UMA HEURÍSTICA BASEADA EM GRASP PARA O PROBLEMA DO CLIQUE MÁXIMO

Isabel G. Barbosa
DCC-UFMG
isabarbo@dcc.ufmg.br

Daniel C. Miranda
DCC-UFMG
danielcm@dcc.ufmg.br

Rone Ilídio
DCC-UFMG
rone@dcc.ufmg.br

Abstract*— Neste artigo é apresentada uma nova heurística baseada em GRASP para o problema do clique máximo. Experimentos com diversos tipos de grafos são realizados, com o intuito de validar a heurística proposta. São realizadas também comparações com o desempenho do algoritmo aproximado de *Ramsey.

I. *Introdução*

Seja $G = (V, A)$ um grafo não direcionado, onde V é o conjunto de vértices e Aé o conjunto de arestas de G. Um clique C é um subconjunto de V tal que quaisquer dois vértices de C são adjacentes entre si. O problema do clique máximo consiste em obter o clique de maior cardinalidade em G.

Este é um problema clássico de otimização que pode ser encontrado em diversas aplicações do mundo real, como em aplicações de visão computacional e nos sistemas de recuperação de informação [5]. Recentemente, aplicações na área de bioinformática têm se tornado importantes. Em particular, uma aplicação do problema nessa área consiste em encontrar o maior número de códigos de DNA que possuem entre si uma distância de edição de tamanho menor ou igual à k [4]. Para isso, o problema é modelado como um grafo, onde cada código é um vértice e uma aresta é colocada entre aqueles códigos com distância de edição menor ou igual ao valor especificado. O maior clique desse grafo é o resultado esperado para o problema.

Encontrar o clique de maior cardinalidade em um grafo G é um problema *NP-Completo* [6], ou seja, até hoje não foi encontrado um algoritmo com complexidade de tempo polinomial que resolva o problema de forma determinística. Dessa forma, acredita-se ser pouco provável que tal solução seja encontrada. Caso isso ocorra, será provado que *P=NP* [15].

Por ser *NP-Completo*, o problema do clique máximo tem sido explorado de duas maneiras: existem algoritmos exatos que resolvem o problema em tempo não-polinomial e algoritmos que executam em tempo polinomial, mas não garantem a solução ótima. Esta segunda classe é constituída por algoritmos aproximados, heurísticas e metaheurísticas, que têm sido desenvolvidos com o objetivo de obter soluções cada vez mais próximas da solução ótima, em tempo polinomial.

Um dos mais conhecidos algoritmos aproximados para o problema do clique máximo baseia-se na teoria de *Ramsey* [3], garantindo uma taxa de aproximação de $O(|V|/(log|V|)^2)$. No entanto, instâncias do problema consideradas difíceis, e que possuem um número elevado de vértices, tipicamente são resolvidas utilizando-se heurísticas e metaheurísticas [14].

Neste trabalho, para resolver o problema do clique máximo, foi criado um algoritmo baseado na metaheurística GRASP [7][8][9]. Para avaliar tal algoritmo foram realizados testes com os grafos de benckmark da coleção da DIMACs (http://dimacs.rutgers.edu). Os resultados dos testes são comparados com os resultados obtidos pelo algoritmo aproximado de *Ramsey*.

II. *GRASP*

Metaheurística é um conjunto de conceitos utilizados para a criação de diferentes métodos heurísticos aplicados a vários tipos de problemas. A idéia principal por trás de uma metaheurística é escapar de ótimos locais. Esse objetivo pode ser alcançado permitindo a escolha de outros movimentos que não sejam os melhores em um determinado momento. Além disso, a geração de soluções iniciais pode ser realizadas através de mecanismos mais aprimorados do que simplesmente adotar soluções aleatórias [2].

