

Otimização da Alocação de Registradores com GRASP, Busca Tabu e Simulated Annealing: Um estudo comparativo

Aline Cardoso dos Santos, Bruno Duarte, Marcelo Teixeira, Marco Antonio Barbosa

Universidade Tecnológica Federal do Paraná

Via do Conhecimento, KM 01, Pato Branco, PR, Brasil

{aline.2019, brunoduarte}@alunos.utfpr.edu.br,

{mtex, mbarbosa}@utfpr.edu.br

RESUMO

Este artigo investiga o uso de metaheurísticas na alocação de registradores em compiladores, problema modelado como uma coloração de grafos. Por se tratar de um problema NP-completo, abordagens exatas tornam-se inviáveis em tempo razoável. Diante disso, foram implementadas e comparadas três metaheurísticas — *Greedy Randomized Adaptive Search Procedure* (GRASP), *Tabu Search* (TS) e *Simulated Annealing* (SA) — com o objetivo de minimizar conflitos sob diferentes restrições de cores. Os experimentos foram realizados em instâncias do conjunto DIMACS, representando cenários com número limitado de registradores. A TS apresentou melhor desempenho em grafos densos, enquanto o SA obteve resultados superiores em grafos regulares e menos conectados. Já o GRASP mostrou-se competitivo em configurações específicas. Os resultados sugerem que essas técnicas têm potencial para aprimorar a qualidade da alocação de registradores e reduzir o consumo de memória, especialmente em ambientes com restrições severas de recursos.

PALAVRAS CHAVE. Alocação de Registradores. Coloração de Grafos. Metaheurísticas.

MH - Metaheurísticas. OD - Otimização Discreta. IA - PO e IA

ABSTRACT

This paper explores the use of metaheuristics for register allocation in compilers, modeled as a graph coloring problem. Due to its NP-complete nature, exact methods are often impractical, while metaheuristics appear as alternatives. In this paper, we implement and compare three metaheuristics — *Greedy Randomized Adaptive Search Procedure* (GRASP), *Tabu Search* (TS), and *Simulated Annealing* (SA) — aiming to minimize conflicts under different color constraints. Experiments were conducted on DIMACS benchmark graphs to simulate limited-register scenarios. TS performed better on dense graphs, SA was more effective on regular and sparse ones, and GRASP showed competitive results in specific cases. The findings suggest that metaheuristics are promising tools for improving register allocation and reducing memory usage, especially under tight hardware constraints.

KEYWORDS. Register Allocation. Graph Coloring. Metaheuristics.

MH - Metaheurísticas. OD - Otimização Discreta. IA - PO e IA

1. Introdução

Compiladores são sistemas computacionais que traduzem código-fonte, escrito em uma linguagem de programação de alto nível, para uma linguagem de máquina executável pelo computador. Durante esse processo, a etapa de alocação de registradores representa uma etapa crítica da geração de código. Esta etapa visa mapear variáveis do programa para um conjunto limitado de registradores físicos da unidade de processamento [Cooper e Torczon, 2023]. Trata-se de um problema classicamente modelado como uma variação do problema de coloração de grafos, sendo classificado como NP-Difícil [Cook, 2006]. Soluções exatas, baseadas em enumeração ou programação inteira, embora teoricamente viáveis para instâncias pequenas, tornam-se computacionalmente limitadas na medida em que o número de variáveis cresce [Quintão Pereira e Palsberg, 2008].

Além de impactar diretamente o desempenho do código gerado, a alocação de registradores também exerce influência sobre a eficiência energética dos sistemas computacionais [Cooper e Torczon, 2023]. Estratégias eficazes de alocação podem reduzir a quantidade de acessos à memória principal, que são operações custosas em termos de consumo energético, contribuindo para uma execução mais sustentável [Beckitt-Marshall, 2021]. Dessa forma, uma alocação bem projetada não apenas otimiza o uso dos recursos computacionais, mas também favorece o desenvolvimento de *software* com menor demanda energética.

De um modo geral, a quantidade de registradores disponíveis varia conforme a arquitetura do processador, mas é limitada, tornando-se fundamental a eficiência do processo de alocação [Stallings, 2010]. Por exemplo, a arquitetura x86 de 32 bits disponibiliza apenas 8 registradores de uso geral, enquanto arquiteturas ARM contam com 16 registradores, e variantes RISC podem contar com até 32 registradores [Cooper e Torczon, 2023; Stallings, 2010]. Assim, políticas ineficientes de alocação podem acarretar operações frequentes de salvamento e restauração de dados na memória, elevando o custo computacional para a execução de um programa [Fischer et al., 2010].

Uma abordagem que permite avaliar a eficiência da alocação de registradores é associar este problema a um problema de Coloração de Grafos [Chaitin et al., 1981]. Desta forma, cada variável de um programa é representada por um nó, e as arestas conectam pares de variáveis que não podem ocupar o mesmo registrador simultaneamente. O objetivo é atribuir cores (registradores) aos nós de modo que dois nós adjacentes nunca compartilhem a mesma cor.

No entanto, o problema de coloração de grafos é reconhecidamente NP-Completo e, como tal, não se conhece um algoritmo que o resolva de forma exata e eficiente (em tempo polinomial) para todos os casos [Cook, 2006]. Embora os métodos exatos forneçam soluções ideais, eles são inviáveis à medida que o tamanho do problema cresce, o que é o caso da alocação de registradores, especialmente em se tratando de compiladores modernos que lidam com milhares de variáveis e exigem tempos de execução reduzidos [Quintão Pereira e Palsberg, 2008; Cooper e Torczon, 2023].

