do tempo de execução do *Simulated Annealing* para a instância $lei450_15c$



Fonte: Autoria própria.

Figura 11 – Número de *spillings* com 8 registradores



(a) Fluxo com Busca Tabu Gulosa

(b) Fluxo com Busca Tabu Aleatória

(c) Fluxo com *Simulated Annealing* Guloso

(d) Fluxo com *Simulated Annealing* Aleatório

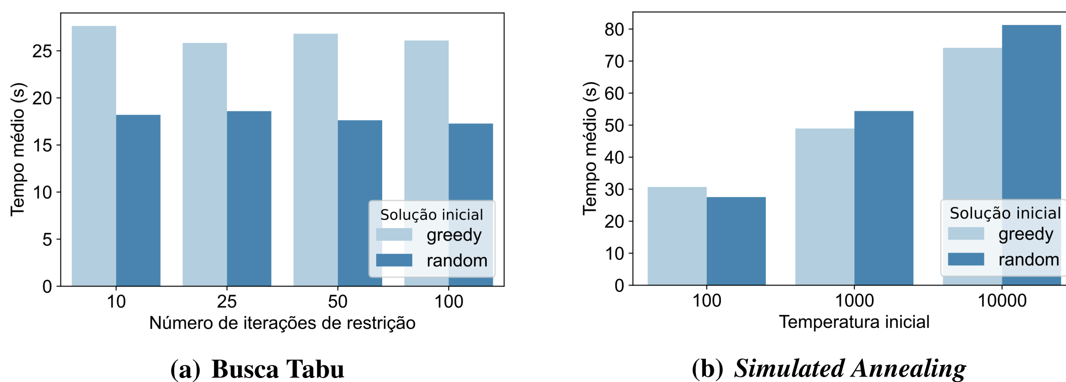
Fonte: Autoria própria.

mostrou-se um fator potencializador de qualidade, visto que os piores resultados foram obtidos com $T_max = 100$. Entretanto, não há como afirmar que todo aumento de temperatura seja positivo, visto que para algumas instâncias o melhor resultado foi obtido com o valor intermediário para o parâmetro.

Quando confrontados os resltados da Busca Tabu com o SA, se observa que as alocações geradas pela Busca apresentam menor número de *spillings*, indicando que este método insere menos operações de *load* e *store* no código-objeto do programa.

Para analisar o tempo de execução dos algoritmos implementados para a alocação, se escolheu a instância *zeroin.i.1*, por apresentar o programa com o maior número de variáveis (ver Tabela 2). O comparativo de tempo entre a Busca Tabu e o *Simulated Annealing* pode ser observado na Figura 12.

Figura 12 – Gráfico comparativo dos tempos de execução do processo de alocação com 8 registradores para a instância *zeroin.i.1*



(a) Busca Tabu **(b) *Simulated Annealing***

Fonte: Autoria própria.

Observando os gráficos da Figura 12, nota-se que em termos gerais a Busca Tabu convergiu em menos tempo. Tal como no GCP, é possível observar que o aumento no parâmetro T da Busca não leva a uma variação considerável do tempo de execução, enquanto que o aumento de T_max leva a aumento considerável no tempo de execução do SA. É notável também que, na Busca Tabu a versão gulosa mostrou-se mais lenta que a aleatória, enquanto que no SA, para dois dos três valores de temperatura inicial, a versão com solução aleatória foi mais lenta.

A Tabela 4 faz um comparativo do menor número médio de *spillings* e seus respectivos tempos de execução.

Por meio da Tabela, é possível confirmar que a Busca Tabu foi capaz de apresentar os melhores resultados, inclusive com menores tempos de execução. Dentre as quatro versões, a Busca Tabu com método guloso figura em primeiro lugar na qualidade das alocações, e a versão com solução aleatória apresenta os menores tempos de execução.