GRASP (*Greedy Randomized Adaptative Search Procedure* ou procedimento de busca gulosa adaptativa aleatória) consiste em uma metaheurística baseada em iterações, que normalmente é utilizada para resolver problemas de otimização combinatória. O procedimento é guloso, uma vez que na construção de uma solução a adição do "melhor" elemento é dada por algum critério que leva em consideração informações locais, não garantindo a otimalidade

```
   procedure GRASP(Max Iterations,Seed)
1    Read Input();
2    for m = 1; . . . ; Max Iterations do
3      Solution Construction(Seed);
4      Solution Local Search(Solution);
5      Update(Solution,Best Solution);
6    end;
7  return Best Solution;
   end GRASP.
```

Código 1. Pseudocódigo de uma metaheurística GRASP

da solução. É adaptativo, pois o elemento escolhido em qualquer iteração ocorre em função dos elementos escolhidos anteriormente.

Cada iteração do GRASP é constituída basicamente de duas fases: fase de construção e fase de busca local. Na fase de construção é criado, de forma gulosa, um conjunto de elementos, sobre o qual um elemento é selecionado aleatoriamente e adicionado a um conjunto solução. Esses passos são repetidos até que uma solução viável seja encontrada. Na fase de busca local a vizinhança dessa soluçao, ou seja, soluções semelhantes a ela, são investigadas até que se encontre a melhor localmente. A melhor solução entre aquelas encontradas em cada iteração é mantida como resultado do procedimento [7][8][9].

O pseudocódigo de um procedimento GRASP pode ser visto no algoritmo exibido no código 1. Tal procedimento recebe como parâmetro o número máximo de iterações que serão realizadas (*Max Iterations*) e uma semente para o fator aleatório (*Seed*). Nas linhas 2, 3 e 4 tem-se a execução da fase de contrução, a execução da fase de busca local e o procedimento que salva a melhor solução encontrada até o momento, respectivamente. Como retorno, tem-se a melhor solução obtida após o término da última iteração.

III. *Trabalhos Relacionados*

Diversos algoritmos exatos para o problema do clique máximo podem ser encontrados na literatura, como algoritmos do tipo "*branch-and-bound*" [4]. Tais algoritmos, a partir de regras pré-definidas, eliminam a análise de vértices e/ou arestas que não levarão à uma solução melhor do que a solução encontrada até o momento. No entanto, a complexidade de tempo para o pior caso desses algoritmos continua sendo exponencial, o que limita a eficiência e aplicação de tais algoritmos a grafos esparsos ou com um número pequeno de vértices. Assim, um número elevado de heurísticas e metaheurísticas têm sido desenvolvidas visando obter soluções cada vez mais próximas da ótima.

Em [10], uma heurística baseada em algoritmos genéticos é proposta como solução para o problema do clique. Esta nova heurística consiste em uma combinação de um algoritmo genético e uma heurística simples que visa otimizar o clique encontrado. O algoritmo genético utiliza uma função de "*fitness*"que descreve apenas uma propriedade do grafo. A heurística aleatória que compõe o algoritmo possui três passos: inicialmente, vértices selecionados de forma randômica são adicionados ao subgrafo, posteriormente, o subgrafo obtido é reduzido à um clique e, por fim, aumenta-se o tamanho do clique por meio de uma heurística *greedy*. A heurística é combinada ao algorimo genético, aplicando os três passos à cada membro da população à cada iteração do algoritmo.

Em [5] é apresentado um algoritmo que realiza uma busca local baseada em uma váriação da busca em profundidade *k-opt local search*. A idéia básica desse tipo de busca é pesquisar uma parte (*k-opt*) significativa de uma determinada vizinhança, realizando movimentos de adição e de remoção conhecidos como *1-opt move.*

Em [1] a metaheurística GRASP é utilizada para encontrar soluções aproximadas para o clique e *quasi-clique* (um subgrafo muito denso) máximos em grafos com um número elevado de vértices. Os autores argumentam que um problema do uso de heurísticas para solucionar o clique máximo é que o ótimo local pode estar distante do ótimo global e, por isso, utilizar um mecanismo como GRASP, busca Tabu, *Simulated Annealing* [11] e algoritmos genéticos pode ser uma solução interessante.

