Brazilian Journal of Development

Um Algoritmo Híbrido para o Problema da Clique Máxima

A Hybrid Algorithm for the Maximum Clique Problem

DOI:10.34117/ bjdv6n6-275

Recebimento dos originais: 08/05/2020
Aceitação para publicação: 11/06/2020

Murilo Oliveira Machado
Mestre em Informática pela Universidade Federal do Rio de Janeiro (UFRJ)
Professor da Universidade Federal do Mato Grosso do Sul
Avenida Rio Branco, 1.270-Bairro Universitário-Corumbá-MS, cep:79.308-902
E-mail: mmbadas@gmail.com

Thiago Soares Marques
Mestre em Sistemas e Computação pela Universidade Federal do Rio Grande do Norte (UFRN)
UFRN-Campus Universitário, Lagoa Nova-Natal-RN, cep:59.072-970
E-mail: thiago@ppgsc.ufrn.br

Elizabeth Ferreira Gouvea Goldbarg
Doutora em Engenharia de Produção pela COPPE-Produções (UFRJ)
Professora da Universidade Federal do Rio Grande do Norte
UFRN-Campus Universitário, Lagoa Nova-Natal-RN, cep:59.072-970
E-mail: beth@dimap.ufrn.br

Marcos Cesar Gouvea Goldbarg
Pós-Doutor pela Universidade Federal de Minas Gerais (UFMG)
Professor da Universidade Federal do Rio Grande do Norte
UFRN-Campus Universitário, Lagoa Nova-Natal-RN, cep:59.072-970
E-mail: gold@dimap.ufrn.br

RESUMO
Este artigo apresenta um algoritmo que realiza a hibridização entre a meta-heurística Simulated Anneling e uma Lista Tabu para resolver o problema da clique máxima. O algoritmo foi avaliado mediante os resultados obtidos para o banco de instâncias do Centro de matemática discreta e ciência da computação teórica (DIMACS). O algoritmo consegue processar um total de 83% das instâncias em menos de 2 minutos e obtém o valor ótimo para 74,6%. Os resultados mostraram-se promissores em contraste com as observações realizadas na literatura. Em comparação com o resultado recente publicado, a média da solução do algoritmo aqui apresentado foi estritamente melhor em 52,11% e empataram em 21,1% das instâncias.

Palavras-chave: Problema da Clique Máxima, Simulated Anneling, Lista Tabu, Algoritmos Híbridos

ABSTRACT
This article presents an algorithm that performs the hybridization between the Simulated Anneling metaheuristic and a Taboo List to solve the problem of maximum clique. The algorithm was evaluated using the results obtained for the Instance Bank of the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). The algorithm is able to process a total of 83% of

Brazilian Journal of Development

instances in less than 2 minutes and obtains the optimal value for 74.6%. The results were promising in contrast to the observations made in the literature. In comparison with the recent published result, the average of the algorithm solution presented here was strictly better in 52.11% and tied in 21.1% of the instances.

Keywords: Maximum Clique Problem, Simulated Annealing, Taboo List, Hybrid Algorithms

1 INTRODUÇÃO

Considere $G(N, E)$ um grafo finito, onde N representa o conjunto de vértices e E o conjunto de arestas. Uma clique é um subgrafo completo de G. Decidir se, em um grafo, existe uma clique de tamanho maior ou igual a uma constante k é um dos 21 problemas NP-completos apresentados em [1]. O problema de otimização associado é o de encontrar a clique de cardinalidade máxima em G. Segundo [2], o Problema da Clique Máxima (PCM) é um dos problemas mais importantes da matemática discreta. Além disso, é um dos problemas combinatórios NP-difíceis mais estudados [3]. Existem diversas aplicações práticas do PCM, dentre elas redes sem fio [4], [5], [6], processamento de vídeos [7], modelos para problemas relacionados ao câncer [8] e computação quântica [9], [10].

Em [11] os autores relatam que a importância do Problema da Clique Máxima (PCM) levou o Centro de Matemática Discreta e Ciência da Computação (DIMACS) a realizar um concurso com essa temática, e que desde então houve um crescimento de publicações abordando o PCM. A revisão bibliográfica realizada em [3] destaca que nas duas últimasdécadas fora detectado um aumento significativo de trabalhos relacionados a este problema e suas variações. Em [12] são descritos os trabalhos apresentados na competição realizada pelo DIMACS.

A resolução de maneira exata de problemas NP-completos pode levar um tempo computacional inadmissível para instâncias de maior porte, assim, algoritmos heurísticos vêm se mostrando uma alternativa mais adequada no desafio de resolver problemas dessa natureza. Os algoritmos heurísticos de maior sucesso em relação à qualidade da solução obtida são baseados em buscas locais, tais como busca local reativa [13], k-opt [14] e busca local com cooperação [15]. Os trabalhos de [3] e [16] reportam que os melhores algoritmos heurísticos para o PCM são fundamentados em busca local, utilizando os operadores de adicionar, remover e trocar vértices. Os trabalhos [17], [18] e [19] utilizam os graus dos vértices como informação para construírem soluções. Os trabalhos [20] e [21] utilizam a heurística DSATUR como base para encontrar uma melhor delimitação do espaço de busca através da ordenação dos vértices do grafo.

Brazilian Journal of Development

Dentre os algoritmos heurísticos propostos para o PCM, muitos são baseados em abordagens meta-heurísticas. Alguns exemplos são: Busca Tabu [22], [11], [23], [24], Colônia de Formigas [25], Simulated Annealing [26] e Busca Harmônica [16].

A hibridização de meta-heurísticas, embora aplicada com sucesso a diversos problemas [27], [28], [29], [30], não é uma técnica amplamente utilizada para o PCM. Este trabalho tem por objetivo, a apresentação de um algoritmo híbrido para este problema. O algoritmo desenvolvido é baseado na técnica Simulated Annealing e utiliza uma lista tabu, além de uma adaptação do algoritmo CP proposto por [31].

A seção 2 apresenta o problema investigado. A seção 3 apresenta uma revisão dos principais trabalhos que abordaram o PCM. A seção 4 apresenta o algoritmo híbrido que foi implementado. Na seção 5 é descrita a metodologia para a realização dos experimentos. Os resultados obtidos são apresentados na seção 6, onde são comparados a resultados recentes publicados na literatura. Finalmente, as conclusões deste trabalho são apresentadas na seção 7.

2 PROBLEMA DA CLIQUE MÁXIMA

Uma clique é definida como um subgrafo completo de um grafo $G(N, E)$ finito, onde N representa o conjunto de vértices e E o conjunto de arestas. Mais precisamente, uma clique de G é um subgrafo $G'(N', E')$ de G tal que $N' \subseteq N, E' \subseteq E$ e existe aresta $(i, j) \in E'$ para todo par de vértices $i, j \in N'$. A clique máxima $\omega(G)$ de um grafo G, é aquela com o maior número de vértices dentre todas as cliques de G.

O modelo proposto em [17] é apresentado nas expressões (1)-(3). Ele utiliza o conceito de grafo complemento. O complemento de um grafo simples G, denotado por, é $\overline{G}(N, \overline{E})$ um grafo simples que possui o mesmo conjunto de vértices de G, e dois vértices são adjacentes em $\overline{G}$ se não são em G. Os parâmetros da formulação são: n, o número de vértices de G, e $\overline{E}$, o conjunto de arestas de $\overline{G}$. A variável binária de decisão x_i, recebe valor 1 se o *i-ésimo* vértice está na clique, caso contrário, x_i recebe valor 0.

$$Maximeze \sum_{i=1}^{n} x_i \quad (1)$$

Sujeita a:

$$x_i + x_j \leq 1, \forall (i, j) \in \overline{(E)} \quad (2)$$

$$x_i \in 0,1, i = 1, \ldots, n \quad (3)$$

A função objetivo (1) maximiza o número de vértices da clique. A restrição (2) garante que cada par de vértices da solução esteja ligado por uma aresta e a condição (3) denota que as variáveis x_i são binárias.

3 TRABALHOS RELACIONADOS

Esta seção apresenta uma revisão dos principais algoritmos exatos e meta-heurísticos utilizados para resolver o PCM, seguindo uma ordem cronológica e separando-os pelos temas abordados. Neste artigo não será apresentado algoritmos heurísticos. As seções 3.1 e 3.2 descrevem, respectivamente, os trabalhos que aplicaram meta-heurísticas e algoritmos exatos ao PCM.