Diante disso, alternativas baseadas em heurísticas e metaheurísticas têm se mostrado promissoras por fornecerem soluções aproximadas, mas de boa qualidade, em tempo aceitável [Talbi, 2009]. Heurísticas construtivas simples, como as baseadas em ordenação por grau de conectividade dos nós, são frequentemente utilizadas por sua simplicidade e rapidez, mas podem gerar soluções subótimas. Em contrapartida, metaheurísticas, como *Simulated Annealing* (SA) [Kirkpatrick et al., 1983], *Tabu Search* (TS) [Glover, 1989; Hansen, 1986] e *Greedy Randomized Adaptive Search Procedure* (GRASP) [Feo e Resende, 1995], buscam explorar o espaço de soluções de maneira sofisticada, incorporando técnicas que visam escapar de ótimos locais e obter colorações úteis na prática. Soma-se a isto o fato de que as metaheurísticas vêm sendo aprimoradas ao longo dos anos e, atualmente, seus desempenhos têm-se mostrado competitivos em relação a heurísticas mais simples [Talbi, 2009].

A literatura sobre alocação de registradores com uso de metaheurísticas concentra-se ma-

goritmicamente em algoritmos genéticos e variantes evolutivas [Kri e Feeley, 2004; Lintzmayer et al., 2012; Terci et al., 2024], colônia de formigas [Lintzmayer et al., 2011] e abordagens híbridas [Mulati et al., 2015]. No entanto, não foram encontrados estudos que explorem sistematicamente a aplicação de GRASP, SA ou TS para o problema da alocação de registradores, tampouco investigações que comparem diretamente o desempenho entre diferentes metaheurísticas sob o mesmo conjunto de instâncias e critérios de avaliação de desempenho. Isso revela uma lacuna interessante da literatura, haja vista que tais métodos têm histórico consolidado de bom desempenho em problemas combinatórios clássicos, como coloração de grafos, *scheduling* e roteamento [Kri et al., 2005].

Neste contexto, este trabalho apresenta um estudo experimental comparativo entre GRASP, SA e TS, aplicados ao problema de alocação de registradores modelado como um problema de coloração de grafos, com experimentação em instâncias padronizadas do conjunto DIMACS [Johnson e Trick, 1996].

Os resultados mostraram que a TS foi, em média, a técnica mais eficaz na redução de conflitos em grafos densos, enquanto o SA demonstrou melhor desempenho em grafos mais regulares e esparsos, com tempos de execução inferiores aos da TS. O GRASP, por sua vez, apresentou desempenho competitivo em instâncias específicas e destacou-se pelo menor tempo médio de execução entre os métodos avaliados. Essas evidências sugerem que a escolha da metaheurística mais adequada depende do perfil estrutural do grafo e das restrições de tempo de execução, indicando um interessante *trade-off* entre qualidade da solução e custo computacional.

2. Referencial Teórico

Uma das etapas mais importantes da compilação de um programa é a alocação de registradores. Um registrador é um componente de memória que pode ser acessado pela CPU com maior velocidade do que o acesso à memória principal a preço de, também, ser uma memória mais cara [Stallings, 2010]. Logo, seu uso eficiente, que fica a cargo de uma função de alocação (alocador), é fundamental para o desempenho em tempo de execução do código compilado [Fischer et al., 2010].

Um alocador tem a responsabilidade de definir, em tempo de compilação, o destino de cada variável manipulada pelo programa. Essencialmente, dois desfechos são possíveis: manter a variável em um registrador ou transferi-la para a memória principal. Para modelar quais variáveis não podem ocupar o mesmo registrador simultaneamente, constrói-se um grafo de interferência. Neste grafo, cada nó representa uma variável (ou, mais precisamente, um intervalo de vida da variável) e uma aresta conecta dois nós se as variáveis correspondentes estão “vivas” (ativas) ao mesmo tempo e, portanto, interferem uma na outra [Chaitin et al., 1981].

A tarefa do alocador, sob essa ótica, equivale a colorir este grafo utilizando um número de k cores igual ao número de registradores disponíveis. Caso a quantidade de registradores (cores) não seja suficiente para colorir o grafo sem conflitos (isto é, sem que nós adjacentes tenham a mesma cor), o que significa que não é possível acomodar todas as variáveis ativas concorrentes apenas nos registradores, o compilador insere operações adicionais de acesso à memória para gerenciar o excedente. A esta operação dá-se o nome de *spilling* (ou derramamento) [Chaitin, 1982]. Assim, o objetivo do alocador é utilizar de forma eficiente o conjunto de registradores disponíveis, minimizando as custosas operações de *spilling* [Cooper e Torczon, 2023]. Contudo, a alocação ótima de registradores (e consequentemente a coloração ótima do grafo de interferência) é um problema NP-Completo, o que o torna computacionalmente uma tarefa altamente desafiadora [Sethi, 1975].

Diante dessa complexidade, abordagens aproximadas são fundamentais. As metaheurísticas representam uma classe de métodos computacionais desenvolvidos para lidar com problemas de alta complexidade, especialmente os classificados como NP-Difíceis [Talbi, 2009].

Uma característica importante das metaheurísticas é a sua flexibilidade. Durante o processo de busca por soluções, elas podem operar tanto com soluções completas (que atendem todas

as condições do problema) quanto com soluções parciais (que resolvem apenas parte dele) [Osman e Laporte, 1996]. Além disso, metaheurísticas permitem a integração de outros métodos, quer sejam eles simples ou complexos, formando abordagens híbridas mais eficazes [Voss et al., 1999]. Essa capacidade de adaptação contribui para a diversificação das estratégias de busca e para o aprofundamento em regiões promissoras do espaço de soluções, reduzindo a chance de o algoritmo ficar preso em ótimos locais [Blum e Roli, 2003].