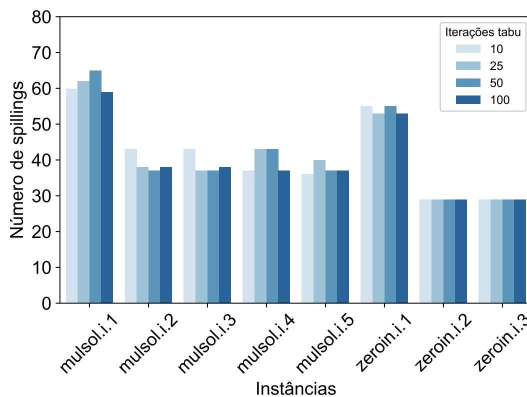
A Figura 13 apresenta os resultados quando disponibilizados 12 registradores para o processo de alocação. Novamente, todas as quatro versões são apresentadas em conjunto.

Nota-se que quando comparado com a Figura 11, o número de *spillings* diminui, obviamente por mais registradores estarem disponíveis.

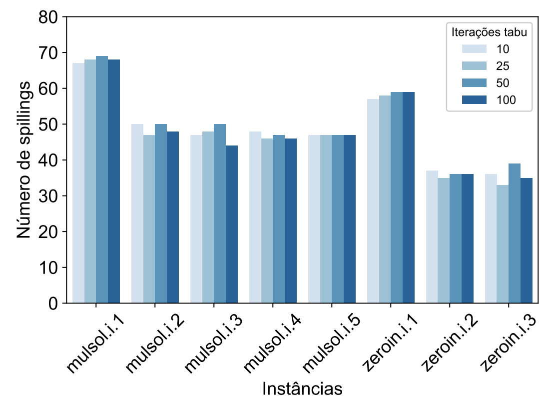
Tabela 4 – Comparativo do menor número médio de spillings obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória.

Instância	Busca Tabu				Simulated Annealing			
	Greedy		Random		Greedy		Random	
	S	T(s)	S	T(s)	S	T(s)	S	T(s)
mulsol.i.1	69,00	28,23	72,00	20,48	78,50	101,44	79,00	75,52
mulsol.i.2	47,00	17,80	50,00	13,47	59,50	163,63	62,00	125,91
mulsol.i.3	45,00	17,86	50,00	12,45	60,00	153,76	63,00	123,85
mulsol.i.4	49,00	17,96	51,00	13,10	59,50	51,21	65,00	63,86
mulsol.i.5	45,00	16,97	50,00	12,78	60,50	96,52	65,00	86,29
zeroin.i.1	62,00	38,80	65,00	23,32	69,00	53,02	73,00	53,59
zeroin.i.2	38,00	26,82	41,00	18,20	49,50	42,55	59,00	23,94
zeroin.i.3	38,00	22,63	41,00	15,39	49,50	120,55	55,00	126,76

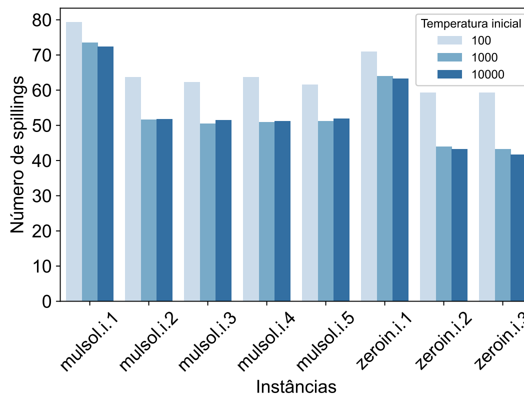
Figura 13 – Número de *spillings* com 12 registradores