3.1 ALGORITMOS META-HEURÍSTICOS

Um dos primeiros trabalhos a utilizar a técnica de busca tabu para o *problema do conjunto independente máximo*, problema equivalente ao PCM, foi apresentado por [32]. Dado um grafo G, o *problema do conjunto independente máximo* consiste em encontrar o maior subconjunto S de N, tal que dois vértices de$S$não são adjacentes. Uma solução do algoritmo é uma partição do conjunto N,$s = (S, S')$, tal que $|S| = k$, onde k é um parâmetro de entrada e um limite superior para o número de estabilidade do grafo. O algoritmo admite que S não seja um conjunto independente e minimiza o número de arestas entre seus vértices. Uma solução s' é vizinha de s se ela difere de s por um par de vértices (x, y), tal que $s' = (S \setminus \{x\} \cup \{y\}, S' \setminus \{y\} \cup \{x\})$. É criada uma lista I de pares (x, y) e o benefício de cada par é avaliado. Quanto menor o número de arestas no conjunto $\{S \setminus \{x\} \cup \{y\}\}$, melhor a avaliação do par (x, y). A cada iteração, o algoritmo seleciona o melhor par que não tenha o *status tabu*. Três listas *tabu* são utilizadas para verificar se o movimento de s para s' pode ser feito. A primeira lista guarda as últimas soluções geradas pelo algoritmo. A segunda guarda os últimos vértices inseridos e, por fim, a terceira lista guarda os últimos vértices removidos.

Em [23] foram apresentadas uma *busca tabu* determinística e uma *busca tabu* probabilística. Na construção da clique através do procedimento determinístico, são utilizadas duas *listas tabu*: T_1 e T_2. T_1 lista as t_1 últimas soluções visitadas e T_2 contém os últimos t_2 vértices retirados de uma solução, onde t_1 e t_2 são parâmetros fornecidos na entrada do algoritmo. No procedimento determinístico é priorizado o vértice com maior adjacência à clique C. A regra probabilística, insere aleatoriamente na clique C um vértice não contido na lista tabu. Caso não seja possível aumentar a clique C, são removidos de forma aleatória k vértices de C, onde k é um parâmetro fornecido na entrada do algoritmo. Em [33] a restrição da *lista tabu* foi modificada, de forma que, cada vez que um vértice $u \in C$ é trocado por outro vértice $v \in N \setminus \{C\}$, u e v são adicionados à *lista tabu*, sendo

que, o vértice v fica proibido I_v iterações de se sair da clique e o vértice u fica proibido I_u iterações de entrar na clique, onde $I_u = 0{,}3 * |N| \setminus \{C\}$ e $I_v = 0{,}3 * |C|$.

Em [34] são estudadas três heurísticas gulosas adaptativas com múltiplos começos (AI, AW e NA). A adaptação de cada estratégia está associada aos pesos que são atribuídos aos vértices, bem como ao tamanho da clique obtida no decorrer de cada busca. Se uma clique maior que as anteriores é encontrada, são aumentados os pesos dos vértices pertencentes a nova clique, caso contrário, é diminuído o peso destes vértices, tal que, essas variações dos pesos são proporcionais ao tamanho da clique. Cada vértice recebe um peso que representa sua importância para a clique, sendo que os pesos são iniciados de duas formas diferentes: proporcionalmente aos graus dos vértices e com pesos iguais para todos os vértices. Na estratégia (AI), o peso inicial em cada reinicio é adaptado de acordo com os resultados das buscas anteriores, sendo que no decorrer da busca os pesos são mantidos fixos, ou seja, os pesos são adaptados somente no início de cada busca. Na estratégia (AW) os pesos são adaptados de acordo com a importância para a clique corrente, ou seja, os pesos são modificados no decorrer da busca, sendo que, os pesos no início não são adaptados com as buscas anteriores. Por fim, a estratégia (NA) não adapta os pesos durante a busca e nem no reinício de uma nova busca.

O trabalho [35] apresenta uma busca gulosa que evita máximos locais adaptando os pesos associados aos vértices. Considere $V(i)$ o conjunto dos vértices adjacentes ao vértice $i \in N$, K um subconjunto de pares de vértices adjacentes, tal que, $K \subseteq N$, e $C_P(K)$ o conjunto de vértices que não são adjacentes a p vértices de K, onde o conjunto $C_P(K)$ é fornecido por $C_P(K) = \{i \in N: |(K \setminus \{V(i)\} = p)|\}, p \in \{0{,}1\}$. [35] propõe uma melhoria em relação a regra gulosa que escolhe o vértice $i \in C_0(K)$, tal que, $\left|\left(V_i \cap C_0(K)\right)\right|$ seja máximo. Neste trabalho é escolhido $i \in C_0(K) \cup C_1(K)$, tal que, $\left|\left(V_i \cap C_0(K)\right)\right|$ seja máximo. Em seguida, um algoritmo guloso adaptativo, é repetidamente iniciado para criar cliques em torno do vértice $i \in C_0(K)$. Com esses reinícios são realizados ajustes dos pesos associados aos vértices, dependendo se $\left|\left(V_i \cap C_0(K)\right)\right|$ ou $V_i \notin C_0(K)$, em seguida é utilizado a seguinte regra gulosa. Seja $z_i = \sum w_j$, onde w_j é o peso do vértice $j \in V(i) \cap C_0(K)$, escolha do vértice $V_i \in C_0(K)$, tal que, z_i seja máximo.

A meta-heurística de colônias de formigas (ACO) para o PCM foi investigada em [25]. O algoritmo ACO gera sucessivamente cliques máximas através da adição repetida de vértices em cliques parciais, e usa o feromônio adicionado aos vértices como uma heurística gulosa para escolher, a cada passo, o próximo vértice para entrar na clique. Foram exploradas duas abordagens diferentes para as trilhas de feromônios, isto é, nas arestas ou nos vértices do grafo. Foi observado o benefício de integrar um procedimento de busca local dentro do algoritmo ACO mostrando que isso melhora o processo de solução. A cada ciclo desse algoritmo, uma formiga constrói uma clique

máxima. Primeiro, ela escolhe aleatoriamente um vértice inicial para inserir na clique e, iterativamente, adiciona vértices que estão conectados a todos os vértices da clique em construção. Encontrando uma clique maior do que as geradas até então, são adicionados feromônios nos vértices que compõem a clique e nas arestas que ligam seus vértices. Dessa forma, os vértices que constituem essas cliques terão mais probabilidades de participarem de novas cliques construídas pelo algoritmo ACO. O feromônio colocado nas arestas aumenta a probabilidades de incluir os pares de vértices conectados. O algoritmo para quando encontra a clique máxima (se conhecida) ou com um número limite de iterações.

Em [24] foi apresentada um algoritmo com múltiplos inícios que utiliza uma *lista tabu* T. O espaço de busca $S \subset N$ é explorado com a trocas de vértices entre os subconjuntos $X \subseteq S$e $Y \subseteq N \setminus \{S\}$, onde o subconjunto crítico X é o menor dos subconjuntos de S que não esteja na *lista tabu*, ou seja, $X = \{u \in S \vee u \notin T, d(u) = MinInS\}$, desde que $d(k)$ seja o grau do vértice k em relação ao subconjunto S definido como $d(k) = |(i \in S \vee (i,k) \in E)|$ e $MinInS = min\{d(u) \vee u \in S, u \notin T\}$. E o subconjunto Y é o maior dos sub conjuntos de S que não estão na *lista tabu*, $Y = \{v \in N \setminus \{S\} \vee v \notin T, d(v) = MaxOutS\}$, onde $MaxOutS = max\{d(v) \vee v \in N \setminus \{S\}, v \notin T\}$. Em outras palavras, a movimentação de troca fica limitada aos subconjuntos X e Y, tal que uma solução vizinha S' é obtida trocando um vértice $u \in X$ com um vértice $v \in Y$.