A seguir são apresentadas as metaheurísticas implementadas e analisadas neste trabalho.

2.1. GRASP

O GRASP é uma metaheurística do tipo *multi-start* desenvolvida por Feo e Resende [1995], que combina estratégias semi-gulosas com mecanismos de aleatoriedade para explorar eficientemente o espaço de soluções S . Seu funcionamento baseia-se em dois componentes principais: uma fase construtiva aleatorizada e uma fase de busca local intensificadora [Resende e Ribeiro, 2003].

Durante a fase construtiva, o algoritmo gera uma solução viável utilizando um critério guloso adaptativo, mas de forma não determinística. Isso é feito por meio de uma Lista Restrita de Candidatos (LRC), composta por elementos que apresentam boas avaliações segundo um critério de otimização. A escolha do próximo elemento a ser incluído na solução é feita aleatoriamente entre os membros da LRC, permitindo diversificação e evitando soluções repetitivas [Talbi, 2009; Goldbarg, 2015].

Em seguida, na fase de busca local, o algoritmo aplica modificações sistemáticas na solução construída, visando aprimorar seu valor na função objetivo. Essa fase é determinística e explora a vizinhança da solução atual, realizando trocas, inserções, remoções ou outras operações específicas do problema até que nenhum ganho adicional possa ser obtido — caracterizando um ótimo local [Blum e Roli, 2003]. Esse processo é repetido várias vezes com diferentes pontos de partida gerados pela fase construtiva, permitindo ao GRASP equilibrar exploração global e intensificação local, tornando-o adequado para resolver problemas combinatórios complexos com eficiência.

2.2. Busca Tabu

Proposta por Glover [1989] e aprimorada por Hansen [1986], a TS tem sido amplamente aplicada e estendida, destacando-se por sua eficácia na exploração inteligente do espaço de soluções. Um de seus principais diferenciais é o uso de uma memória adaptativa, que registra características de soluções já visitadas, a fim de evitar ciclos e estagnações em ótimos locais [Glover e Laguna, 1997; Osman e Laporte, 1996].

O termo tabu refere-se à ideia de proibir certos movimentos ou ações durante a busca, evitando que a solução volte a um ponto que já tenha sido explorado [Blum e Roli, 2003], reduzindo a possibilidade de estagnação ou ciclagem de soluções. Para isso, a TS utiliza uma estrutura de memória de curto prazo, a lista tabu, que armazena soluções recentemente visitadas [Osman e Laporte, 1996], restringindo-as. Assim, reduz-se o esforço computacional desperdiçado [Goldbarg, 2015].

O processo inicia com uma solução s do conjunto de soluções viáveis S . Em seguida, gera uma lista de soluções candidatas, escolhendo a melhor entre elas. Então, verifica se a condição de parada foi atingida; se não, atualiza a Lista Tabu (que proíbe temporariamente certas soluções) e as condições de aspiração (que permitem aceitar soluções tabus em casos específicos). Esse ciclo prossegue até que a condição de parada seja satisfeita, momento em que o algoritmo apresenta a melhor solução encontrada [Blum e Roli, 2003].

2.3. Simulated Annealing

Proposta por Kirkpatrick et al. [1983], esta metaheurística se baseia na analogia com o aquecimento e resfriamento controlado de materiais sólidos, que permite ao material atingir um

estado de mínima energia estrutural [Talbi, 2009; Goldberg, 2015]. No processo físico, o material é inicialmente aquecido a uma alta temperatura, o que proporciona energia suficiente para que os átomos se movimentem livremente, explorando diferentes configurações. À medida que a temperatura diminui gradualmente, essas partículas tendem a se organizar em arranjos mais estáveis e de menor energia. De maneira análoga, no SA, o espaço de soluções S de um problema é explorado estocasticamente, permitindo, inclusive, a aceitação temporária de soluções piores, para escapar de ótimos locais e alcançar soluções globais melhores [Blum e Roli, 2003].

O SA é uma metaheurística que opera iterativamente, começando com uma temperatura inicial alta que é gradualmente reduzida ao longo da busca. A velocidade dessa redução é controlada por uma taxa de resfriamento (frequentemente um fator α , $0 < \alpha < 1$). A escolha desta taxa é crucial: um valor de α muito menor que 1 (resultando em resfriamento rápido) pode limitar a exploração do espaço de soluções, aumentando o risco de o algoritmo convergir prematuramente para um ótimo local [Talbi, 2009; Goldberg, 2015]. Por outro lado, um α próximo de 1 promove um resfriamento lento, permitindo uma exploração mais ampla.

O aceite, ou não, de soluções piores segue a distribuição de Boltzmann [Van Laarhoven e Aarts, 1987], dada por $P(\Delta E, Temp) = e^{\frac{-\Delta E}{Temp}}$, em que ΔE é a diferença da energia (qualidade) das soluções avaliadas, e $Temp$ é a temperatura atual do sistema. Temperaturas mais altas tendem a permitir maior aceitação de soluções piores, enquanto temperaturas mais baixas são mais restritivas.

3. Implementação e Experimentos Computacionais

Os resultados deste trabalho são constituídos de três partes: 1) modelagem e implementação, 2) experimentos computacionais e 3) avaliação dos resultados, conforme descritas a seguir.

