

Iterated Local Search no Problema da Clique Máxima

Gabriel Cardoso de Carvalho

Resumo: Esse artigo mostra os resultados de uma implementação do Iterated Local Search no problema da Clique Máxima baseada em decisões aleatórias com o objetivo de comparar com os resultados de outras implementações disponíveis na literatura para as instâncias do DIMACS. Os resultados são, como esperado, piores do que em soluções gulosas, porém espera-se que uma solução aleatória seja mais rápida, já que menos decisões são tomadas.

1 Introdução

O problema de encontrar a Clique Máxima (CM) é extremamente conhecido e estudado, pois inúmeros problemas práticos de diversas áreas diferentes, como biologia computacional, economia e análise de redes sociais podem ser modelados como CM. Além disso, a sua versão de decisão foi um dos primeiros problemas a serem provados NP-Completo.

Ele pode ser definido da seguinte maneira, seja o grafo $G = (V, E)$ onde $V = 1, 2, \dots, n$ é o conjunto de vértices e $E \subseteq V \times V$ é o conjunto de arestas, uma Clique $C \subseteq V$ é tal que $\forall i, j \in C, (i, j) \in E$, ou seja, todos os vértices em C são adjacentes entre si. Ou ainda, C é um subgrafo completo de G . O problema da clique máxima é o problema de encontrar a clique de cardinalidade máxima do grafo G .

Diversas soluções foram propostas, tanto métodos exatos quanto heurísticas e metaheurísticas [2, 3, 4]. As propostas no geral tendem a utilizar o sistema *breach and bound* nos métodos exatos e heurísticas gulosas em buscas locais, preferindo vértices de maior grau. De maneira geral, os algoritmos gulosos partem de uma clique C inicial que contém apenas um vértice e um conjunto N_C de vértices $v \in V$ que são os vértices vizinhos à C , ou seja, $\forall u \in C, v$ é adjacente à u . Daí o algoritmo adiciona vértices de N_C em C , escolhendo sempre o vértice de N_C que tem o maior grau no subgrafo $G(N_C)$, até que C seja *maximal*, ou seja, até que não exista uma clique C' maior que C tal que $C \subseteq C'$. Em outros trabalhos esse método é chamado de *Busca Local 1-opt* [5].

A metaheurística implementada nesse artigo é a *Iterated Local Search (ILS)*, que pode ser resumida como uma metaheurística que cria, de maneira iterativa, uma sequência de soluções geradas por uma heurística interna (ou busca local) [6]. É esperado que as soluções providas pelo ILS sejam melhores do que uma

simples repetição da heurística de maneira aleatória.

Nesse artigo é proposta uma implementação do ILS focada na aleatoriedade, de modo a comparar seu desempenho com métodos gulosos, como a implementação do IKLS de *Katayama* [1], que utiliza a *Busca Local k-opt (KLS)* [5] como busca local, que é uma generalização da busca local 1-opt, onde adiciona-se k vértices à clique por vez, permitindo retirar vértices da clique para isso, de maneira dinâmica, ou seja, o k não é fixo, e uma perturbação baseada na *menor conectividade por arestas (LEC-KLS)*, onde escolhe-se um vértice v que não pertence à C , de maneira que v seja adjacente à menor quantidade de vértices de C . Então, adiciona-se v à C e remove de C todos os vértices que não são vizinhos à v .

A seção 2 apresenta como foi feita a implementação do ILS, enquanto a seção 3 cobre toda a implementação e os resultados experimentais do método sobre as instâncias do DIMACS [4]. A seção 4 Apresenta as conclusões e os trabalhos futuros.

2 ILS

Esta implementação do ILS segue o padrão de Glover [6] utilizando o algoritmo 1, onde *GeraSolucaoInicial* trata-se da função que retorna uma clique maximal, a *BuscaLocal* parte de uma clique maximal e busca nas vizinhanças uma clique maior, *Perturbacao* recebe uma clique maximal e retira uma quantidade aleatória de vértices dessa clique, em alguns casos retirando todos os vértices, caso em que ocorre uma reinicialização, e *CritérioAceitacao* que escolhe se a próxima perturbação será feita na solução antiga ou na atual.

