Universidade de Brasília

Instituto de Ciência Exatas
Departamento de Ciência da Computação

Análise de Algoritmos:

Problema de *Minimum Vertex Cover*

Curso: Mestrado Profissional em Computação Aplicada

Disciplina: Algoritmos e Estruturas de Dados

Professor: Edison Ishikawa

Alunos: Denise Soares Magalhães
João Laterza
Lucas Kuniyoshi

BRASÍLIA, DF
Março, 2021

SUMÁRIO

1. INTRODUÇÃO

Considere um grafo não direcionado $G(V, E)$, sendo V o conjunto de vértices e E o conjunto de arestas ligadas por dois vértices (u, v) em V. O *Vertex Cover* (VC) é um subconjunto de vértices $S \subseteq V$ que conecta todas as arestas de G, tal que $u \in S$, ou $v \in S$, ou ambos (KUMAR et al., 2009).

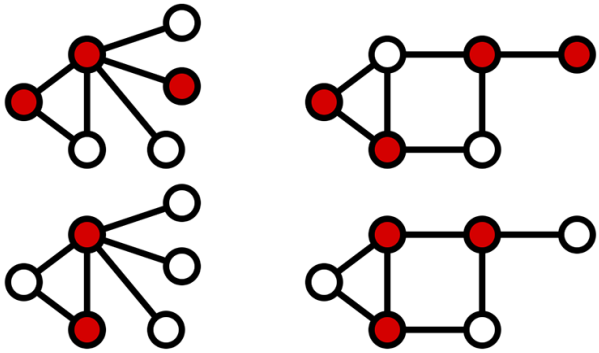


Figura 1 - Exemplos de Vertex Cover em grafos não direcionados
Fonte: https://en.wikipedia.org/wiki/Vertex_cover

O VC é um problema combinatorial com aplicação prática em diversas áreas, como redes de computadores e bioinformática. (EVANS, 1998). Os principais problemas de VC se remetem à verificação, em um dado grafo, se nele está o contido um VC de tamanho k especificado, ou na busca pelo menor VC em um grafo, também conhecida como problema do *Minimum Vertex Cover* (MVC) (KUMAR, 2009).

Sendo o VC um problema NP-Completo, resolvê-lo por meio de uma solução exata em tempo polinomial seria improvável. Uma forma de contornar essa situação seria com valores de entrada pequenos, de modo que a resolução em tempo exponencial de um algoritmo ainda fosse factível. Outra saída seria por meio de soluções quase ótimas, que, na prática, costumam ser boas o suficiente, recorrendo-se, para isso, a algoritmos aproximados (LEBZA e AYMEN, 2019).

Uma das abordagens mais práticas para a otimização é por meio de algoritmos heurísticos. Abordando metodologias de pesquisa poderosas e flexíveis, estes algoritmos têm enfrentado com sucesso problemas práticos difíceis, buscando produzir soluções de boa qualidade em tempo de processamento razoável, i.e., bom o suficiente para fins práticos. Meta-heurísticas são adaptações de heurísticas que comumente buscam adaptar ideologias de outras ciências e as combinam com resultados práticos.

O Algoritmo Genético (GA) é um algoritmo meta-heurístico inspirado nas ideias evolutivas de seleção natural e genética (KOTECHA e GAMBHAVA, 2003). Tratam-se

de procedimentos de busca para convergir em uma solução ótima baseando-se na teoria de sobrevivência do mais apto (BHASIN e AHUJA, 2012). O GA tem sido aplicado a uma grande variedade de problemas de otimização combinatória, incluindo o MVC (HAN e KIM, 2008). Diversas implementações de GA para o problema de VC têm sido realizadas, com reporte de soluções de excelente qualidade em tempo factível para diversos grafos (EVANS, 1998).

O presente trabalho tem por objetivo estudar o problema NP-Completo de otimização, MVC, propondo um algoritmo exato para solução do problema e comparando o seu desempenho com os de algoritmos aproximados obtidos no contexto da meta-heurística do algoritmo genético, de modo a propor a melhor abordagem para uma aplicação prática envolvendo a instalação de câmeras de vigilância em um bairro com alto índice de criminalidade.