O trabalho de [12] apresenta um algoritmo paralelo exato baseado no paradigma mestre-escravo para o problema do clique máximo para grafos diversos, sem peso e podendo ser direcionados ou não. O algoritmo foi implementado utilizando MPI e incorpora uma variante do GRASP para obter o conjunto independente máximo de vértices.

IV. *Descrição da Heurística Proposta*

Este trabalho propõe uma nova heurística, inspirada na metaheurística GRASP, para a solução do problema do clique. Porém, foi necessário realizar pequenas alterações em relação ao funcionamento padrão dessa metaheurística, para que o algoritmo se adapte melhor ao problema em questão.

A. Algoritmo Baseado na Metaheurística GRASP

Como toda heurística, esse algoritmo, necessariamente, não obtém a resposta ótima. Ele calcula uma solução que tem como objetivo ser o mais perto possível da ótima. Esta solução é obtida em tempo polinomial, ou seja, em um tempo significavelmente inferior à algoritmos exatos para entradas muito grandes.

O algoritmo possui as mesmas fases encontradas na heurística GRASP, contudo, algumas adequações foram realizadas. Na fase de construção do GRASP, uma solução é obtida de forma aleatória a partir de uma lista de elementos anteriormente gerada por meio de uma estratégia gulosa. Na heurística proposta, esta fase foi implementada como uma busca aleatória por um vértice no grafo. Esse vértice é sempre escolhido entre os 10% (dez por cento) dos vertices que possuem maiores graus, ou seja, entre os vértices mais conectados do grafo. Essa opção parte da hipótese de que um vértice com grau elevado possui uma maior probabilidade de fazer parte do clique máximo.

Como no GRASP, esse algoritmo também possui uma fase de busca local. Nesta fase, a partir do vértice obtido na fase anterior e dos vértices adjacentes a ele, são contruídos e analisados vários cliques, para que o maior seja salvo. O pseudocódigo da heurística proposta pode ser visto no código 2.

As linhas 4 e 5 correspondem à fase de contrução, onde um vértice v_0 é obtido aleatoriamente e seus adjacentes são salvos em uma lista. A fase de busca local é executada nas linhas 6, 7, 8, 9 e 10. As operações da busca local são realizadas *a* vezes, onde *a* corresponde ao número de vértices adjacentes à v_0. Em cada execução, coloca-se o primeiro adjacente obtido na última posição da lista (linha 10). Essa operação tenta evitar que o algoritmo encontre uma máxima local, uma vez que o resultado depende da ordem de inserção dos vértices adjacentes.

Essas duas fases são repetidas m vezes, ou seja, são escolhidos aleatoriamente m vértices e realizadas buscas locais para cada um deles, onde m é um parâmetro informado pelo usuário.

Análise de complexidade: No pior caso, que ocorre quando o grafo $G(V, A)$ é completo, esse algoritmo possui complexidade igual a $O(m * |V|^2)$. A fase de construção possui complexidade $O(max((|V|log|V|), (|V|/10))) = O(|V|log|V|)$, pois ela é composta por uma ordenação dos vértices $(O(|V|log|V|))$ e a seleção aleatória de um entre os 10% dos $|V|$ vértices de grau mais elevado$(O(|V|/10))$. A fase de busca local insere $|V|$ vezes $|V| - 1$ vértices ao clique que está sendo montado, ou seja, sua complexidade é $O(|V|^2)$. Como essas fases são executadas separadamente *m* vezes, obtém-se uma complexidade final no pior caso de $O(m * ((|V|log|V|) + |V|^2)) = O(m * |V|^2)$.