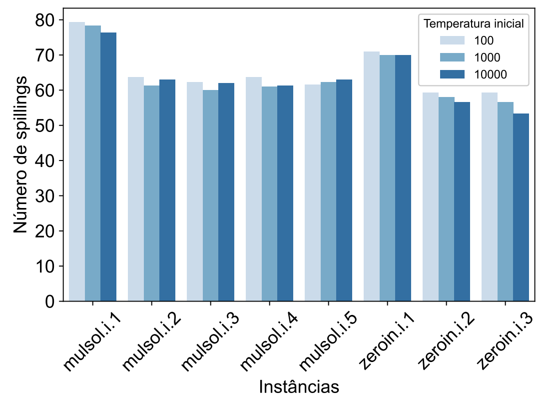
(a) Fluxo com Busca Tabu Gulosa



(b) Fluxo com Busca Tabu Aleatória



(c) Fluxo com *Simulated Annealing* Guloso



(d) Fluxo com *Simulated Annealing* Aleatório

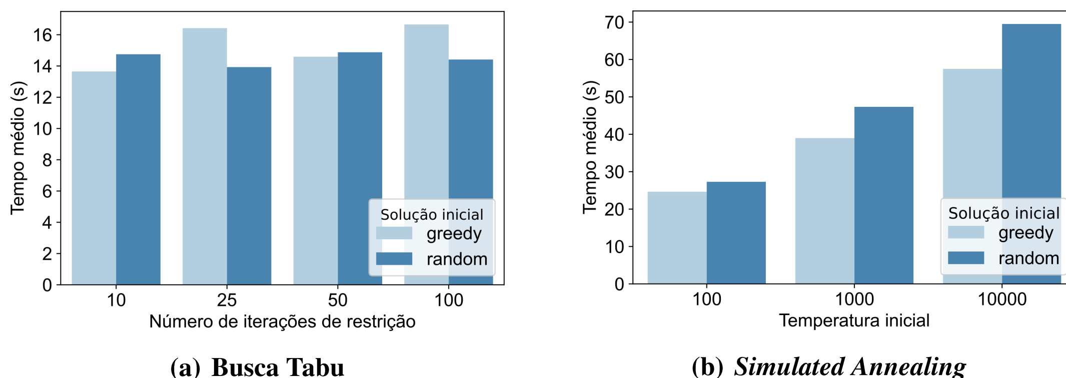
Fonte: Autoria própria.

Ao analisar cada método individualmente, é possível observar que tanto para a Busca Tabu quanto para o SA a versão com método guloso gerou menos *spillings*. Além disso, o comportamento observado a partir da variação dos parâmetros segue o mesmo da Figura 11, com T não introduzindo muita diferença na Busca Tabu, e com T_max apresentando soluções com melhor qualidade em temperaturas mais elevadas.

A comparação direta das meta-heurísticas revela que, no contexto geral, a Busca Tabu apresenta alocações de melhor qualidade que o SA. Em algumas instâncias há diferença próxima de 20% quando comparando as meta-heurísticas com mesmo método de solução inicial.

Novamente, se faz necessária uma análise do tempo de execução do processo de alocação utilizando cada meta-heurística. A Figura 14 apresenta esse comparativo. É possível visualizar

Figura 14 – Gráfico comparativo dos tempos de execução do processo de alocação com 12 registradores para a instância *zeroin.i.1*



(a) Busca Tabu **(b) *Simulated Annealing***

Fonte: Autoria própria.

que a Busca Tabu apresenta menor tempo de execução. Quanto a variação dos parâmetros, na Busca Tabu a diferença entre os tempos de execução não ultrapassou 4 s ao utilizar outros valores de T. No SA, novamente o aumento do valor de T_max leva a um aumento no tempo de execução.

Por fim, a Tabela 5 apresenta um comparativo do menor número médio de *spillings* e do tempo de execução obtido por cada versão do alocador implementado.

Tabela 5 – Comparativo do menor número médio de spillings obtidos por cada meta-heurística, considerando as soluções iniciais gulosa e aleatória.

Instância	Busca Tabu				Simulated Annealing			
	Greedy		Random		Greedy		Random	
	S	T(s)	S	T(s)	S	T(s)	S	T(s)
mulsol.i.1	59,00	22,69	67,00	20,75	72,00	26,99	75,00	27,07
mulsol.i.2	37,00	14,23	47,00	12,54	49,00	100,42	58,00	109,45
mulsol.i.3	37,00	13,02	44,00	10,97	50,00	13,24	59,00	14,45
mulsol.i.4	37,00	12,75	46,00	11,11	50,50	13,54	60,00	14,90
mulsol.i.5	36,00	12,05	47,00	11,02	48,50	33,58	57,00	36,38
zeroin.i.1	53,00	25,43	57,00	21,39	62,00	90,97	67,00	94,41
zeroin.i.2	29,00	13,65	35,00	13,93	41,50	100,13	53,00	120,17
zeroin.i.3	29,00	15,67	33,00	12,96	40,50	18,79	51,00	22,19

A partir da Tabela, é possível inferir que a Busca Tabu com método guloso apresenta os melhores resultados, a versão com solução aleatória apresenta os melhores tempos de execução. Este comportamento é o mesmo observado a partir da Tabela 4.