2. ALGORITMO EXATO

O objetivo do MVC é encontrar um VC de menor tamanho possível k para G. O MVC requer o desenvolvimento de todos os subconjuntos de G possíveis para separar aqueles que se enquadrem como VC. Assim, MVC é um tipo particular de problemas computacionais denominado problema de otimização (LEWIS, 1983).

Uma forma direta de resolver o problema MVC de um grafo G é remover vértices iterativamente e verificar, a cada remoção, se o grafo resultante G' ainda satisfaz o conceito de VC. Na última iteração, i.e., quanto todas as combinações já foram testadas, o conjunto de grafos VC de menor quantidade de vértice é a resposta para o MVC.

Uma forma de realizar estas iterações é por meio de Busca em Profundidade (*Depth-First Search* - DFS). Como mostra a figura 2, o processo pode ser ilustrado como uma árvore de decisão. Se o grafo G possui uma quantidade de vértices $|V| = n$, então, em um ambiente iterativo de verificação, o algoritmo retira o primeiro vértice (n_1), verificando se o grafo resultante ainda é um VC. Se sim, o processo de DFS é mantido, sendo possível retirar outros vértices $n_{12}, n_{123}, \ldots\ n_{12\ldots m_1}$ e fazendo as suas respectivas verificações. Ao chegar em m_1, ponto de parada para a iteração iniciando no vértice 1, o algoritmo aponta que aquela DFS não retorna mais um grafo resultante que é VC, sendo, então, possível iniciar as combinações resultantes da remoção do segundo vértice, n_2. A iteração irá continuar até o último vértice do grafo, em seu respectivo ponto de parada para a iteração $n_{n(n+1)\ldots m_n}$.

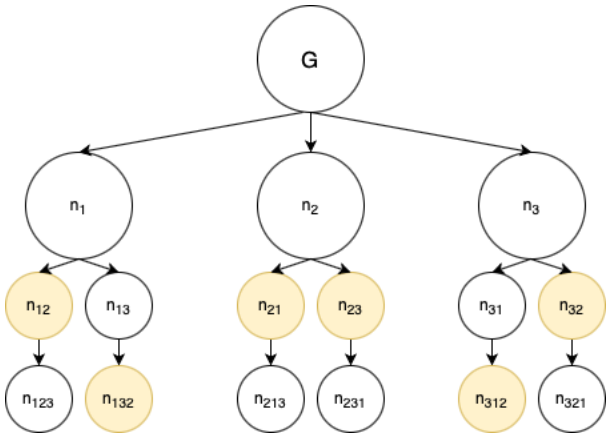


Figura 2 - Grafo com |V| = 3 em formato de 'V'. Em amarelo os pontos de parada, sendo o último círculo branco antes do ponto de parada o VC retornado. O MVC será o VC com a menor quantidade de vértices.
Fonte: elaborado pelos autores.

O algoritmo 1 fornece os passos para a obtenção do MVC, sendo sua entrada um grafo G, com n vértices. A cada iteração, é listado o grafo modificado de G, que é VC do mesmo, se este tiver tamanho menor ou igual dos grafos já listados. Pode-se reparar que, no pior cenário, esta iteração terá $O(p(n) * n!)$, sendo o $p(n)$ função polinomial para comparar se o grafo modificado de G é VC, emitir certificados e verificar se o grafo resultante é menor que os grafos já listados. Já a parte fatorial resulta da busca por todas as possibilidades de combinações.

Algoritmo 1: MVC (G)

Entrada: Um grafo $G = \{V, E\}$

Saída: $G' = \{S, E'\}$, sendo que (1) cada aresta em G tem ao menos um vértice representante em S e (2) não é possível remover nenhum vértice de S sem violar (1).