3.1. Modelagem e Implementação

As metaheurísticas foram implementadas na linguagem de programação C++ e compiladas com gcc++ 11.4.0, utilizando o padrão C++17 e parâmetro de otimização -O3. Esta linguagem foi escolhida por seu desempenho e controle refinados sobre o uso de memória. A linguagem Python, versão 3.10.12, foi utilizada para a análise estatística dos resultados e visualização dos dados, com o auxílio das bibliotecas NumPy, Pandas, Matplotlib e SciPy. Todos os experimentos computacionais foram conduzidos em uma máquina equipada com processador Intel® Core™ i7-7700HQ, 16GB de RAM e sistema operacional Pop!_OS 22.04 LTS, uma distribuição Linux voltada para produtividade e desempenho em tarefas computacionais, especialmente em ambientes de desenvolvimento, devido à sua estabilidade, suporte nativo a ferramentas de programação e gerenciamento eficiente de recursos.

A escolha dos parâmetros usados nas metaheurísticas foi fundamentada em valores clássicos da literatura, como os sugeridos por Feo e Resende [1995] para GRASP, Glover [1989] para TS e Kirkpatrick et al. [1983] para SA, e complementada por ajustes empíricos. Esses ajustes buscaram equilibrar a qualidade das soluções e o custo computacional. Em particular, os limites de iterações, tamanhos das listas (Tabu e Restrita) e taxas de resfriamento foram definidos após testes preliminares, garantindo desempenho competitivo dentro de tempos de execução razoáveis.

Para inicializar cada execução, foi gerada uma solução inicial viável por meio de uma heurística construtiva simples: percorre-se sequencialmente a lista de vértices e atribui-se a cada um a menor cor disponível que não viole as restrições de adjacência. Por exemplo, considere um grafo com quatro vértices $V = \{v_1, v_2, v_3, v_4\}$ e arestas $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}\}$, com três cores disponíveis $\{1, 2, 3\}$. Uma solução inicial viável poderia ser a atribuição $(v_1 \rightarrow 1, v_2 \rightarrow 2, v_3 \rightarrow 1, v_4 \rightarrow 3)$, que respeita a restrição de que vértices adjacentes não podem receber a mesma cor. Essa solução inicial serve como ponto de partida para as metaheurísticas, que então buscam melhorias por meio de seus operadores de busca específicos.

3.1.1. GRASP

Na fase construtiva do GRASP, a LRC é controlada por um fator de aleatoriedade α , que varia entre 0 e 1, na qual valores mais baixos tornam a construção mais gulosa, enquanto valores mais altos favorecem a diversidade. O método recebe como parâmetros o grafo, o número máximo de cores, o número máximo de iterações T e o fator de aleatoriedade. No experimento, adotou-se $T = 200$, buscando equilíbrio entre a qualidade da solução e o custo computacional, e $\alpha = 0,1$, promovendo variedade sem comprometer a eficiência. Quando a solução inicial apresenta conflitos, um mecanismo de *spilling* é ativado para tentar uma nova alocação mais eficaz. Em seguida, aplica-se uma fase de busca local, que tenta reduzir o número de conflitos por meio de trocas ou recolorações locais, otimizando a solução antes da próxima iteração.

3.1.2. Tabu Search

A TS inicia com uma solução fornecida pelo método guloso. Seus parâmetros incluem o grafo, a solução inicial, o número máximo de cores, o número máximo de iterações e o tamanho da lista tabu. Os valores definidos para os dois últimos foram número de iterações $T = 1000$ e tamanho da lista tabu $T_{TABU} = 30$. A vizinhança é gerada por uma função que considera o grafo, a solução atual e o limite de cores. O critério de movimento na vizinhança consiste na troca da cor de um único vértice. Em cada iteração da geração de vizinhança, são sorteados aleatoriamente até 50 vértices distintos, e para cada um tenta-se atribuir uma nova cor aleatória diferente da atual. A qualidade de cada possível troca é avaliada com base na variação no número de conflitos gerado por ela. A troca que resulta na maior redução de conflitos (ou no menor aumento, caso nenhuma redução seja possível) é aplicada, gerando assim a solução vizinha a ser considerada na iteração atual da busca tabu. Portanto, soluções vizinhas com menos conflitos substituem a atual. Assim, sempre que uma nova melhor solução é encontrada, ela é armazenada como solução global.

3.1.3. Simulated Annealing

Além do grafo, da solução inicial s_0 fornecida pelo método guloso, do grafo e do limite de cores disponíveis, o SA foi configurado com os seguintes parâmetros: temperatura inicial $T_0 = 1000$, taxa de resfriamento $\alpha = 0,995$ e número máximo de iterações $T_{max} = 1000$. Esses parâmetros permitem uma fase inicial de ampla exploração, com resfriamento gradual para intensificar a busca à medida que a temperatura diminui. O algoritmo pode encerrar antecipadamente caso encontre uma solução sem conflitos antes de atingir T_{max} . A geração de novas soluções é estocástica, em que, em cada iteração, um único vértice é escolhido aleatoriamente e recebe uma nova cor também aleatória. Essa escolha introduz aleatoriedade controlada no processo de busca, característica central do SA, permitindo escapar de ótimos locais.

3.2. Experimentos Computacionais

O objetivo principal dos experimentos foi avaliar a eficácia dos algoritmos implementados em termos da capacidade de minimizar conflitos na coloração, visando reduzir o número de *spills* o que, no contexto de alocação de registradores, reflete diretamente na eficiência da alocação.

Para avaliar a qualidade dos algoritmos implementados, foi realizada uma série de experimentos usando instâncias de teste do conjunto DIMACS. Este é um *benchmark* consolidado na literatura para a avaliação de algoritmos de Coloração de Grafos [Johnson e Trick, 1996]. A Tabela 1 ilustra as instâncias utilizadas nos experimentos, bem como suas principais características: nome da instância, grupo ao qual ela pertence, número de vértices, número de arestas e o menor número de cores conhecido na literatura (denotado por K). Este conjunto de instâncias reflete diferentes cenários, abrangendo desde instâncias sintéticas, criadas para testar comportamentos específicos dos algoritmos, até instâncias que refletem aplicações reais, como os grafos de interferência entre variáveis em compiladores.