Os autores de [36] aplicam ao PCM uma meta-heurística chamada de filtragem molecular. O modelo representa soluções do PCM como moléculas de DNA codificadas em cadeias de *bits*. Caso as soluções sejam inviáveis, essas "moléculas" são filtradas (removidas) análogos aos processos bioquímicos de separação do DNA. Essa heurística depende da geração sequencial e progressiva do espaço de busca parcial, gerado com o particionamento de uma clique em duas partes. Este algoritmo possui em seu cerne duas etapas: (1) a codificação de soluções candidatas e (2) a remoção de soluções não válidas. Uma clique C é codificada por uma sequência de *n bits* atribuindo valor 1 ao *i-ésimo bit* somente quando $v_i \in C$ Cada combinação possível dessas cadeias de *bits* podem codificar ou não uma clique. Depende da existência de arestas entre os vértices com valor 1. Por exemplo, considerando um grafo de 8 vértices, uma sequência binária 10011001, é equivalente à uma clique com os vértices v_1, v_4, v_5 e v_8. A cada iteração, um par de vértices $v_i, v_j \in N$ não adjacentes com $1 \leq i < j \leq n$ é considerado e, todas as cadeias que incluem esse par de vértices são separadas em duas cliques. Cada com n combinação possível de uma clique de tamanho com k (*k-clique*) é armazenada em um vetor $B_{n,k}$. As cadeias de *bits* que representam *k-cliques* inexistentes são eliminadas, ou seja, as cadeias de *bits* que não possuem arestas entre qualquer par de *bit* com valor 1 são excluídas.

Brazilian Journal of Development

O trabalho de [16] apresenta uma meta-heurística fundamentada na neurociência para o PCM que mostrou ser eficiente para algumas famílias de grafos da base de dados do DIMACS. A meta-heurística proposta por [16] escolhe no início, aleatoriamente, um vértice com $v_i \in N$ e o insere em um subconjunto H chamado de harmonia. Então, um vértice v_j é selecionado aleatoriamente de um conjunto P que contém todos os vértices adjacentes ao vértice v_i. Em seguida, todos os vértices em que não são adjacentes a v_j são excluídos. O processo é repetido até esvaziar o conjunto P. Este algoritmo utiliza um conjunto de soluções chamado de memória harmônica, que é semelhante à escolha dos indivíduos mais aptos em algoritmos genéticos (AG). Para usar a memória de forma eficaz, normalmente é atribuído um parâmetro chamado taxa de memória, um número entre 0 e 1 que vai definir qual porcentagem da memória harmônica será utilizada. Caso muito baixa (perto de 0), apenas algumas das melhores soluções são selecionadas e, assim, a convergência do algoritmo pode ser lenta. Se a taxa é muito alta (perto de 1), serão avaliados todos os indivíduos contidos na memória harmônica, podendo ocasionar a não exploração de soluções piores que as contidas na memória harmônica. O componente taxa de ajuste de *pitch* (PAR) corresponde a avaliar uma solução ligeiramente diferente no algoritmo. Dada a *i-ésima* componente H_{old}^{i} tem-se uma nova *i-ésima* componente H_{new}^{i} retirada da memória. A nova *i-ésima* componente H_{new}^{i} é utilizada somente se a componente aleatória $r_i \in [-1,1]$ seja menor ou igual à taxa de ajuste de *pitch* (PAR), onde $1 \leq i \leq |N|$. No caso em que a avaliação de H_{old}^{i} seja melhor que a pior avaliação das soluções contidas na memória, será realizado a troca da pior solução da memória por H_{old}^{i}.

3.2 ALGORITMOS EXATOS

Em [3] os autores relatam que os algoritmos fundamentados a partir do método *Branch & Bound (B&B)* se diferem principalmente por determinar os limites inferiores e superiores, bem como a escolher uma estratégia adequada para ramificação e corte do espaço de busca. Em [37] os autores relatam que existe um consenso, de que os melhores algoritmos exatos para o PCM são derivadas do método *B&B*.

O algoritmo CP apresentado em [31], é uma das principais referências para os algoritmos exatos aplicados ao PCM e, serve de base para muitos outros. O algoritmo gera uma árvore de enumeração parcial. Cada nó da árvore de busca corresponde a um conjunto de vértices que representa uma clique. Considere η e η' dois vértices da árvore de busca tais que η' é filho de η. Então, a clique representada em η' contém o conjunto de vértices representado em η mais um vértice adjacente a todos os vértices de η. Inicialmente, os vértices do grafo são ordenados por seu grau tal que, v_1 é o vértice de menor grau em G, v_2 é o vértice de menor grau no grafo $G \setminus \{v_1\}$, e

Brazilian Journal of Development

generalizando, v_k é o vértice de menor grau no grafo $G \setminus \{v_1, \ldots, v_{k-1}\}$. O algoritmo gera n cliques, $C_1, \ldots, C_n$.C_1 é a maior clique que contém o vértice v_1, C_2 é a maior clique que contém o vértice v_2 no grafo $G \setminus \{v_1\}$, e daí por diante. Considere v_{di} o vértice relacionado ao estado da árvore de busca no nível d no passo i, ou seja, no nível d temos os vértices $v_{d1}, \ldots, v_{di}, \ldots, v_{dm}$ adicionados à clique correspondente ao nó pai destes vértices. O trabalho mostra que o nó v_{di} não precisa ser expandido se$d + (m - i) \leq num$, onde num é o tamanho da maior clique gerada pelo algoritmo. Neste caso, o número de vértices na clique de maior tamanho gerada a partir do vértice v_{di} seria menor que num.

Os autores do trabalho apresentado em [20] utilizam o método *B&B* e para calcular o limite superior recorrem ao algoritmo heurístico DSATUR, [38], para coloração de vértices. A coloração de um grafo consiste em atribuir a menor quantidade de cores para cada vértice, tal que, nenhum vértice adjacente tenha a mesma cor. O número cromático $\chi(G)$ é definido como a menor quantidade de cores necessárias para colorir um grafo G. Considerando que a partir do teorema de *Turan* podemos deduzir que $\omega(G) \leq \chi(G)$, o número cromático pode ser utilizado como limite superior da clique máxima, para mais detalhes ver [11]. O algoritmo proposto em [20] considera os vértices $v_1, v_2, \ldots, v_n$ de um grafo G em qualquer ordem. Em seguida, inicia a construção de uma clique C e a coloração do grafo G, onde se define $\omega' = |C|$ como limite inferir para $\omega(G)$, tal que, se $\chi(G) = \omega'$ a clique máxima foi encontrada e $\omega(G) = \omega'$. Caso contrário, a técnica *B&B* computa os n subgrafos $H_i, i = 1,2, \ldots, n.$ induzido pelo conjunto de vértices $N_i \cup \{v_i\}, N_i := V(v_i) \cap \{v_1, v_2, \ldots, v_{i-1}\}$, onde $V(v_i)$ é conjunto de vértices adjacentes ao vértice v_i, ou seja, a vizinhança de v_i. Pode-se reduzir a árvore de busca considerando que $\chi(V(v_i))$ é o número cromático dos vértices vizinhos ao vértice v_i, tal que, se $\chi(V(v_i)) < \omega'$ não há clique maior que ω'que contenha v_i, então a busca pode deletar o vértice v_i e continuar a explorar $G \setminus \{v_i\}$.

Em [39] foi utilizada uma técnica semelhante à Programação Dinâmica armazenando os subgrafos produzidos no decorrer da busca e melhorando o limite superior do algoritmo CP. Seja o conjunto de vértices $S_i = \{v_i, v_{i+1}, \ldots, v_n\}$, o algoritmo CP considera primeiro a clique formada por S_1 seguida por S_2 e, assim por diante, como descrito no início desta sessão. O algoritmo apresentado em [39] realiza este procedimento em ordem inversa, construindo inicialmente uma clique em S_n contendo o vértice v_n , seguido de S_{n-1} contendo v_{n-1} e assim por diante. Seja uma função $c(i)$que avalia a maior clique em S_i, desde que$1 \leq i \leq n - 1$ temos $c(i) = c(i + 1)$ ou $c(i) = c(i + 1) + 1$. Se houver uma clique em S_i de tamanho $c(i + 1) + 1$ que contem o vértice v_i , então procura-se por novas cliques a partir de $c(n) = 1$. Se a clique é encontrada temos $c(i) =$

Brazilian Journal of Development

$c(i+1)+1$, caso contrário $c(i)=c(i+1)$. Por fim, o tamanho da clique máxima será dado por $c(1)$. Seja s uma quantidade de cores que um grafo pode ser colorido sem que vértices adjacentes tenham a mesma cor. Valores anteriores de $c(i)$ permitem a poda da árvore da seguinte forma. Se for procurar uma clique com tamanho maior que s, pode-se descartar essa busca se considerar que v_i será o *(j+1)-ésimo* vértice e$j+c(i)\leq s$. O limite inferior é obtido calculado o valor de $d(i)$, que corresponde ao maior tamanho de uma clique contendo os vértices $v_1, v_2, \ldots, v_i$, tal que, se $i=n$ o problema foi resolvido. Os valores de $d(i)$ são calculados para $1\leq i\leq n/2$. Em seguida é verificado o valor de $c(i+1)+d(i)$ para $i\leq n/2$, e pode-se cancelar a busca se para algum $i, c(i+1)+d(i)<s$ i, pois não existe clique de tamanho s.