A seguir, são apresentadas as principais conclusões obtidas a partir dos resultados observados.

5 CONCLUSÃO

A Coloração de Grafos é um problema extremamente abordado na computação, tanto em sua versão original, quanto nos problemas derivados, como o Problema de Agendamentos e a Alocação de Registradores. Devido a grande importância do GCP, este trabalho propôs implementar e analisar meta-heurísticas clássicas e já familiares ao problema: a Busca Tabu e *Simulated Annealing*. Além disso, teve como objetivo incorporar essas meta-heurísticas ao processo de Alocação de Registradores no processo de compilação, visando investigar o motivo de poucas abordagens com meta-heurísticas a este problema.

Ao executar o trabalho, a versão do GCP abordada foi o $k-$GCP, um problema que busca decidir se um grafo pode ser colorido ou não com k cores, e que é NP-Completo. Em relação a Alocação de Registradores, este foi resolvido como um problema derivado do GCP.

A utilização de meta-heurísticas busca melhores resultados, ao sacrificar tempo de execução, principalmente quando comparadas com heurísticas. Dessa maneira, além de buscar soluções de qualidade, é necessário otimizar as meta-heurísticas, de modo que sejam mais rápidas.

O objetivo deste trabalho era comparar a performance das meta-heurísticas Busca Tabu e *Simulated Annealing*, no GCP e na Alocação de Registradores.

Quanto ao GCP, diante dos resultados obtidos, foi possível concluir que nos melhores casos de cada meta-heurísticas a Busca Tabu gerou colorações com menos conflitos, enquanto que o SA teve, proporcionalmente menor tempo de execução. Além disso, as versões com solução inicial obtidas por Método Guloso, se mostraram capazes de alcançar colorações com menos conflitos que a versão com solução inicial aleatória. Outro ponto a ser evidenciado, é que ambos os métodos geraram menor número de conflitos para casos de instâncias dos grupos MYC e SGB, que são grafos construídos segundo algum critério que não a aleatoriedade. Já as instâncias do grupo DSJ, compostas por grafos aleatórios, demonstraram-se extremamente difíceis de colorir de maneira ótima, gerando alto índice de conflitos. Além disso, as versões implementadas das meta-heurísticas não foram capazes de colorir de maneira ótima todas as instâncias testadas, mesmo que o número de cores disponibilizado para cada uma fosse o mínimo conhecido que pudesse colorir os grafos –ou, em alguns casos, o número cromático do grafo.

Em relação a Alocação de Registradores, as meta-heurísticas foram incorporadas a uma versão do algoritmo de Chaitin para alocação de registradores, em que as etapas de simplificar e selecionar são incorporadas a meta-heurística. Para esse problema, foi possível concluir que a

Busca Tabu foi capaz de obter melhores alocações, tanto em número de conflitos, quanto em tempo de execução, o que difere do GCP, em que o SA era mais rápido. Acredita-se que esse comportamento seja decorrente da versão implementada do algoritmo de Chaitin, em que após cada tentativa de alocação, a variável que tivesse o maior número de interferência com as demais era marcada para *spilling*, e, como a cada iteração do algoritmo, o SA gerou colorações com mais conflitos que a Busca Tabu, acaba precisando de maior tempo de execução para zerar o número de conflitos, aumentando tanto o tempo de execução, quanto o número de *spillings*. Entretanto, ao pensar a respeito da viabilidade das meta-heurísticas implementadas para a Alocação de Registradores, percebe-se que o tempo de execução é relativamente elevado. A maior instância submetida as meta-heurísticas para o processo de alocação, a *zeroin.i.1*, cujo grafo contém 211 vértices (equivalente a 211 variáveis), teve, nas execuções com menor número de *spillings*, tempos de execução de 38 *s* para 8 registradores (Tabela 4) e 26 *s* para 12 registradores (Tabela 5), ambos tempos consideravelmente altos para o processo de compilação.