Para garantir uma análise abrangente, foram selecionadas instâncias de diferentes famílias presentes no conjunto DIMACS. DSJ é a família de grafos aleatórios comumente utilizada na avaliação de heurísticas para a coloração [Johnson et al., 1991]. MYC contém grafos derivados da construção de Mycielski, úteis para representar estruturas mais simples e bem definidas [Hongmei, 2006]. LEI contém grafos de tamanho médio (450 vértices), com número cromático k previamente conhecido, utilizados para validação da qualidade da solução.

Tabela 1: Parâmetros das instâncias de teste utilizadas de cada grafo

Instância	Grupo	Vértices	Arestas	K	Densidade (%)
DSJC250.1	DSJ	250	6436	8	20,68
DSJC250.5	DSJ	250	15668	8	50,37
DSJC500.1	DSJ	500	12458	12	9,99
DSJC500.5	DSJ	500	62624	48	50,20
DSJC1000.1	DSJ	1000	49629	20	9,94
le450.5a	LEI	450	5714	5	5,66
le450.5b	LEI	450	5734	5	5,68
le450.5c	LEI	450	9803	5	9,70
le450.5d	LEI	450	9757	5	9,66
le450.15a	LEI	450	8168	15	8,09
le450.15b	LEI	450	8169	15	8,09
le450.15c	LEI	450	16680	15	16,51
le450.15d	LEI	450	16750	15	16,58
le450.25a	LEI	450	8260	25	8,18
le450.25b	LEI	450	8263	25	8,18
le450.25c	LEI	450	17343	25	17,17
le450.25d	LEI	450	17425	25	17,25
myciel3	MYC	55	20	4	1,35
myciel4	MYC	115	71	5	1,08
myciel5	MYC	235	236	6	0,86
myciel6	MYC	475	755	7	0,67
myciel7	MYC	955	2360	8	0,52

No contexto deste trabalho, os grafos foram avaliados considerando um número fixo de $K = 8$ cores, simulando a quantidade de registradores disponíveis em um processador e representando o pior caso para alocação, a fim de avaliar a robustez dos métodos quando sob circunstâncias de limitações severas. Em alguns casos, o número de cores foi insuficiente para alocar os programas de forma ótima, forçando o *spilling*, o que permitiu avaliar a capacidade dos métodos de reduzir a quantidade de *spills* e, simultaneamente, minimizar conflitos de alocação. Embora existam processadores com maior número de registradores, os experimentos se limitaram a esses valores para explorar como as técnicas lidam com situações de escassez de recursos e na otimização do uso de registradores.

Inicialmente, foram realizadas 10 execuções por instância, mas os resultados apresentaram variações maiores, o que indicou que a aleatoriedade dos algoritmos estocásticos estava influenciando de forma significativa a avaliação do desempenho. Com a adoção de 50 execuções por instância, foi possível reduzir as flutuações e obter uma análise mais confiável e representativa dos métodos, garantindo maior precisão nas comparações de desempenho.

4. Resultados e Discussões

A Tabela 2 apresenta os desempenhos médios das três metaheurísticas, mostrando a média de conflitos $\pm$ desvio padrão e o tempo médio de execução $T(s)$. A TS obteve os melhores resultados em 16 das 23 instâncias, evidenciando sua capacidade de escapar de mínimos locais e gerar soluções de maior qualidade. Entretanto, seus tempos de execução foram significativamente maiores, chegando a 1.266 segundos ($\approx$ 21 minutos) em *DSJC250.1* e entre 17 e 18 horas em instâncias maiores, como *DSJC500.5* e *DSJC1000.1*, comprometendo sua viabilidade prática em compiladores.

Tabela 2: Resultados médios com desvio padrão e tempo de execução (em segundos) das metaheurísticas para alocação de registradores ($k = 8$ cores, 50 execuções por instância)