Os autores do trabalho [21] propuseram três algoritmos que realizam melhorias do algoritmo CP. Considerando P o conjunto de vértices$N$ordenados conforme a explicação para o algoritmo CP no início desta seção, C_{max} a maior clique obtida pelo algoritmo com a união de um vértice p adjacente a todos vértices de uma clique corrente C'. É determinado o conjunto $P'=P\cap V(v)$, onde $V(v)=\{u\in N\vee\{u,v\}\in E\}$ é a vizinhança do nó $v\in N$, e o grau do vértice $v\in N$ é definido por $\delta(v)=\left|\left(V(v)\right)\right|$, em seguida, é copiado para o conjunto $P'\forall p\in P$, desde que exista $\{p,v\}\in E$. A próxima etapa é responsável por retirar os vértices $v\in P'$, tal que, $\delta_{P'}(V)+|(C')|<|(C_{max})|$, seguido da adição $C'\cup\{v\}$ e subtração $P'\setminus\{v\}$ para $\forall v\in P'$, tal que, $\delta_{P'}=|(P')|-1$. O segundo algoritmo utiliza heurísticas para colorir o subgrafo $G'=G\setminus\{v\in P'\}$ formado pelos vértices do conjunto P', em seguida utiliza o número cromático $\chi(G')$ para remover os vértices do conjunto P' que não podem aumentar a clique C'. O terceiro algoritmo combina as estratégias de remoção de vértices utilizado pelo primeiro e segundo algoritmo. Se $\chi(G')+|(C')|<|C_{max}|$então $v\in P'$ poderá ser removido com segurança.

No ano seguinte ao trabalho [21] os autores de [2] apresentam melhorias do método de coloração de vértices visto com uma extensão da coloração de subgrafos. O algoritmo insere cada vértice p do grafo na respectiva classe de sua cor C_k, tal que, p não seja adjacentes aos vértices desta classe. Se o vértice atual p tiver pelo menos um vértice adjacente em cada classe $C_1, C_2, \ldots, C_k$, então o vértice p será inserido em uma nova classe C_{k+1}. Depois de atribuir a classe de cor para todos os vértices de P, esses vértices são reordenados em P conforme aparecem em cada classe C_k e na ordem crescente em relação ao índice k. O limite superior do tamanho da clique máxima é o número classes, esse limite depende muito de como os vértices são apresentados para esse algoritmo. O limite superior é mais justo quando os vértices em P são considerados em uma ordem não crescente em relação ao seu grau. O procedimento considera o conjunto C' para a clique corrente e C_{max} para a maior clique encontrada durante a busca. E incia-se com a operação $p\cup C'$, onde $p\in P$, em

seguida é computado $P' = P \cap V(p)$, onde $V(p)$ é o conjunto de todos vértices adjacentes ao vértice p. Da mesma forma que algoritmo visto em [31], é realizado sucessivas chamadas recursivas até que $P' = \emptyset$. / E se C' é maximal e $|(C')| > |(C_{max})|$, então C_{max} é substituído por C'e remove-se p de C'e P. Para descartar vértices, é realizado um procedimento de numeração dos vértices do conjunto P' em ordem crescente em relação aos números de k cores em cada chamada recursiva do algoritmo. Para cada $p \in P'$ é atribuído um inteiro positivo $I[p]$ utilizando um algoritmo de coloração guloso. Se $(p,r) \in E$ então $I[p] \neq I[r]$, se $I[p] = 1$ ou $I[p] = k > 1$ então existem vértices $p_1 \in V(p), p_2 \in V(p), \ldots, p_{k-1} \in V(p)$ no conjunto P' com $I[p_1] = 1, I[p_2] = 2, \ldots$, $I[p_{k-1}] = k - 1$. Consequentemente, e desde que $C' + Max\{I[p], p \in P'\} \leq C_{max}$, pode-se desconsiderar P'.

O procedimento de numeração e classificação apresentado em [2] é melhorado no algoritmo introduzido pelos autores de [40], no qual o limite superior é alcançado quando os vértices em P são apresentados ao procedimento de coloração guloso em uma ordem crescente de seus graus. Sendo necessário reordenar (decrescente em relação aos números de cores) somente os vértices no conjunto P com números de cores grandes o suficiente para serem adicionados à clique atual, C. Qualquer vértice $v \in P$ com um número de cores abaixo do limiar $K_{min} < |C_{max}| - |C| + 1$ nunca será adicionado à clique atual e é mantido na mesma ordem em que foi apresentado ao algoritmo de coloração. O procedimento ainda é melhorado por uma coloração gulosa em ordem decrescente do grau de seus vértices e por proposições associadas ao problema da satisfatibilidade. O algoritmo apresentado em [41] substitui a ordenação inicial dos vértices que possuem os mesmos graus por uma coloração heurística igual ao algoritmo proposto em [2]. Em seguida, foi realizada uma nova modificação na ordenação inicial dos vértices fazendo que os vértices do conjunto P sejam coloridos sempre na mesma ordem. Em [42], foi apresentado outro algoritmo que possui uma melhora no desempenho da memória cache com a inclusão de uma matriz de adjacência e permutações dos vértices no início da execução do algoritmo.

4 ALGORITMOS

Devido a proposta apresentada neste artigo recorrer ao algoritmo visto em [31] com uma pequena adaptação, inicialmente será descrito o algoritmo 1 seguido pelo algoritmo 2, e por fim, todas sub-rotinas utilizadas pelo algoritmo 2.

Brazilian Journal of Development

4.1 ALGORITMO CP ADAPTADO

Em sua forma original, o algoritmo 1 considera inicialmente como entrada uma clique C vazia, e um conjunto P com todos os vértices do grafo G ordenados em forma crescente de seus respectivos graus. A variável global C_{max} na linha 3 é utilizada para representar a maior clique obtida após cada chamadas recursivas da linha 14. A componente *bounding* é realizada na linha 9 e a componente *branching* na linha 10.

Algorithm 1: ALGORITMO CP

```
1  Entrada C,P
2  Saída C_max
3  C_max ← C
4  CP(C,P)
5  Função CP (C, P)
6  if |C| > |C_max| then
7  |  C_max ← C
8  end
9  if |C| + |P| > |C_max| then
10 |  for todo p ∈ P do
11 |  |  P ← P \ {p}
12 |  |  C_a ← C ∪ {p}
13 |  |  P_a ← P ∩ Γ(p)
14 |  |  CP(C_a, P_a)
15 |  end
16 end
17 Fim da Função CP
```

O limite inferior é representado pela condição da linha 6 e o limite superior é calculado na linha 9. A árvore de busca é construída entre as linhas 9-15. Ou seja, caso o limite $|C| + |P|$ permita encontrar uma clique maior que C_{max}, a subárvore com raiz na clique C vai ser mais explorada. A linha 12 é responsável por construir C a com a união de um vértice p com a clique C. A função $\Gamma(p)$ seleciona todos vértices adjacentes ao vértice adicionado a clique C na linha 12, e a operação da linha 13 encontra todos vértices que podem aumentar a clique C_a. O algoritmo 1 adaptado não inicia a clique C vazia, é utilizado o algoritmo heurístico 5 para construir a clique C, os vértices do conjunto P são selecionados recorrendo ao algoritmo 4.

4.2 ALGORITMO CPH

Esta seção apresenta um algoritmo populacional, denominado CPH, que hibridiza a meta-heurística *Simulated Annealing* e uma *lista tabu LT* com limite de TLT vértices para cada indivíduo.

Brazilian Journal of Development

A primeira linha do algoritmo 2 recebe como entrada um grafo G, o tamanho da população TP, a temperatura inicial $T0$, a quantidade de iterações QI, o parâmetro de incremento da iteração pI, o tamanho da lista tabu TLT e o grau $d(v_n)$ de cada vértice do grafo G. O algoritmo 2 retorna a maior clique obtida (linha 2). Cada indivíduo inicial da população é gerado utilizando um algoritmo 5, onde o vértice com maior grau tem mais chance de ser escolhido primeiro.