V. *Experimentos*

Para avaliar o desempenho da heurística proposta foram realizados testes com dois grupos de grafos: grafos gerados de forma aleatória e grafos de *benchmark* da coleção da *DIMACs*. Esses benchmarks são compostos de grafos de diversos tamanhos e densidades, para os quais o tamanho máximo do clique é dado. A tabela 1 apresenta maiores informações sobre os benchmarks. Duas medidas foram utilizadas como indicadores para as análises: o tempo de execução da heurística e o tamanho do clique encontrado. Todos os experimentos foram realizados em uma máquina Pentium 4 2.4MHz com tecnologia *hyperthread*, 512Mb de memória RAM, operando com o sistema operacional *Suse Linux 10*. Os algoritmos foram executados 10 vezes e utilizou-se a média dos valores para serem plotadas no gráfico.

Benchmark	Vértices	Arestas	Maior clique
A	450	83198	30
B	595	148859	35
C	760	247106	40
D	945	386854	45
E	1150	580603	50
F	1272	714129	53
G	1400	869624	56
H	1534	1049256	59

TABLE I
CARACTERÍSTICAS DOS *Benchmark*

Com a finalidade de avaliar o comportamento da heurística desenvolvida, esta foi comparada com o *Ramsey*, uma solução proposta por [3] que consiste de um algoritmo aproximado baseado na exclusão de subgrafos. A complexidade de tempo do Ramsey é $O(|A| * n * logn)$, conforme pode ser visto em [13]. Foram utilizados os valores 3 e 10 para o parâmetro *m* da heurística.

A figura 1 apresenta a proximidade do tamanho do clique encontrado em relação à solução ótima normalizada em 100%. Verifica-se que o tamanho dos cliques obtidos pela heurística aqui proposta são em média cerca de 80% do tamanho do resultado ótimo, contra cerca de 70% dos resultados apresentados pelo algoritmo *Ramsey*.

```
    procedimento HeuristicaProposta(Número de iterações,Semente)
1       Le Entrada();
2       Constroi a lista com 10% dos vértices de maior grau
3       para m = 1 até Número de Iterações faça inicio
4           Seleciona um vértice aleatoriamente entre os de maior grau e o insere na solução;
5           Cria uma lista ordenada pelo grau com os vertices adjacentes ao escolhido
6           para a = 1 até o número de vértices adjacentes faça
7               para cada vértice adjacente
8                   Se formar um clique adiciona o adjacente na solução;
9                   Verifica se é a melhor solução já encontrada;
10              Coloca o primeiro elemento da lista na última posição
11      fim;
12      Retorna a melhor solução
    fim.
```

Código 2. Pseudocódigo da Heurística Proposta

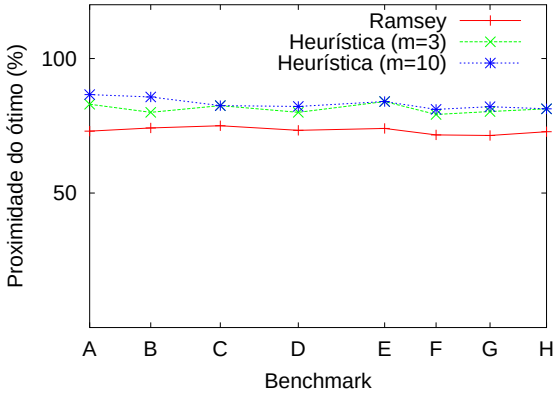

Fig. 1. Comparação em relação ao ótimo.

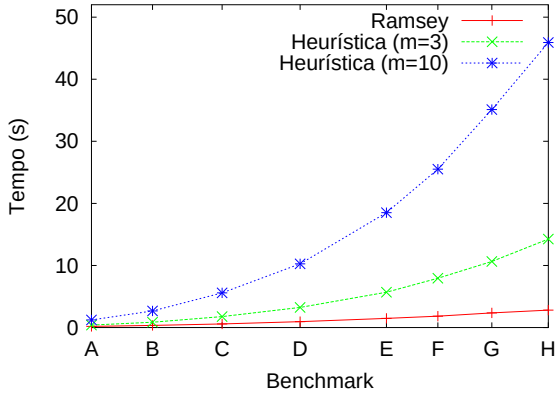

Fig. 2. Comparação em relação ao tempo de execução.