Instância	GRASP		TS		SA		Melhor Método
	Média $\pm$ DP	T(s)	Média $\pm$ DP	T(s)	Média $\pm$ DP	T(s)	
DSJC250.1	27.74 $\pm$ 1.80	15.83	27.58 $\pm$ 1.48	1266.49	27.69 $\pm$ 1.55	759.81	TS
DSJC250.5	122.48 $\pm$ 2.54	137.74	115.65 $\pm$ 2.35	11019.24	119.06 $\pm$ 2.48	6611.15	TS
DSJC500.1	87.62 $\pm$ 3.27	107.28	84.93 $\pm$ 2.62	8609.63	85.48 $\pm$ 2.67	5165.77	TS
DSJC500.5	1122.10 $\pm$ 5.87	782.01	1118.21 $\pm$ 4.60	62723.60	1119.81 $\pm$ 3.58	37634.17	TS
DSJC1000.1	201.14 $\pm$ 5.30	790.98	176.12 $\pm$ 4.75	63439.21	192.74 $\pm$ 4.98	38063.53	TS
le450_5a	68.52 $\pm$ 15.17	84.95	64.38 $\pm$ 11.61	7617.64	64.92 $\pm$ 12.12	4570.55	TS
le450_5b	63.60 $\pm$ 11.17	85.38	62.24 $\pm$ 9.29	7923.48	67.10 $\pm$ 10.75	4754.04	TS
le450_5c	31.14 $\pm$ 7.74	93.53	24.02 $\pm$ 6.39	8075.25	25.36 $\pm$ 6.68	4845.21	TS
le450_5d	30.96 $\pm$ 8.65	100.09	22.42 $\pm$ 7.10	8983.24	22.78 $\pm$ 7.22	5389.93	TS
le450_15a	29.42 $\pm$ 1.19	74.55	26.94 $\pm$ 1.04	6894.33	27.68 $\pm$ 1.08	4136.58	TS
le450_15b	28.32 $\pm$ 1.36	70.85	26.52 $\pm$ 1.12	6504.54	27.04 $\pm$ 1.19	3942.71	TS
le450_15c	236.78 $\pm$ 3.89	202.16	240.88 $\pm$ 3.29	18048.7	234.26 $\pm$ 3.70	10829.22	SA
le450_15d	239.66 $\pm$ 3.66	204.85	243.10 $\pm$ 3.35	18096.5	237.12 $\pm$ 3.57	10857.60	SA
le450_25a	131.82 $\pm$ 1.50	101.60	126.48 $\pm$ 1.32	8064.1	127.02 $\pm$ 1.71	4838.46	TS
le450_25b	128.60 $\pm$ 1.95	88.35	124.87 $\pm$ 1.66	7077.2	126.99 $\pm$ 1.84	4246.32	TS
le450_25c	50.82 $\pm$ 1.63	156.03	48.76 $\pm$ 1.34	12582.76	49.28 $\pm$ 1.42	7549.56	TS
le450_25d	50.58 $\pm$ 1.64	153.28	48.20 $\pm$ 1.50	12281.32	49.00 $\pm$ 1.46	7368.77	TS
myciel3	0.00 $\pm$ 0.000	0.04	0.00 $\pm$ 0.000	0.13	0.00 $\pm$ 0.000	0.06	Empate
myciel4	0.00 $\pm$ 0.000	0.13	0.00 $\pm$ 0.000	0.44	0.00 $\pm$ 0.000	0.25	Empate
myciel5	0.00 $\pm$ 0.000	0.47	0.00 $\pm$ 0.000	1.47	0.00 $\pm$ 0.000	0.85	Empate
myciel6	0.00 $\pm$ 0.000	2.38	0.00 $\pm$ 0.000	7.91	0.00 $\pm$ 0.000	4.19	Empate
myciel7	0.00 $\pm$ 0.000	13.19	0.00 $\pm$ 0.000	43.37	0.00 $\pm$ 0.000	26.29	Empate

Pode-se observar que o GRASP, em termos gerais, apresentou desempenho inferior ao das demais estratégias, tanto na redução de conflitos quanto em termos de desvio padrão. Ainda assim, em algumas instâncias específicas, como *DSJC250.1*, *le450_25c* e *le450_25d*, seu desempenho aproximou-se ao das outras abordagens, sugerindo certa adaptabilidade do método. A TS obteve, na maioria dos casos, os menores números médios de conflitos, mantendo vantagem sobre o GRASP nesse aspecto. No entanto, essa superioridade veio acompanhada de tempos de execução consideravelmente mais altos, muitas vezes superando os das demais metaheurísticas por margens expressivas. Por exemplo, para instâncias como *DSJC250.1*, o tempo de execução da TS foi de 1266.49 segundos, enquanto o GRASP levou apenas 15.83 segundos.

O SA, por sua vez, apresentou desempenho competitivo tanto em qualidade quanto em tempo, destacando-se nas instâncias *le450_15c* e *le450_15d*, nas quais obteve os melhores resultados em número de conflitos. Em geral, a análise conjunta da qualidade da solução e do tempo de execução indica que o SA tende a fortalecer a razão entre eficácia e eficiência computacional, sendo particularmente atrativo em cenários em que o tempo é um fator crítico.

Nas instâncias do grupo DSJ, que apresentam maior densidade de arestas e, portanto, maior complexidade estrutural, os tempos de execução foram significativamente maiores, especialmente nas instâncias DSJC1000.1 e DSJC500.5, nas quais os métodos levaram mais tempo para executar. Em termos de redução de conflitos, a TS se destaca, pois, por exemplo, em DSJC1000.1, o número médio de conflitos foi consideravelmente menor em comparação às outras abordagens, indicando sua maior eficácia na exploração do espaço de soluções. Essa tendência foi observada também na maioria das instâncias LEI, que apresentam diferentes níveis de conectividade. No entanto, para as bibliotecas le450_15c e le450_15d, a TS teve o pior desempenho entre as metaheurísticas avaliadas.

Finalmente, o SA obteve resultados muito próximos aos da TS em relação ao desempenho. Sua capacidade de explorar soluções concorrentes, aceitando temporariamente soluções piores para escapar de mínimos locais, foi fundamental em instâncias como DSJC1000.1 e le450_15d. Embora tenha sido ligeiramente inferior à TS em alguns casos, o SA ainda obteve resultados competitivos, especialmente em instâncias de menor densidade e mais regulares.

Portanto, é possível observar que o tamanho e a densidade dos grafos têm um impacto significativo no desempenho dos algoritmos, tanto em termos de qualidade quanto de tempo de execução. Em grafos densos como DSJC500.5 e DSJC1000.1, as instâncias apresentaram um número maior de conflitos devido ao elevado número de adjacências, o que aumenta a chance de sobreposição de cores. Além disso, as metaheurísticas levaram mais tempo para serem executadas nessas instâncias, com tempos de execução consideravelmente mais altos. Em contrapartida, grafos como o Myciel, que são altamente estruturados, não apresentaram conflitos, já que suas características favorecem soluções ótimas, mesmo em cenários restritivos. Esses grafos também exigiram tempos de execução muito menores, destacando a eficiência das metaheurísticas quando aplicadas a problemas mais estruturados.