Os indivíduos evoluem mediante operações de troca, retirada e inserção de vértices. Para cada temperatura é reiniciada a população, contudo o melhor indivíduo é mantido na população. Os vértices contidos em LT, ficam proibidos serem inseridos em seus respectivos indivíduos. A lista tabu LT é iniciada na linha 3.

Algorithm 2: ALGORITMO CPH

```
1  Entrada: G, TP, T0, QI, pI, TLT, d(v_n)
2  Saída: Clique
3  LT ← ∅
4  POP ← Gera População(G, TP, d(v_n))
5  Temperatura ← T0
6  while Temperatura > 0 do
7      for i = 1 até QI do
8          for C_i ∈ POP do
9              C' ← Remoção de Vértices(C_i, d(v_n))
10             Γ(C') ← Gera Vizinhança(C', G, LT)
11             C ← CP(C', Γ(C'))
12             if |C| > |C_i| + 1 then
13                 C_i ← C
14             else
15                 LT ← Lista Tabu(C_i, C, LT, TLT)
16                 R ← rand(0, 1)
17                 if R < Temperatura/T0 then
18                     C' ← Troca de
                         Vértices(G, C_i, d(v_n))
19                     C'' ← Remoção de
                         Vértices(C', d(v_n))
20                     C_i ← Busca
                         Heurística(C'', G, LT, d(v_n))
21                 else
22                     C' ← Troca de
                         Vértices(G, C_i, d(v_n))
23                     C'' ← Remoção de
                         Vértices(C', d(v_n))
24                     Γ(C'') ← Gera
                         Vizinhança(C'', G, LT)
25                     C_i ← CP(C'', Γ(C''))
26                 end
27             end
28             i=i+1
29         end
30         Clique ← Melhor Indivíduo(POP)
31     end
32     POP ← ∅
33     POP ← Gera População(G, TP − 1, d(v_n))
34     POP ∪ Clique
35     QI ← QI + pI
36     Temperatura ← Temperatura − 1
37 end
```

Brazilian Journal of Development

O procedimento da linha 4 gera a população inicial com TP indivíduos. A temperatura corrente é iniciada com $T0$ na linha 5. O laço entre as linhas 6 e 37 itera enquanto a temperatura for maior que zero. O laço entre as linhas 7 e 31 realiza tentativas em melhorar os indivíduos para uma determinada Temperatura, tal que, seguindo a metáfora do *Simulated Anneling* para Temperatura mais baixa, a quantidade de tentativas QI será aumentada pelo fator pI na linha 35. O laço entre as linhas 8 e 29 é responsável por selecionar cada indivíduo C_i presente na população POP. O procedimento Remoção de Vértices (linha 9,19,23), remove um vértice do indivíduo C_i utilizando uma roleta em função do grau de cada vértice. O procedimento Gera Vizinhança (linhas 10 e 24) é responsável por gerar o conjunto de vértice que são adjacentes a todos vértices da clique inserida na entrada do procedimento, contudo não gera vértices que estejam na *lista tabu LT*. O procedimento da linha 11 procurar aumentar a clique com o algoritmo 1, sendo que, após retirar um vértice da clique seguido da aplicação de um algoritmo exato, o aumento de uma unidade não representa melhora do indivíduo. Dessa forma, a melhoria é verificada se a condição da linha 12 for satisfeita, ou seja, espera-se que a clique aumente em mais de uma unidade. Dessa forma, seguindo a metáfora do *Simulated Anneling*, a condição da linha 12 garante que sempre será aceito uma melhora do indivíduo, tal que, o indivíduo da população é atualizado na linha 13, caso contrário, o vértice adicionado no procedimento da linha 11 é inserido na lista tabu LT pelo procedimento da linha 15. Em seguida, será sorteado um número entre 0 e 1 na linha 16 e a condição da linha 17 determina se a solução corrente será diversificada pelo algoritmo 5 (linha 20) ou intensificada pelo algoritmo 1 (linha 25). O procedimento das linhas 18 e 22 realizam a troca de um vértice do indivíduo C_i por um vértice contido no grafo G que seja adjacente a $|C| - 1$ vértices de C_i. Caso exista um vértice para ser trocado, ele é retirado de C_i, caso contrário não é realizado a troca. O procedimento da linha 30 verifica se o maior indivíduo da população na iteração corrente é maior que o maior indivíduo encontrado no decorrer da busca, se verdade, é atualizado a solução Clique. O procedimento das linhas 32 e 33 reinicia a população para cada temperatura decrementada na linha 36. O algoritmo 3 recebe como entrada o indivíduo C_i da população POP, a clique C resultante do procedimento da linha 11 pelo algoritmo 2, a *lista tabu LT* e o tamanho da lista tabu TLT.

Brazilian Journal of Development

Algorithm 3: LISTA TABU

```
1  Entrada: C_i, C, LT, TLT
2  Saída: LT
3  for v_i ∈ C do
4      if ((v_i ∉ C_i) then
5          if |LT| < TLT then
6              LT ∪ v_i
7          else
8              LT \ v_1
9              LT ∪ v_i
10         end
11     end
12 end
```

O laço entre a linha 3 e 12 percorre todos vértices v_i da clique C. A condição da linha 4 verifica se v_i não esta contido no indivíduo C_i, caso seja satisfeita, o vértice v_i é inserido na lista tabu LT pelo procedimento da linha 6 ou 9. A condição da linha 5 verifica o limite da *lista tabu*, caso esteja cheia, o procedimento da linha 8 remove o primeiro vértice v_1 da *lista tabu* LT.

O algoritmo 4 é responsável por encontrar todos vértices do grafo G que podem aumentar a clique C e não pertencem a *lista tabu* LT. Recebe como entrada uma clique C, o grafo G e a *lista tabu* LT. A saída é o conjunto de vértices $\Gamma(C)$ que são adjacentes a todos vértices da clique C.

Algorithm 4: GERA VIZINHANÇA

```
1  Entrada: C, G, LT
2  Saída: Γ(C)
3  for v_i ∈ G do
4      cot ← 0
5      for v_j ∈ C do
6          if ∃(v_i, v_j) ∈ G then
7              cot ← cot + 1
8          end
9      end
10     if cot = |C| then
11         sinal ← F
12         for v_j ∈ LT do
13             if v_i = v_j then
14                 sinal ← V
15             end
16         end
17         if sinal = F then
18             Γ(C) ← v_i
19         end
20     end
21 end
```

Brazilian Journal of Development

O laço entre a linha 3 e 21 percorre todos vértices do grafo G, o laço entre a linha 5 e 9 percorre os vértices da clique C, a condição da linha 6 verifica se existe uma aresta entre os vértices v_i e v_j, caso a condição da linha 6 seja satisfeita, é acrescentado na linha 7 uma unidade a variável cot que foi inicializada na linha 4. A condição da linha 10 verifica se o vértice v_i é adjacente a todos vértices de C. Caso seja satisfeita a condição da linha 10, incia-se o laço da linha entre a 12 e 16 que percorre os vértices da *lista tabu LT*, caso a condição da linha 13 seja satisfeita, o sinal iniciado na linha 11 como falso e atualizado para verdadeiro na linha 14. Por fim, a condição da linha 17, verifica se o vértice v_i não pertence a *lista tabu LT*, caso a condição seja satisfeita, é adicionado o vértice v_i ao conjunto $\Gamma(C)$ na linha 18.

O algoritmo 5 gera uma clique composta pelos vértices do conjunto C. Ele recebe como entrada uma clique C, um grafo G, a *lista tabu LT* e o grau $d(v_n)$ de cada vértice do grafo G.