A figura 2 apresenta o gráfico com o tempo de execução dos algoritmos avaliados. É interessante notar que com o aumento do número de vértices e arestas dos grafos também aumentou o tempo de execução dos algotitmos, como era de se esperar de acordo com a análise de complexidade dos mesmos. Verifica-se, então, que o tempo de execução da heurística foi sempre superior ao tempo do *Ramsey*, para os valores de *m* utilizados.

Na figura 3 é apresentado o tempo de execução dos algoritmos com aumento do número de arestas. Os grafos utilizados para esses experimentos mantiveram o número de vértices em 1000, enquanto o número de arestas variava, ou seja, aumentou-se a densidade dos grafos. Nota-se que a heurística obteve tempos de execução menores que *Ramsey* para grafos menos densos. Contudo, verifica-se que com densidades mais altas a heurística aumenta o tempo de execução de forma considerável. No gráfico da figura 1 esse comportante também pode ser visto, uma vez que os grafos utilizados nesses experimentos possuiam densidades mais altas que os gráficos utilizados nos experimentos da figura 3.

Esse comportamento da heurística pode se entendido pelo fato de com grafos mais densos os conjuntos de onde os clique são obtidos são maiores, aumentando com isso o tempo de execução da fase de busca local. Esse aumento foi mais acentuado quando a heurística assume valor de $m = 10$, o que já era esperado devido ao fato de que quanto maior for esse valor maior será o número de iterações realizadas (seção 4).

O gráfico da figura 4 mostra as tamanhos dos cliques obtidos nos experimentos mencionados acima. Verifica-se que o *Ramsey* obteve cliques menores ou iguais aos obtidos pela heurísticas em todos os experimentos realizados. Além disso, pode-se verificar que o parâmetro *m* pouco contribuiu na qualidade das soluções encontradas.

VI. *Conclusões e Trabalhos Futuros*

O objetivo inicial desse trabalho foi criar uma heurística para a solução do problema do clique.

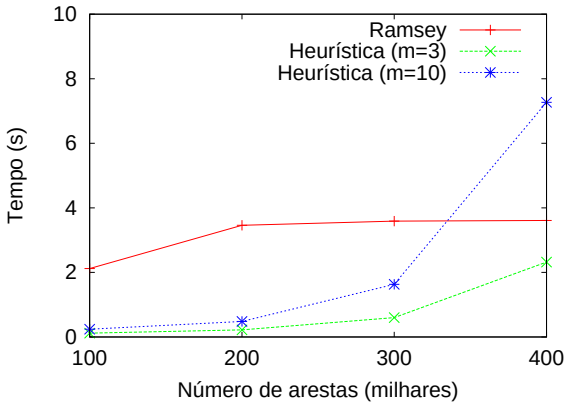

Fig. 3. Tempo de execução para o aumento do número de arestas

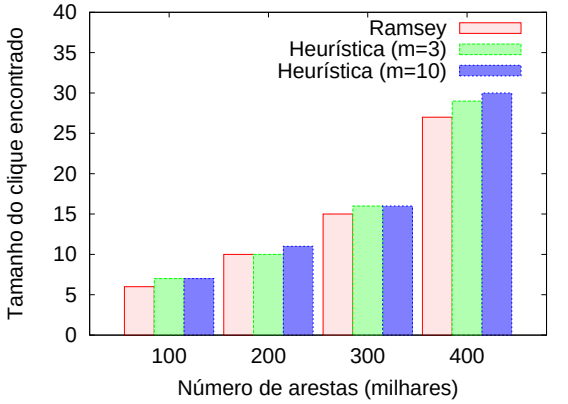

Fig. 4. Tamanho do clique encontrado com a variação da densidade

Foi então apresentado um algoritmo onde foi implementado uma heurísticas baseada na metaheurística GRASP para solução de tal problema. Com a finalidade de avaliar o desempenho da heurística proposta, foram realizados diversos experimentos, nos quais utilizou-se o algoritmo de *Ramsey*, bem conhecido na literatura, como fator de comparação de desempenho.