Diante do exposto, é possível concluir que das versões implementadas das meta-heurísticas Busca Tabu e *Simulated Annealing*, a mais adequada tanto para o GCP quanto para a Alocação de Registradores, é a Busca Tabu, por apresentar soluções de maior qualidade, com pouca ou nenhuma desvantagem em relação ao tempo de execução.

5.1 TRABALHOS FUTUROS

Futuras pesquisas relativas ao GCP, utilizando as meta-heurísticas implementadas neste trabalho, poderiam incluir os seguintes tópicos:

- Estudar a utilização de outras estruturas de vizinhança, para o processo de movimentação e busca local, visando aumentar a abrangência da busca por soluções melhores.
- Abordar outras combinações de parâmetros, como o parâmetro T da Busca Tabu, e o parâmetro T_max do *Simulated Annealing*.
- Modificar a estrutura da lista tabu, inserindo um novo parâmetro, como um fator limitante para o número de movimentos que podem estar restritos.
- Estudar a viabilidade de um mecanismo de reaquecimento no *Simulated Annealing*, para aumentar o grau de perturbação e fuga de ótimos locais.

Em relação a Alocação de Registradores, seguindo a abordagem da coloração de grafos, além dos tópicos acima, novas pesquisas poderiam abordar:

- O estudo de uma nova incorporação das meta-heurísticas ao algoritmo de Chaitin, de modo que etapas como Coalescer, Simplificar e Selecionar possam ser realizadas de maneira mais inteligente.
- A utilização de outros critérios para seleção da variável a ser marcada para *spilling*.
- Uma possível hibridização das meta-heurísticas Busca Tabu e *Simulated Annealing*, visando aproveitar as vantagens de ambas.

REFERÊNCIAS

AHMED, S. Applications of graph coloring in modern computer science. **International Journal of Computer and Information Technology**, v. 3, n. 2, p. 1–7, 2012.

AHO, A. V. *et al.* **Compiladores: Princípios, técnicas e ferramentas**. 2. ed. [*S. l.*]: LTC, 2007.

AOKI, T.; ARANHA, C.; KANOH, H. PSO algorithm with transition probability based on hamming distance for graph coloring problem. In: IEEE. 2015 IEEE International Conference On Systems, Man, And Cybernetics. [*S. l.: s. n.*], 2015. P. 1956–1961.

BERNSTEIN, D. *et al.* Spill code minimization techniques for optimizing compliers. **ACM SIGPLAN Notices**, ACM New York, NY, USA, v. 24, n. 7, p. 258–263, 1989.

BRÉLAZ, D. New methods to color the vertices of a graph. **Communications of the ACM**, ACM New York, NY, USA, v. 22, n. 4, p. 251–256, 1979.

BRIGGS, P. **Register allocation via graph coloring**. 1992. Tese (Doutorado) – Rice University.

BRIGGS, P.; COOPER, K. D.; KENNEDY, K. *et al.* Coloring heuristics for register allocation. **Acm Sigplan Notices**, ACM New York, NY, USA, v. 24, n. 7, p. 275–284, 1989.

BRIGGS, P.; COOPER, K. D.; TORCZON, L. Improvements to graph coloring register allocation. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM New York, NY, USA, v. 16, n. 3, p. 428–455, 1994.

BROWN, J. R. Chromatic scheduling and the chromatic number problem. **Management science**, INFORMS, v. 19, 4-part-1, p. 456–463, 1972.

CARLSON, J. A.; JAFFE, A.; WILES, A. **The millennium prize problems**. [*S. l.*]: Citeseer, 2006.

CHAITIN, G. J. Register allocation & spilling via graph coloring. **ACM Sigplan Notices**, ACM New York, NY, USA, v. 17, n. 6, p. 98–101, 1982.

CHAITIN, G. J. *et al.* Register allocation via coloring. **Computer languages**, Elsevier, v. 6, n. 1, p. 47–57, 1981.