Algorithm 5: BUSCA HEURÍSTICA

```
1  Entrada: C, G, LT, d(v_n)
2  Saída: C
3  Γ(C) ← Gera Vizinhança(C, G, LT)
4  while Γ(C) ≠ ∅ do
5  |   V ← ∅
6  |   for i ∈ Γ(C) do
7  |   |   V ← V ∪ {d(Γ(C))}
8  |   end
9  |   C ← C ∪ {rand(V)}
10 |   Γ(C) ← Gera Vizinhança(C, G, LT)
11 end
```

Na linha 3 recorre ao algoritmo 4 para obter o conjunto $\Gamma(C)$ de vértices adjacentes a clique C. Enquanto o conjunto $\Gamma(C)$ não for vazio, o laço entre as linhas 4 e 11 itera. Será realizado a inserção de vértices na clique C enquanto a condição do laço na linha 4 for satisfeita. A linha 5 inicializa a lista V que vai armazenar o grau $d(v_n)$ de cada vértice vizinho a clique C. O laço entre as linhas 6 e 8 é responsável por criar a lista V com os graus dos vértices de $\Gamma(C)$. O procedimento *rand()* recebe a lista V, em seguida, sorteia utilizando uma roleta proporcional ao grau de cada vértice $v_n \in V$. O vértice com maior grau tem maior probabilidade de ser escolhido. O vértice escolhido é adicionado ao conjunto C na linha 9. No final de sua da execução, retorna o conjunto C que representa uma clique.

O algoritmo 6 constrói uma população onde cada indivíduo é uma clique. Os dados de entrada são o grafo G, tamanho da população TP e o grau$d$ (v_n) de cada vértice do grafo G.

Brazilian Journal of Development

Algorithm 6: Gera População

```
1 Entrada: G, TP, d(v_n)
2 Saída: POP
3 POP ← ∅
4 while |POP| ≤ TP do
5 |   C ← ∅
6 |   C ← Busca Heurística (C, G, d(v_n))
7 |   POP ← POP∪C
8 end
```

A saída é composta por todos os indivíduos gerados enquanto o laço 4-8 itera, no qual é realizada a chamada do algoritmo 5 na linha 6. A linha 7 é responsável por adicionar a clique C na população *POP*.

O algoritmo 7 é responsável por realizar a remoção de um vértice, recebe como entrada uma clique contida no conjunto C, o número de vértices $|N|$ e o grau $d(v_n)$ de cada vértice do grafo G. A saída é a clique C de entrada sem o vértice v_i. A linha 3 inicializa a lista T que vai armazenar a diferença entre a cardinalidade do grafo G e o grau $d(v_i)$ no grafo de cada vértice v_i da clique, em seguida, o laço da linha 4 cria a lista T. O sorteio realizado na linha 7 oferece mais chances em remover o vértice de menor grau da clique C.

Algorithm 7: REMOÇÃO DE VÉRTICES

```
1 Entrada: C, |N|, d(v_n)
2 Saída: C \ {v_i}
3 T ← ∅
4 for v_i ∈ C do
5 |   T ← T ∪ {|N| − d(v_i)}
6 end
7 v_i ← rand(T)
8 C_max ← C \ {v_i}
```

O algoritmo 8 é responsável pela troca de vértices, recebe como entrada um grafo G, uma clique C e o grau $d(v_n)$ de cada vértice do grafo G. A saída é uma nova clique C sem o vértice v_i removido na linha 9 e com o vértice v_j adicionado na linha 8.

Brazilian Journal of Development

Algorithm 8: TROCA DE VÉRTICES

```
1 Entrada: G, C, LT
2 Saída: C
3 V ← ∅
4 for j ∈ Γ(J) do
5 |   V ← V ∪ {d(Γ(J))}
6 end
7 v_j ← Γ(rand(V))
8 C ← C ∪ {v_j}
9 C ← C \ {v_i}
```

A linha 3 inicializa a lista V que vai armazenar o grau $d(v_j)$ no grafo de cada vértice de J, onde J é o conjunto com os vértices do grafo G que são adjacentes a $n-1$ vértices da clique C. O procedimento da linha 7 recebe a lista V e sorteia, utilizando uma roleta proporcional ao grau de cada vértice, um vértice de G. O vértice v_i removido na linha 9 é o único vértice não adjacente ao vértice v_j adicionado na linha 8.

5 METODOLOGIA DE TESTES

O algoritmo proposto foi comparado ao resultado publicado recentemente no trabalho de [16]. Foram utilizadas as 80 instâncias do banco DIMACS (http://iridia.ulb.ac.be/fmascia/maximumclique/DIMACS-benchmark) as quais possuem resultados ótimos conhecidos. Os experimentos foram executados em um computador DELL Inspiron 5566, com processador Intel Core i5-7200U de 3.1 GHz e 4Gb de RAM, sistema operacional Ubuntu 16.04 xenial. O algoritmo proposto foi implementado na linguagem C++ e executado 50 vezes para cada instância, sendo calculada a média das soluções, desvios padrões e média do tempo requerido para cada instância. Os resultados relatados para o algoritmo de [16] foram retirados do trabalho original.

É analisada a quantidade de soluções ótimas e o tempo necessário para resolver as instâncias com o algoritmo 2. Foram retiradas as soluções apresentadas em [16] e realizado a comparação com soluções obtidas pelo algoritmo 2, bem como, destacado quais foram as famílias das instâncias que o algoritmo 2 obteve melhores soluções que os resultados retirados do trabalho [16]. A escolha dos resultados do trabalho de [16] decorre de ser uma publicação recente.

Para reportar o desempenho real da máquina utilizada nos experimentos, realizou-se um procedimento padrão adotado pelas pesquisas que envolvem o PCM. O site do DIMACS disponibiliza o algoritmo dfmax [31] e um conjunto de instâncias para serem resolvidas, em seguida,

o pesquisador reporta o tempo obtido para solucionar o PCM para essas instâncias. O tempo obtido com a resolução destas instâncias, serve para que outros pesquisadores, possam comparar o tempo de execução de algoritmos em suas máquinas, com o tempo obtido pelo algoritmo executado na máquina em que foi implementado o algoritmo aqui apresentado. O tempo (em segundos) de execução obtido pelo algoritmo dfmax para as instâncias r.100.b, r.200.b, r.300.b, r.400.b e r.500.b foram respectivamente 0,00, 0,06, 0,35, 1,94 e 7,29.

O software IRACE [43] foi utilizado para definir os seguintes parâmetros: tamanho da população, TP, temperatura inicial, $T0$, quantidade de iterações, QI, o incremento da quantidade de iteração para cada temperatura, pI, tamanho da lista tabu TLP. As instâncias que serviram de entrada para o IRACE foram escolhidas de forma aleatória, sendo elas: brock800.4, c250.9, c f at500.10, gen200.p0.9.44, hamming8.4, mann.a81, johnson8.24, phat300.3 e san400.0.7.2, em seguida, atribuídas ao software IRACE para realizar seu processamento estatístico. Estas instâncias não foram utilizadas no experimento com o algoritmo CPH. O IRACE recebe como entrada um conjunto de valores possíveis para os parâmetros livres, dessa forma, foi atribuído para TP e $T0$ os seguintes valores 5, 10, 15, 20, 25 e 30, para QI os valores de 10, 20, 30, 40, 50, 60, 70, 80, 90 e 100, e para pI e TLP os parâmetros de 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10. O software retornou os seguintes parâmetros: TP = 25, $T0$ = 5, QI= 10, pI = 6 e TLP =4.

6 RESULTADOS

A tabela 1 em anexo apresenta os resultados 71 instâncias do banco de dados DIMACS. As colunas identificam o nome da instância, a solução ótima retirado do site do DIMACS, a média das soluções obtidas pelo algoritmo 2, o desvio padrão do algoritmo 2, a melhor solução obtida pelo algoritmo 2, a média das soluções obtidas pelo algoritmo HS, para mais detalhes dos resultados do algoritmo HS vale apena ver [17].

O algoritmo CPH somente não processa 1 instância em menos de 2 minutos, 3 instâncias em menos de 1 minuto e 12 instâncias (16,9%) em menos de 10 segundos. Consegue processar um total de 59 instâncias (83%) em menos de 115 segundos, um total de 55 instâncias (77%) em menos 85 segundos e um total de 43 instâncias (60%) em menos 45 segundos. Obteve o valor ótimo para 53 instâncias (74,6%).

Processa cada instância da família Brock com 800 vértices em menos de 3 segundos. Processa a instância c2000.5 com 2000 vértices em 8,09 segundos. Soluciona todas instâncias da família cfat com até 500 vértices em 6,2 segundos. Para família Phat, processa todas as 3 instâncias com 1000 vértices em 28,5 segundos e todas instâncias com menos de 1000 vértices, totalizando

4200 vértices, em 17 segundos. Foi capaz de processar a instância c4000.5 de 4000 vértices em 20,25 segundos e obteve a média distante em 12% da solução ótima. Dentre as instâncias da família san, somente a instância san_1000 não foi solucionada em menos de 11 segundos, e somente san400_0.5_1 não foi solucionada em menos de 5 segundos, e para todas instâncias da família san foi encontrada a solução ótima.