Em praticamente todos os experimentos, a heurística aqui proposta obteve cliques iguais ou maiores do que os cliques encontrados pelo algoritmo *Ramsey*. Nos experimentos, também pode-se perceber que o desempenho da heurística em relação ao tempo de execução é sensível ao aumento da densidade do grafo. Verifica-se que para grafos maiores e com uma densidade elevada o algoritmo *Ramsey* executou mais rapidamente. Porém, em grafos cuja densidade era menor, a heurística obteve tempos de execução menores.

Como trabalhos futuros, podem ser criados e implementados novos procedimentos que realizam as fases de contrução e de busca local da heurística de forma mais eficiente. O desempenho do algoritmo pode ser reavaliado, utilizando para isso comparações com outros algoritmos encontrados na literatura.

O ponto mais importante que pode ser verificado em trabalhos futuros é a paralelização do algoritmo. De acordo com diversos autores, algoritmos que utilizam GRASP são facilmente paralelisáveis. A heurística aqui proposta também possui essa característica, uma vez que as fases que compõem o algoritmo podem ser executadas em processadores direntes de forma totalmente independente. Pode-se então criar um paralelismo de controle que provavelmente diminuiria o tempo de execução do algoritmo.

REFERENCES

[1] J. Abello, P.M. Pardalos, and M.G.C. Resende. On maximum clique problems in very large graphs. In J. Abello and J. Vitter, editors, *External memory algorithms and visualization*, volume 50 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 119–130. American Mathematical Society, 1999.

[2] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[3] Ravi Boppana and Magnús M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. In J. R. Gilbert and R. Karlsson, editors, *SWAT 90 2nd Scandinavian Workshop on Algorithm Theory*, volume 447, pages 13–25, 1990.

[4] Chad Brewbake. Parallel implementation of the cliquer algorithm for computing the max clique of a random graph of order 900. 2005.

[5] S. Busygin, S. Butenko, and P. Pardalos. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere, 2002.

[6] Tomas H. Cormen. Algoritmos teoria e pratica. In *Algoritmos*, 2002.

[7] Fabiano V. de Alvarenga and Marcelo L. Rocha. Uma metaheurística grasp para o problema da Árvore geradora de custo mínimo com grupamentos utilizando grafos fuzzy. *INFOCOMP Journal of Computer Science*, 5(1):66–75, 2006.

[8] C. C. de Souza and E. M. Macambira. Um grasp para o problema da clique máxima com peso nas arestas. *XIX Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, Outrubro 1997.

[9] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6 (1995), 109–133.*, 1995.

[10] Elena Marchiori. A simple heuristic based genetic algorithm for the maximum clique problem. In *SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing*, pages 366–373, New York, NY, USA, 1998. ACM Press.

[11] Leslie Mateus. Grasp — wikipédia, a enciclopédia livre, 2005. [Online; accessed 22-Junho-2006].

[12] P. Pardalos, J. Rappe, and M. Resende. An exact parallel algorithm for the maximum clique problem. *High Performance Algorithms and Software in Nonlinear Optimization*, 1998.

[13] Vangelis T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Comput. Surv.*, 29(2):171–209, 1997.

[14] W. Pullan and H.H. Hoos. Dynamic local search for the maximum clique problem, 2006.

[15] Nivio Ziviani. Projeto de algoritmos. In *Projeto de Algoritmos - Com implementação em Pascal e C*, 2004.