1 Verifica se $G_{modificado}$ é um VC de $G_{original}$

Se sim, verifica se é a primeira recursão

Se sim, anexa $G_{modificado}$ em G'

Se não, verifica tamanho de $G_{modificado}$

2 Se menor que o tamanho do grafo em G'

Remove grafo em G'

Anexa $G_{modificado}$ em G'

3 Se tamanho de $G_{modificado}$ é igual ao G'

Verifica se $G_{modificado}$ está em G'

Se não, anexa $G_{modificado}$ em G'

4 Para cada vértice V_i em $G_{modificado}$

Remover V_i de $G_{modificado}$, tal que:

$G_{modificado(novo)} = G_{modificado} - V_i$

Aplicar o processo de MVC($G_{modificado(novo)}$)

---

3. CLASSE DE COMPLEXIDADE DO PROBLEMA

3.1. *Nondeterministic Polynomial Time* (NP)

A classe de complexidade de um algoritmo pode ser medida de diversas maneiras, que incluem ou não seu tempo de processamento ou o acúmulo de dados na memória. Porém, dado um problema que o custo de processamento e memória é visivelmente alto, (1) como achar um algoritmo que reduza consideravelmente sua complexidade? E caso, não seja possível encontrar tal algoritmo, (2) como provar a intratabilidade? Ambos os problemas podem ser considerados igualmente difíceis. Mas sabendo que, por décadas, uma série de problemas tem sidos abordados por diversos pesquisadores, através de diferentes metodologias, demonstrar que o seu problema se resume a um destes, comprova ao menos que ele é tão difícil quanto. Esta é a maneira de se caracterizar um problema como NP-Completo (LEWIS, 1983).

Embora saber que um problema é NP-Completo não ajuda diretamente em sua solução, ele demonstra claramente quais metodologias podem ser aplicadas para se conseguir resultados adequados, com determinadas restrições.

Para o problema de *Minimum Vertex Cover* (MVC), em uma solução direta, pode-se partir de um vértice, e ir testando quais combinações de outros vértices existem, tais que contenham todas as arestas do grafo (ao menos um vértice desta aresta). Não é possível determinar, ao encontrar um VC do grafo original, que não há um outro subconjunto VC menor antes de testar todas as combinações possíveis. Este tipo de problema é considerado de otimização. O MVC pode ser transformado em um problema de decisão, tipicamente abordado para análise de complexidade: dado um grafo G e um número inteiro k, existe um subconjunto de G com k vértices que cobrem todas as arestas, com ao menos um vértice desta aresta? Este problema pode ser repetido até devolver negativamente, sendo o último G'_n o MVC, e k o número de vértices deste grafo.

No problema MVC, se encontrado um conjunto de soluções, os mesmos podem ter seu certificado verificado através de um algoritmo em tempo polinomial.

Algoritmo 2: Verificador (G, G')

Entrada: Um grafo $G = \{V, E\}$ e um outro grafo $G' = \{S, E'\}$, tal que $S \supseteq V$, sendo que cada aresta em G tem ao menos um vértice representante em S, i.e., G' é assumido como VC de G.

Saída: 'Verdadeiro' (1) se cada aresta em G tem ao menos um vértice representante em S e (2) se não é possível remover nenhum vértice de S sem violar (1).

1 Para cada V_i (u_i, v_i) em G, se $\exists S_j \mid u_i \; ou \; v_i \in S_j$

Retorna $Verdadeiro$

2 Para cada S_j in G', se $G' - S_j$ aplicado em (1) != $Verdadeiro$

Retorna $Verdadeiro$

Assim, demonstra-se que MCV é um problema em NP.

3.2. NP-Completo

Visto que o problema MVC é um problema não determinístico em tempo polinomial (NP), deve-se verificar se o mesmo é NP-Completo, sendo assim possível reduzi-lo para outros problemas NP em tempo polinomial.

Se V' é um Conjunto Independente (CI) de $G = \{V, E\}$, não existem dois vértices adjacentes contidos em V', i.e., para todo para todos vértices em V' não há arestas que os conectem. Como consequência, para cada aresta $e = (u, v)$ em G, pelo menos um vértice de u, v deve estar no subconjunto $(V - V')$. Além disto, nenhum vértice de $(V - V')$ pode ser removido de G sem que uma aresta e seja removida de E. Assim, $(V - V')$ é, por definição, o *Vertex Cover* de G. Como resultado, uma vez que G contenha um CI de tamanho k, G também contém um *Vertex Cover* de tamanho $|V| - k$ (OPENDAS, 2017).