O desvio padrão foi menor que 0,5 para 49% das instâncias, representando que as soluções de cada execução destas instâncias ficaram próximas à solução média. Em comparação com o resultado recente encontrado na literatura [16], a média da solução do algoritmo apresentado neste artigo foi estritamente melhor em 37 instâncias (52,11%), 15 instâncias (21,1%) obtiveram o mesmo valor e somente perdeu em 19 instâncias (26,7%).

Pode-se observar na tabela 1 que, em relação a qualidade da solução para a família phat, somente para 3 das 14 instâncias o algoritmo CPH não é melhor que os resultados do algoritmo HS. Pode-se notar também, uma melhora significativa nas soluções das instâncias 500_2, 1000_2 e 1000_3. E para as 10 instâncias da família san, o algoritmo CPH só não foi melhor que o HS instância 200_07_2. Só não foi melhor em 4 instâncias para família Brock, sendo melhor para maioria dos grafos Brock com a 800 vértices. Foi melhor em todas instâncias da família gen. Para instâncias da sanr, só não foi melhor que a 200_0.9.

7 CONCLUSÕES

O algoritmo apresentado neste artigo conseguiu aproveitar a qualidade da meta-heurística Simulated Annealing em alternar a intensificação e diversificação da busca realizada por uma clique máxima. A lista tabu utilizada evita a inserção de vértices que não obtiveram êxitos em melhorar a clique. Dessa forma, quando a metáfora do Simulated Anneling escolhe entre diversificar ou intensificar a solução, não existe chance desse vértice entrar na clique. O Reinicio aleatório da população para cada temperatura preserva a melhor indivíduo, dessa forma não perde uma solução promissora. O algoritmo foi capaz de processar todas 71 instâncias do experimento em menos de 14 minutos. Teve uma execução rápida em relação ao resultado recente publicado por [16] obteve um melhor desempenho em qualidade de solução. Pretende-se como trabalhos futuros utilizar informações obtidas com a coloração de vértices, realizar cruzamento entre os melhores indivíduos da população e explorar mais critérios para lista tabu, como evitar inserir vértices que foram removidos ou trocados sem sucessos.

AGRADECIMENTOS

"O presente trabalho foi realizado com apoio da Fundação Universidade Federal de Mato Grosso do Sul – UFMS/MEC – Brasil";

"O presente trabalho foi realizado com apoio da Universidade Federal de Rio Grande do Norte – DIMAP- Brasil";

REFERÊNCIAS

[1] KARP, R. M. Reducibility among combinatorial problems. In:. Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department. Boston, MA: Springer US, 1972.

p. 85–103. ISBN 978-1-4684-2001-2. doi.org/10.1007/978-1-4684-2001-2 9i.

[2] TOMITA, E.; SEKI, T. An efficient branch-and-bound algorithm for finding a maximum clique. In: CALUDE, C. S.; DINNEEN, M. J.; VAJNOVSZKI, V. (Ed.). Discrete Mathematics and Theoretical Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 278–289. ISBN 978-3-540-45066-5.

[3] WU, Q.; HAO, J.-K. A review on algorithms for maximum clique problems. European Journal of Operational Research, v. 242, n. 3, p. 693 – 709, 2015. ISSN 0377-2217. Disponível em: hhttp://www.sciencedirect.com/science/ article/pii/S0377221714008030i.

[4] Li, X. et al. A distributed transmission scheduling algorithm for wireless networks based on the ising model. In: 2018 IEEE Statistical Signal Processing Workshop (SSP). [S.l.: s.n.], 2018. p. 6–10. ISSN null.

[5] DOUIK, A. et al. A Tutorial on Clique Problems in Communications and Signal Processing. 2018.

[6] DOUIK, A. et al. Delay reduction in multi-hop device-to-device communication using network coding. IEEE Transactions on Wireless Communications, PP, 09 2014

[7] CHO, S.; BYUN, H. A space-time graph optimization approach based on maximum cliques for action detection. IEEE Transactions on Circuits and Systems for Video Technology, v. 26, n. 4, p. 661–672, April 2016. ISSN 1558-2205.

[8] WONG, S. W. et al. Modeling tumor progression via the comparison of stage-specific graphs. Methods, v. 132, p. 34 –41, 2018. ISSN 1046-2023. Comparison and Visualization Methods for High-Dimensional Biological Data.

[9] CHAPUIS, G. et al. Finding maximum cliques on a quantum annealer. In: Proceedings of the Computing Frontiers Conference. New York, NY, USA: Association for Computing Machinery, 2017. (CF'17), p. 63–70. ISBN 9781450344876.doi.org/10.1145/3075564.3075575i.

[10] PELOFSKE, E.; HAHN, G.; DJIDJEV, H. Solving large maximum clique problems on a quantum annealer. In: FELD, S.; LINNHOFF-POPIEN, C. (Ed.). Quantum Technology and Optimization Problems. Cham: Springer International Publishing, 2019. p. 123–135. ISBN 978-3-030-14082-3.

[11] CAVIQUE L.; REGO, C. T. Estruturas de vizinhança e procura local no problema da clique máxima. Investigação Operacional, v. 22, p. 1–18, 2002.

[12] JOHNSON, D. J.; TRICK, M. A. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993. USA: American Mathematical Society, 1996. ISBN 0821866095.

[13] BATTITI, R.; PROTASI, M. Reactive local search for the maximum clique problem 1. Algorithmica, Springer, v. 29, n. 4, p. 610–637, 2001.

[14] KATAYAMA, K.; HAMAMOTO, A.; NARIHISA, H. An effective local search for the maximum clique problem. Information Processing Letters, v. 95, n. 5, p. 503 – 511, 2005. ISSN 0020-0190.

[15] PULLAN, W.; MASCIA, F.; BRUNATO, M. Cooperating local search for the maximum clique problem. Journal of Heuristics, Springer, v. 17, n. 2, p. 181–199, 2011.

[16] ASSAD, A.; DEEP, K. A heuristic based harmony search algorithm for maximum clique problem. OPSEARCH, v. 55, n. 2, p. 411–433, Jun 2018. ISSN 0975-0320.

[17] PARDALOS, P.; XUE, J. The maximum clique problem. Journal of Global Optimization, v. 4, p. 301–328, 04 1994.

[18] BOMZE, I. M. et al. The maximum clique problem. In: . Handbook of Combinatorial Optimization: Supplement Volume A. Boston, MA: Springer US, 1999. p. 1–74. ISBN 978-1-4757-3023-4.

[19] BUTENKO, S.; PARDALOS, P. M. Maximum Independent Set and Related Problems, with Applications. Tese (Doutorado)

— University of Florida, USA, 2003. AAI3120100.

[20] BABEL, L.; TINHOFER, G. A branch and

bound algorithm for the maximum clique problem. Zeitschrift für Operations Research, v. 34, n. 3, p. 207–217, May 1990. ISSN 1432-5217.

[21] FAHLE, T. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: MÖHRING, R.; RAMAN, R. (Ed.). Algorithms — ESA 2002. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 485–498. ISBN 978-3-540-45749-7.

[22] SORIANO, P.; GENDREAU, M. Tabu search algorithms for the maximum clique problem. v. 26, 01 1994.

[23] GENDREAU, M.; SORIANO, P.; SALVAIL, L. Solving the maximum clique problem using a tabu search approach. Annals of Operations Research, v. 41, n. 4, p. 385–403, Dec 1993. ISSN 1572-9338.

[24] WU, Q.; HAO, J.-K. An adaptive multistart tabu search approach to solve the maximum clique problem. Journal of Combinatorial Optimization, v. 26, 07 2013.

[25] SOLNON, C.; FENET, S. A study of aco capabilities for solving the maximum clique problem. Journal of Heuristics, v. 12, n. 3, p. 155–180, May 2006. ISSN 1572-9397.

[26] HOMER, S.; PEINADO, M. Experiments with polynomial-time clique approximation algorithms on very large graphs. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, v. 26, p. 147–168, 1996.

[27] DUAN, H. et al. Three-dimension path planning for ucav using hybrid meta-heuristic aco-de algorithm. Simulation Modelling Practice and Theory, v. 18, n. 8, p. 1104 – 1115,

2010. ISSN 1569-190X. Modeling and Simulation for Complex System Development.

[28] HOMBERGER, J.; GEHRING, H. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. European Journal of Operational Research, Elsevier, v. 162, n. 1, p. 220–238, 2005.