CHAMS, M.; HERTZ, A.; DE WERRA, D. Some experiments with simulated annealing for coloring graphs. **European Journal of Operational Research**, Elsevier, v. 32, n. 2, p. 260–266, 1987.

CHARTRAND, G.; ZHANG, P. **Chromatic graph theory**. [*S. l.*]: CRC press, 2019.

COOK, S. The P versus NP problem. **The millennium prize problems**, p. 87–104, 2006.

COOPER, K.; TORCZON, L. **Engineering a compiler**. [*S. l.*]: Elsevier, 2011.

DAS, D.; AHMAD, S. A.; KUMAR, V. Deep Learning-based Approximate Graph-Coloring Algorithm for Register Allocation. In: IEEE. 2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar). [*S. l.: s. n.*], 2020. P. 23–32.

DASGUPTA, S.; PAPADIMITRIOU, C. H.; VAZIRANI, U. V. **Algorithms**. [*S. l.*]: McGraw-Hill Higher Education New York, 2008.

DOKEROGLU, T.; SEVINC, E. Memetic Teaching–Learning-Based Optimization algorithms for large graph coloring problems. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 102, p. 104282, 2021.

DORNE, R.; HAO, J.-K. A new genetic local search algorithm for graph coloring. In: SPRINGER. INTERNATIONAL Conference on Parallel Problem Solving from Nature. [*S. l.: s. n.*], 1998. P. 745–754.

DORNE, R.; HAO, J.-K. Tabu search for graph coloring, T-colorings and set T-colorings. In: META-HEURISTICS. [*S. l.*]: Springer, 1999. P. 77–92.

FAN, G. Circular chromatic number and Mycielski graphs. **Combinatorica**, Springer, v. 24, n. 1, p. 127–135, 2004.

FARACH-COLTON, M.; LIBERATORE, V. On local register allocation. **Journal of Algorithms**, Elsevier, v. 37, n. 1, p. 37–65, 2000.

FLEURENT, C.; FERLAND, J. A. Genetic and hybrid algorithms for graph coloring. **Annals of Operations Research**, Springer, v. 63, n. 3, p. 437–461, 1996.

GALINIER, P.; HAO, J.-K. Hybrid evolutionary algorithms for graph coloring. **Journal of combinatorial optimization**, Springer, v. 3, n. 4, p. 379–397, 1999.

GAREY, M. R.; JOHNSON, D. S.; STOCKMEYER, L. Some simplified NP-complete problems. In: PROCEEDINGS of the sixth annual ACM symposium on Theory of computing. [*S. l.: s. n.*], 1974. P. 47–63.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. USA: W. H. Freeman & Co., 1990.

GENDREAU, M.; POTVIN, J.-Y. *et al.* **Handbook of metaheuristics**. [*S. l.*]: Springer, 2010. v. 2.

GENDREAU, M.; POTVIN, J.-Y. Tabu search. In: SEARCH methodologies. [*S. l.*]: Springer, 2005. P. 165–186.

GEORGE, L.; APPEL, A. W. Iterated register coalescing. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM New York, NY, USA, v. 18, n. 3, p. 300–324, 1996.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & operations research**, Elsevier, v. 13, n. 5, p. 533–549, 1986.

GLOVER, F. Tabu search—part I. **ORSA Journal on computing**, Informs, v. 1, n. 3, p. 190–206, 1989.

GLOVER, F.; TAILLARD, E. A user's guide to tabu search. **Annals of operations research**, Springer, v. 41, n. 1, p. 1–28, 1993.

GOLDBARG, M. **Otimização combinatória e programação linear-2a ediç**. [*S. l.*]: Elsevier Brasil, 2005. v. 2.

GONZALEZ, T. F. Basic Methodologies and Applications. In: HANDBOOK of Approximation Algorithms and Metaheuristics, Second Edition. [*S. l.*]: Chapman e Hall/CRC, 2018. P. 27–41.

GOODRICH, M. T.; TAMASSIA, R. **Projeto de algoritmos: fundamentos, análise e exemplos da internet**. [*S. l.*]: Bookman Editora, 2009.