Algoritmo 1: Estrutura do ILS

Data: Grafo G , inteiro n_{iter}
 $s_0 = \text{GeraSolucaoInicial}(G);$
 $s^* = \text{BuscaLocal}(s_0);$
 $k = 0;$
while k for menor que n_{iter} **do**
 $s' = \text{Perturbacao}(s^*);$
 $s^{*'} = \text{BuscaLocal}(s');$
 $s^* = \text{CritérioAceitacao}(s^*, s^{*'});$
 $k_{++};$
end

Os detalhes de cada função são descritos nas subseções seguintes.

2.1 Geração da Solução Inicial

A função *GeraSolucaoInicial*(G) somente escolhe um vértice aleatório v de G e chama a função *geraSolucao*(G, v).

A função $geraSolucao(G, v)$ gera a solução inicial ao criar uma clique C de tamanho 1 utilizando o vértice v . Essa clique é um estrutura que contém os vértices que a compõe, o seu tamanho, e um conjunto N_C de vértices que podem ser adicionados à ela. A partir daí, adiciona-se vértices aleatórios de N_C à C e atualiza seu tamanho e N_C até que a clique C seja maximal, como mostra o algoritmo 2.

Algoritmo 2: função $geraSolucao$

Data: Grafo G , vértice v
 $s_0 =$ clique contendo v ;
 $N_{s_0} = V_v$;
 $k = 1$;
while $N_C \neq \emptyset$ **do**
 $u =$ vértice aleatório de N_C ;
 $s_0 = s_0 + u$;
 $N_{s_0} = N_{s_0} \cap V_u$;
 k_{++} ;
end
return clique s_0 maximal

No algoritmo 2, N_{s_0} representa os vértices que podem ser adicionados na clique s_0 , enquanto V_v representa os vértices adjacentes ao vértice v . O algoritmo para quando não há mais nenhum vértice que possa ser adicionado a C .

Além de ser utilizada para criar a solução inicial, a função $GeraSolucaoInicial(G)$ é chamada novamente sempre que a Perturbação decide que deve ser feita uma reinicialização.

2.2 Busca Local

A busca local vai receber uma clique C , e a partir dessa clique, vai tentar melhorá-la olhando para as vizinhanças de C . Temos duas vizinhanças, N_1 e N_2 que são explicadas mais a frente. Elas são conjuntos de cliques vizinhas a C , dada alguma propriedade. Dados esses conjuntos, a busca local irá simplesmente maximizar cada uma das cliques e escolher uma delas.

Definimos aqui $N_i(C)$ como a vizinhança da clique C , tal que, $\forall C' \in N_i(C)$, $C' \subset C, k(C') = k(C) - i$. Portanto as vizinhanças que usaremos, N_1 e N_2 , são subcliques de C com tamanhos $k - 1$ e $k - 2$, respectivamente, onde k é o tamanho da clique C .

N_1 tem tamanho k , ou seja, temos um vizinho para cada vértice de C . Para N_2 , por outro lado temos $\binom{k}{2}$ vizinhos, ou ainda, $\frac{k^2}{2}$. Dessa forma, é possível utilizar a técnica de *Best Improvement* (testar todos e escolher o melhor) para N_1 , porém para N_2 , se faz necessário utilizar *First Improvement* (escolher o primeiro que melhorar a solução atual). O algoritmo 3 mostra o pseudo-código da busca local.

2.3 Perturbação

2.4 Critério de Aceitação

3 Resultados Experimentais

4 Conclusão

References

- [1] Katayama, Kengo, Masashi Sadamatsu, and Hiroyuki Narihisa. "Iterated k-opt local search for the maximum clique problem." *Lecture Notes in Computer Science* 4446 (2007): 84.
- [2] Wu, Qinghua, and Jin-Kao Hao. "A review on algorithms for maximum clique problems." *European Journal of Operational Research* 242.3 (2015): 693-709.
- [3] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization (suppl. Vol. A)*, pp. 1-74. Kluwer, 1999.
- [4] D.S. Johnson and M.A. Trick. *Cliques, Coloring, and Satisfiability*. Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.
- [5] K. Katayama, A. Hamamoto, and H. Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, Vol. 95, No. 5, pp. 503-511, 2005.
- [6] Glover, Fred W., and Gary A. Kochenberger, eds. *Handbook of metaheuristics*. Vol. 57. Springer Science & Business Media, 2006.