A Figura 1 apresenta a comparação do desempenho médio das metaheurísticas em termos de conflitos e seus respectivos tempos médios de execução. Para facilitar a análise visual, as instâncias foram agrupadas em três conjuntos: DSJ, LEI e MYC. Cada grupo é exibido em um gráfico separado, com escalas verticais independentes, já que os valores de conflitos variam significativamente entre os conjuntos, o que inviabilizaria uma comparação adequada em um único eixo compartilhado. A escala logarítmica foi utilizada para os tempos de execução, em segundos, permitindo uma melhor visualização das diferenças entre as metaheurísticas, especialmente quando há variações de ordem de grandeza.

Observa-se que a TS apresenta desempenho superior na maioria das instâncias, especialmente nos grafos mais densos do grupo DSJ, enquanto SA mostra competitividade em casos específicos, e GRASP tende a apresentar resultados inferiores na média. Quando o número de cores disponíveis não é suficiente para eliminar todos os conflitos, o alocador recorre a um algoritmo de *spilling*. Nesses cenários, o desempenho das metaheurísticas também se reflete na capacidade de minimizar o número de *spillings*, além dos próprios conflitos de alocação.

A análise dos resultados indica que a TS obteve, em média, soluções com menor número de cores em instâncias maiores, superando GRASP e SA em 70% dos casos com mais de 100 vértices. No entanto, em termos de tempo computacional, o GRASP destacou-se como a abordagem mais eficiente, apresentando os menores tempos médios de execução em todas as instâncias, inclusive nas de maior porte, como DSJC1000.1 e DSJC500.5. O SA, por sua vez, mostrou desempenho intermediário, sendo mais rápido que a TS, mas ainda mais lento que o GRASP.

Observa-se, portanto, um *trade-off* entre qualidade da solução e tempo de execução, na qual a TS é preferível quando o objetivo é minimizar o número de registradores alocados, enquanto o GRASP se mostra mais vantajoso em aplicações onde a velocidade de resposta é mais crítica. O

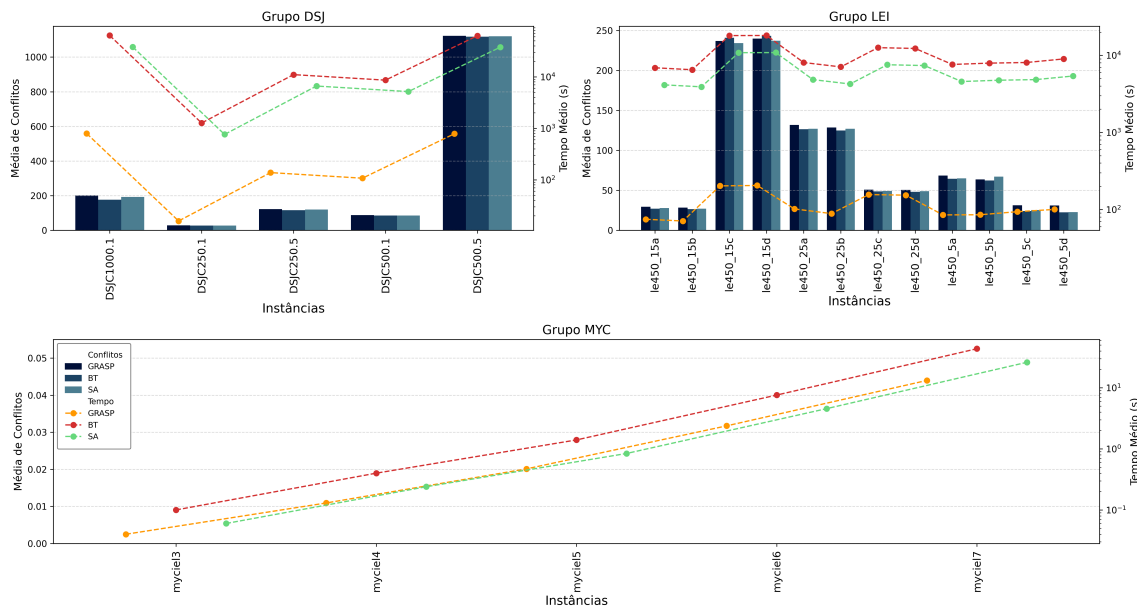


Figura 1: Comparação de desempenho e tempo médio entre os métodos de coloração de grafos

SA pode ser visto como uma alternativa intermediária, com qualidade de solução próxima à do TS, mas com um custo computacional significativamente menor do que o da própria TS. Além disso, é possível notar que o desempenho relativo das metaheurísticas varia de acordo com características estruturais dos grafos, já que a TS se destacou em grafos densos como os da família DSJ, enquanto o GRASP se mostrou altamente competitivo em grafos menores ou mais regulares, como os da família MYC.

4.1. Análise Estatística dos Resultados

Para verificar a significância estatística das diferenças observadas entre as metaheurísticas, aplicou-se inicialmente o teste de Friedman sobre os resultados de número de conflitos obtidos nas instâncias analisadas. O valor da estatística foi $H = 21,53$ com $p\text{-value} < 0,0001$, indicando que há diferenças estatisticamente significativas entre pelo menos dois dos métodos comparados.

Na sequência, aplicaram-se testes de Wilcoxon pareados para comparações par a par entre os algoritmos. Os resultados são apresentados na Tabela 3.

Tabela 3: Resultados dos testes estatísticos de comparação das metaheurísticas

Comparação	Estatística (W)	p-value	Significância
GRASP vs SA	1,000	0,0004	Sim
GRASP vs TS	18,000	0,0056	Sim
SA vs TS	31,000	0,0312	Sim

Tais resultados confirmam que as diferenças observadas nas soluções encontradas pelas metaheurísticas são estatisticamente significativas ao nível de 5%. Em particular, o GRASP apresentou desempenho inferior ao TS e ao SA, com diferenças robustas nos testes. Já entre SA e TS, embora a diferença seja menor, ainda assim é estatisticamente significativa.