GOODWIN, D. W.; WILKEN, K. D. Optimal and Near-optimal Global Register Allocation Using 0–1 Integer Programming. **Software: practice and experience**, Wiley Online Library, v. 26, n. 8, p. 929–965, 1996.

GU, J. *et al.* **Algorithms for the satisfiability (SAT) problem: A survey**. [*S. l.*], 1996.

HANSEN, P. The steepest ascent mildest descent heuristic for combinatorial programming. In: CONGRESS on numerical methods in combinatorial optimization, Capri, Italy. [*S. l.: s. n.*], 1986. P. 70–145.

HERTZ, A.; WERRA, D. de. Using tabu search techniques for graph coloring. **Computing**, Springer, v. 39, n. 4, p. 345–351, 1987.

HINDI, M. M.; YAMPOLSKIY, R. V. Genetic algorithm applied to the graph coloring problem. In: PROC. 23rd Midwest Artificial Intelligence and Cognitive Science Conf. [*S. l.: s. n.*], 2012. P. 61–66.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence (Complex Adaptive Systems) (English Edition)**. 1st. [*S. l.*]: A Bradford Book, 1975.

HUNTER, J. D. Matplotlib: A 2D graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. DOI: `10.1109/MCSE.2007.55`.

JOHNSON, D. S.; ARAGON, C. R. *et al.* Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. **Operations research**, INFORMS, v. 39, n. 3, p. 378–406, 1991.

JOHNSON, D. S.; TRICK, M. A. **Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993**. [*S. l.*]: American Mathematical Soc., 1996. v. 26.

KARP, R. M. Reducibility among combinatorial problems. In: COMPLEXITY of computer computations. [*S. l.*]: Springer, 1972. P. 85–103.

KEMPE, A. B. On the geographical problem of the four colours. **American journal of mathematics**, JSTOR, v. 2, n. 3, p. 193–200, 1879.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **science**, American association for the advancement of science, v. 220, n. 4598, p. 671–680, 1983.

KNUTH, D. E. The stanford graphbase: a platform for combinatorial algorithms. In: SODA. [*S. l.: s. n.*], 1993. v. 93, p. 41–43.

KOES, D.; GOLDSTEIN, S. C. A progressive register allocator for irregular architectures. In: IEEE. INTERNATIONAL Symposium on Code Generation and Optimization. [*S. l.: s. n.*], 2005. P. 269–279.

KONG, T.; WILKEN, K. D. Precise register allocation for irregular architectures. In: IEEE. PROCEEDINGS. 31st Annual ACM/IEEE International Symposium on Microarchitecture. [*S. l.: s. n.*], 1998. P. 297–307.

KRI, F.; FEELEY, M. Genetic instruction scheduling and register allocation. In: IEEE. XXIV international conference of the chilean computer science society. [*S. l.: s. n.*], 2004. P. 76–83.

KRI, F.; GÓMEZ, C.; CARO, P. Overview of metaheuristics methods in compilation. In: SPRINGER. MEXICAN International Conference on Artificial Intelligence. [*S. l.: s. n.*], 2005. P. 483–493.

KUMAR, S. N.; PANNEERSELVAM, R. A survey on the vehicle routing problem and its variants. Scientific Research Publishing, 2012.

KURDAHI, F. J.; PARKER, A. C. REAL: A program for register allocation. In: PROCEEDINGS of the 24th ACM/IEEE Design Automation Conference. [*S. l.: s. n.*], 1987. P. 210–215.

LEIGHTON, F. T. A graph coloring algorithm for large scheduling problems. **Journal of research of the national bureau of standards**, v. 84, n. 6, p. 489–506, 1979.

LEWANDOWSKI, G.; CONDON, A. Experiments with parallel graph coloring heuristics and applications of graph coloring. **Johnson and Trick (1996)**, p. 309–334, 1996.

LIBERATORE, V.; FARACH-COLTON, M.; KREMER, U. Evaluation of algorithms for local register allocation. In: SPRINGER. INTERNATIONAL Conference on Compiler Construction. [*S. l.: s. n.*], 1999. P. 137–152.