[29] RIOS , B. H. O.; GOLDDARG, E. F. G.; Quesquén, G. Y. O. A hybrid metaheuristic using a corrected formulation for the traveling car renter salesman problem. In: 2017 IEEE Congress on Evolutionary Computation (CEC). [S.l.: s.n.], 2017. p. 2308–2314.

[30] FERNANDES, I. F. d. C. Meta-heurísticas híbridas aplicadas ao problema da árvore geradora multiobjetivo. Dissertação (Mestrado) , Brasil, 2018.

[31] CARRAGHAN, R.; PARDALOS, P. M. An exact algorithm for the maximum clique problem. Operations Research Letters, v. 9, n. 6, p. 375 – 382, 1990. ISSN 0167-6377.

[32] FRIDEN, C.; HERTZ, A.; WERRA, D. de.Stabulus: A technique for finding stable sets in large graphs with tabu search. Computing, v. 42, n. 1, p. 35–44, Mar 1989. ISSN 1436-5057.

[33] FLEURENT, C.; FERLAND, J. A. Genetic and hybrid algorithms for graph coloring. Annals of Operations Research, v. 63, n. 3, p. 437–461, Jun 1996. ISSN 1572-9338.

[34] JAGOTA, A.; SANCHIS, L. A. Adaptive, restart, randomized greedy heuristics for maximum clique. Journal of Heuristics, v. 7, n. 6, p. 565–585, Nov 2001.

[35] GROSSO, A.; LOCATELLI, M.; CROCE,

F. D. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. Journal of Heuristics, v. 10, n. 2, p. 135–152, Mar 2004. ISSN 1572-9397.

[36] ORDÓÑEZ-GUILLÉN, N.; MARTÍNEZ-PéREZ, I. Heuristic search space generation for maximum clique problem inspired in biomolecular filtering. Journal of Signal Processing Systems, v. 83, 08 2015.

[37] ZÜGE, A. P.; CARMO, R. On comparing algorithms for the maximum clique problem. Discrete Applied Mathematics, Elsevier, v. 247, p. 1–13, 2018.

[38] 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.

[39] ÖSTERGÅRD, P. R. A fast algorithm for the maximum clique problem. Discrete Applied

Mathematics, Elsevier, v. 120, n. 1-3, p. 197–207, 2002.

[40] KONC, J.; JANEZIC, D. An improved branch and bound algorithm for the maximum clique problem. proteins,v. 4, n. 5, 2007.

[41] TOMITA, E.; KAMEDA, T. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. Journal of Global optimization,

Springer, v. 37, n. 1, p. 95–111, 2007.

[42] TOMITA, E. et al. A simple and faster branch-and-bound algorithm for finding a maximum clique. In: SPRINGER. International Workshop on Algorithms and Computation.[S.l.], 2010. p. 191–203.

[43] LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, v. 3, p. 43 – 58, 2016. ISSN 2214-7160.

Brazilian Journal of Development

ANEXO

Tabela 1- Resultados obtidos com a execução do algoritmo CPH e a média da solução do algoritmo HS retiradas da referência [17].

Instâncias DIMACS	Solução Ótima	Média da Solução CPH	Desvio Padrão da Solução CPH	Média do Tempo CPH	Desvio Padrão do Tempo CPH	Melhor Solução CPH	Média da Solução HS
Brock200.1	21	20,14	0,35	0,57	0,01	21	20
Brock200.2	12	11	0,56	0,39	0,01	12	10.7
Brock200.3	15	13,96	0,66	0,46	0	15	11.3
Brock200.4	17	15,84	0,54	0,49	0	17	16.10
Brock400.1	27	23,24	0,58	1,25	0,01	25	20
Brock400.2	29	23,64	1,21	1,25	0,01	29	24.1
Brock400.3	31	23,56	1,25	1,25	0,01	31	20.9
Brock400.4	33	23,48	0,61	1,25	0	25	24
Brock800.1	23	19,12	0,32	2,43	0,03	20	10.4
Brock800.2	24	19,26	0,48	2,5	0,11	20	19.3
Brock800.3	25	19,16	0,54	2,6	0,04	21	9.8
c125.9	34	34	0	0,55	0	34	33.6
c500.9	57	51,36	0,95	2,91	0,03	54	53.9
c1000.9	68	59,12	1,49	9,43	0,82	62	62.7
c2000.5	16	14,9	0,3	8,09	0,65	15	14.7
c2000.9	77	66,04	1,25	35,04	2,27	69	68.4
c4000.5	18	15,88	0,43	20,25	0,16	17	15.4
cfat200_1	12	12	0	0,27	0	12	12
cfat200_2	24	24	0	0,53	0	24	24
cfat200_5	58	58	0	1,48	0	58	58
cfat500_1	14	14	0	0,69	0	14	14
cfat500_2	26	26	0	1,47	0,11	26	26
cfat500_5	64	64	0	4,06	0,12	64	64
DSJC500_5	13	12,48	0,5	1,09	0,07	13	12.1
DSJC1000_5	15	13,68	0,47	2,88	0,37	14	13.7
gen200_p0.9_55	55	55	0	0,99	0,03	55	45.9
gen400_p0.9_55	55	48,28	0,92	2,3	0,13	50	39.7
gen400_p0.9_65	65	53,12	6,65	2,34	0,13	65	50
gen400_p0.9_75	75	69,94	9,72	2,4	0,11	75	55.8
hamming6_2	32	32	0	0,35	0,02	32	32
hamming6_4	4	4	0	0,11	0	4	4
hamming8_2	128	128	0	3,28	0,09	128	128
hamming10_2	512	507,46	14,86	75,35	2,91	512	512
hamming10_4	40	37,72	1,68	10,73	15,17	40	38.3

Brazilian Journal of Development

johnson8_44	14	14	0	0,21	0	14	14
johnson16_24	8	8	0	0,2	0	8	8
johnson32_24	16	16	0	1,13	0	16	16
keller4	11	11	0	0,36	0	11	11
keller5	27	25,52	0,78	2,79	0,32	27	26.4
keller6	58	49,32	1,32	45,05	1,77	52	52.9
mann_a9	16	16	0	0,19	0	16	16
mann_a27	126	122,96	0,56	7,6	0,14	124	125.4
mann_a45	345	335,9	0,67	111,65	3,87	337	342.2
phat300_1	8	8	0	0,41	0,01	8	8
phat300_2	25	25	0	1,05	0,01	25	24.8
phat500_1	9	9	0	0,73	0,03	9	6
phat500_2	36	35,88	0,32	2,33	0,07	36	15
phat500_3	50	48,74	0,59	2,7	0,07	50	33.7
phat700_1	11	10,76	0,51	1,03	0,01	11	9.4
phat700_2	44	43,86	0,35	4,09	0,2	44	43
phat700_3	62	60,5	0,88	4,74	0,1	62	61
phat1000_1	10	10	0	2,45	0,28	10	5
phat1000_2	46	45,2	0,77	11,6	0,48	46	13
phat1000_3	68	62,98	1,43	14,49	0,94	66	22.9
phat1500_1	12	11,02	0,14	5,25	0,23	12	10.8
phat1500_2	65	62,82	1,16	27,33	2,96	65	63.1
phat1500_3	94	89	2	26,28	4,33	93	92.4
san200_0.7_1	30	28,44	4,22	0,83	0,17	30	18
san200_0.7_2	18	15,44	1,13	1	0,17	18	17.7
san200_0.9_1	70	70	0	1,31	0,02	70	65.6
san200_0.9_2	60	60	0	1,02	0,01	60	46.5
san200_0.9_3	44	41,6	3,37	0,85	0	44	38.3
san400_0.5_1	13	9,34	1,94	10,98	0,58	13	8
san400_0.7_1	40	27,4	8,24	4,78	9,13	40	18
san400_0.7_3	22	16,92	1,18	1,93	0,17	22	14.8
san400_0.9_1	100	98,46	7,62	2,94	0,04	100	78.4
san1000	15	9,38	0,48	282,08	229,9	10	9
sanr200_0.7	18	17,78	0,41	0,52	0	18	11
sanr200_0.9	42	40,8	0,72	0,92	0	42	41
sanr400_0.5	13	12,28	0,45	0,83	0	13	7
sanr400_0.7	21	20,06	0,46	1,14	0	21	16.3