Essas evidências sugerem que a **TS** é a técnica mais eficaz entre as avaliadas, tanto em termos de qualidade da solução (número de cores), quanto por apresentar diferenças estatisticamente

relevantes em relação às demais abordagens.

5. Conclusão

Este trabalho avaliou o uso das metaheurísticas GRASP, TS e SA na alocação de registradores, modelada como um problema de coloração de grafos. A proposta buscou simular cenários com restrição severa de registradores, utilizando instâncias do conjunto DIMACS sob um limite fixo de cores.

As metaheurísticas foram implementadas e testadas sobre instâncias do benchmark DIMACS, sob um limite fixo de 8 cores, representando um caso crítico para alocação. Os experimentos revelaram que a Busca Tabu apresentou, em média, os melhores resultados em termos de minimização de conflitos, especialmente em grafos mais densos, como os da família DSJ. Por outro lado, o SA mostrou-se competitivo, oferecendo um bom equilíbrio entre qualidade da solução e tempo de execução, com destaque em grafos regulares e menos conectados. Já o GRASP se destacou pelo baixo tempo de execução, sendo a abordagem mais eficiente em termos computacionais, embora com desempenho inferior na qualidade das soluções em cenários mais complexos.

A análise estatística confirmou que as diferenças entre os métodos são significativas, reforçando que não há uma técnica universalmente superior para todos os casos. Em vez disso, os resultados apontam para um *trade-off* claro entre eficácia e eficiência computacional, sendo a TS preferível quando a prioridade é minimizar conflitos e o GRASP mais indicado quando o tempo de resposta é fator crítico. O SA surge como uma solução intermediária promissora.

Referências

- Beckitt-Marshall, J. (2021). Improving energy efficiency through compiler optimizations. Honors Projects, Bowdoin College.
- Blum, C. e Roli, A. (2003). Metaheuristics in combinatorial optimization. *ACM Computing Surveys (CSUR)*, 35(3):268–308.
- Chaitin, G. J. (1982). Register allocation & spilling via graph coloring. *ACM Sigplan Notices*, 17(6):98–101.
- Chaitin, G. J., Auslander, M. A., Chandra, A. K., Cocke, J., Hopkins, M. E., e Markstein, P. W. (1981). Register allocation via coloring. *Computer languages*, 6(1):47–57.
- Cook, S. (2006). The p versus np problem. *The millennium prize problems*, p. 87–104.
- Cooper, K. e Torczon, L. (2023). *Engineering a Compiler*. Elsevier, Cambridge.
- Feo, T. e Resende, M. (1995). Greedy randomized adaptive search procedures. *J Glob Optim*, 6: 109–133.
- Fischer, C., Cytron, R., e LeBlanc, R. (2010). *Crafting a Compiler*. Pearson Education, Boston.
- Glover, F. e Laguna, M. (1997). *Tabu Search*. Springer New York, NY.
- Glover, F. (1989). Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206.
- Goldbarg, E. (2015). *Otimização Combinatória e Meta-heurísticas - Algoritmos e Aplicações*. GEN LTC.

- Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on numerical methods in combinatorial optimization, Capri, Italy*, p. 70–145.
- Hongmei, L. (2006). Circular chromatic number and mycielski graphs. *Mathematics and Computer Science*, 32(6):673–681.
- Johnson, D. S. e Trick, M. A. (1996). *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc.
- Johnson, D. et al. (1991). Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406.
- Kirkpatrick, S., Gelatt, C., e Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220 (4598):671–680.
- Kri, F. e Feeley, M. (2004). Genetic instruction scheduling and register allocation. In *XXIV international conference of the chilean computer science society*, p. 76–83. IEEE.
- Kri, F., Gómez, C., e Caro, P. (2005). Overview of metaheuristics methods in compilation. In *Mexican International Conference on Artificial Intelligence*, p. 483–493. Springer.
- Lintzmayer, C. N., Mulati, M. H. d. S., e Faustino, A. (2011). Register allocation with graph coloring by ant colony optimization. In *2011 30th International Conference of the Chilean Computer Science Society*, p. 247–255. IEEE.
- Lintzmayer, C. et al. (2012). Register allocation by evolutionary algorithm. In *2012 31st International Conference of the Chilean Computer Science Society*, p. 209–215.
- Mulati, M. H., Lintzmayer, C. N., e da Silva, A. F. (2015). The hybrid colorant-rt algorithms and an application to register allocation. *Inteligencia Artif.*, 18(55):81–111.
- Osman, I. e Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63:511–623.
- Quintão Pereira, F. M. e Palsberg, J. (2008). Register allocation by puzzle solving. *SIGPLAN Not.*, 43(6):216–226.
- Resende, M. G. C. e Ribeiro, C. C. (2003). *Greedy Randomized Adaptive Search Procedures*, p. 219–249. Springer US.
- Sethi, R. (1975). Complete register allocation problems. *SIAM journal on Computing*, 4(3):226–248.
- Stallings, W. (2010). *Arquitetura e Organização de Computadores*. Pearson Prentice Hall.
- Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons.
- Terci, G. S., Abdulhalik, E., Bayrakci, A. A., e Boz, B. (2024). Bitea: Bitvertex evolutionary algorithm to enhance performance for register allocation. *IEEE Access*, 12:115497–115514.
- Van Laarhoven, P. J. e Aarts, E. H. (1987). Simulated annealing. In *Simulated annealing: Theory and applications*, p. 7–15. Springer.
- Voss, S. et al. (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Springer Science & Business Media.