LINTZMAYER, C. N.; MULATI, M. H.; SILVA, A. F. da. Register allocation with graph coloring by ant colony optimization. In: IEEE. 2011 30th International Conference of the Chilean Computer Science Society. [*S. l.: s. n.*], 2011. P. 247–255.

LOUDEN, K. C.; SILVA, F. S. C. **Compiladores-Princıpios e Práticas**. [*S. l.*]: Cengage Learning Editores, 2004.

LÜ, Z.; HAO, J.-K. A memetic algorithm for graph coloring. **European Journal of Operational Research**, Elsevier, v. 203, n. 1, p. 241–250, 2010.

ŁUKASIK, S.; KOKOSIŃSKI, Z.; ŚWIĘTOŃ, G. Parallel simulated annealing algorithm for graph coloring problem. In: SPRINGER. INTERNATIONAL Conference on Parallel Processing and Applied Mathematics. [*S. l.: s. n.*], 2007. P. 229–238.

MCKINNEY, W. Data Structures for Statistical Computing in Python. In: WALT, S. van der; MILLMAN, J. (Ed.). **Proceedings of the 9th Python in Science Conference**. [*S. l.: s. n.*], 2010. P. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

MOGENSEN, T. Æ. **Introduction to compiler design**. [*S. l.*]: Springer, 2017.

PARDALOS, P. M.; MAVRIDOU, T.; XUE, J. The graph coloring problem: A bibliographic survey. In: HANDBOOK of combinatorial optimization. [*S. l.*]: Springer, 1998. P. 1077–1141.

PEREIRA, F. M. Q. **A survey on register allocation**. [*S. l.*], 2008.

POLETTO, M.; SARKAR, V. Linear scan register allocation. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM New York, NY, USA, v. 21, n. 5, p. 895–913, 1999.

QU, G.; POTKONJAK, M. Analysis of watermarking techniques for graph coloring problem. In: PROCEEDINGS of the 1998 IEEE/ACM international conference on Computer-aided design. [*S. l.: s. n.*], 1998. P. 190–193.

SALARI, E.; ESHGHI, K. An ACO algorithm for graph coloring problem. In: IEEE. 2005 ICSC Congress on computational intelligence methods and applications. [*S. l.: s. n.*], 2005. 5–pp.

SANTOS, P. R.; LANGLOIS, T. **Compiladores - Da Teoria à Prática**. 1. ed. [*S. l.*]: LTC, 2018. ISBN Compiladores - Da Teoria à Prática.

SETHI, R. Complete register allocation problems. **SIAM journal on Computing**, SIAM, v. 4, n. 3, p. 226–248, 1975.

SHIMOSAKA, G. **Estratégias de Solução para o Problema de Alocação de Registradores em Compiladores**. [*S. l.: s. n.*], 2019. P. 52.

STOCKMEYER, L. Planar 3-colorability is polynomial complete. **ACM Sigact News**, ACM New York, NY, USA, v. 5, n. 3, p. 19–25, 1973.

SUN, W. *et al.* A solution-driven multilevel approach for graph coloring. **Applied Soft Computing**, Elsevier, v. 104, p. 107174, 2021.

TALBI, E.-G. **Metaheuristics: from design to implementation**. [*S. l.*]: John Wiley & Sons, 2009. v. 74.

TEAM, T. pandas development. **pandas-dev/pandas: Pandas**. [*S. l.*]: Zenodo, fev. 2020. DOI: 10.5281/zenodo.3509134. Disponível em: https://doi.org/10.5281/zenodo.3509134.

VAN LAARHOVEN, P. J.; AARTS, E. H. Simulated annealing. In: SIMULATED annealing: Theory and applications. [*S. l.*]: Springer, 1987. P. 7–15.

WASKOM, M. L. seaborn: statistical data visualization. **Journal of Open Source Software**, The Open Journal, v. 6, n. 60, p. 3021, 2021. DOI: 10.21105/joss.03021. Disponível em: https://doi.org/10.21105/joss.03021.

WERRA, D. de. An introduction to timetabling. **European journal of operational research**, Elsevier, v. 19, n. 2, p. 151–162, 1985.