Adicionalmente, pode-se notar que, se G possui um CI de tamanho k, o grafo complementar de G, G_c, possuirá um subconjunto de vértices tais que cada par de dois vértices do subconjunto é conectado por uma aresta, i.e., o inverso de CI. Esse subconjunto é um Clique. Assim, se V' é o Conjunto Independente de G, então existe $V'_c = \bar{V}'$, sendo o tamanho do Clique é equivalente ao tamanho do Conjunto Independente (TAYLOR, 2020).

Diante do exposto, verifica-se que existe uma relação direta entre Clique, CI e VC. Também é possível ter algoritmos de tempo polinomial que mapeiem um problema abordado em um destes tópicos específicos para um outro. Ou seja, dado o problema de VC, é possível ter um algoritmo de tempo polinomial que, usando uma sub-rotina de outro algoritmo que resolve Clique ou CI, forneça a solução. Conclui-se, assim, que VC é ao menos tão difícil quanto encontrar o CI ou o Clique de um grafo, que já foram provados

serem problemas NP-Completo. Consequentemente, VC é também NP-Completo (KARP, 1972;

4. VALIDAÇÃO E DESEMPENHO DO ALGORITMO EXATO

Para verificar se as soluções encontradas pela abordagem exata, descrita no presente estudo, de fato representam os MVC dos grafos inseridos, é necessário testá-la com exemplos onde as soluções são conhecidas. Foram efetuados 12 testes, com base nos exemplos registrados por Ashay Dharwadker em "*The Vertex Cover Algorithm*" (2011). A tabela 1 traz o nome dos grafos usados como *benchmark*, quantidade de vértices e arestas, o MVC conhecido, a quantidade de recursões utilizada pelo algoritmo exato proposto e o MVC obtido.

Tabela 1 - Grafos conhecidos testados no algoritmo exato. Fonte: DHARWADKER, 2011.

Dados conhecidos					**Algoritmo Exato**	
Nº	**Nome do Grafo**	**Vértices**	**Arestas**	**MVC**	**Recursões**	**MVC**
1	**Tetraedro**	4	6	3	16	3
2	**Bipartido de Kuratowski $K_{3,3}$**	6	9	3	120	3
3	**Octaedro**	6	12	4	60	4
4	**Bondy-Murty G_1**	7	12	4	163	4
5	**Grafo de roda W_8**	8	14	5	442	5
6	**Cubo**	8	12	4	688	4
7	**Petersen**	10	15	6	2.560	6
8	**Bondy-Murty G_2**	11	28	7	4.879	7
9	**Grötzsch**	11	20	6	5.911	6
10	**Herschel**	11	18	5	15.283	5
11	**Icosaedro**	12	30	9	1.944	9
12	**Bondy-Murty G_3**	14	21	7	259.000	7

Nota-se que, embora o total de recursões mude conforme a quantidade e disposição dos vértices nos grafos, o algoritmo exato proposto cumpre sua função de encontrar o MVC. Faz-se necessário entender como é o comportamento do algoritmo para diferentes tipos de grafos, não somente pela quantidade de vértices, já que grafos com menos vértices podem ter mais recursões que grafos com mais vértices (e.g., Icosaedro com 12 vértices necessita somente 1.944 recursões, enquanto Herschel, Grötzsch e Bondy-Murty G_2, com 11 vértices, necessitam de 15.283, 5.911 e 4.879 recursões, respectivamente). As ilustrações dos grafos utilizados na tabela 1 encontram-se no anexo A.

4.1. Análise do desempenho por tipo de grafo

Como demonstrado anteriormente, dependendo do tipo de grafo sobre o qual é aplicado algoritmo de solução exata, diferentes quantidades de recursões são necessárias, consequentemente demandando mais ou menos tempo de processamento.

Utilizando um computador com processador Inter(R) Core(TM) i7-8700T CPU 2.40GHz, 32 GB de memória RAM e sistema operacional de 64 bits, Windows 10 Enterprise, e implementando o algoritmo exato através de Python (v3.7.1) com auxílio de bibliotecas como numpy (v1.15.4), pandas (v0.23.4), scipy (v1.2.1) e networkx (v2.5), foram feitos testes de desempenho do algoritmo variando a quantidade de vértices (V) de três tipos diferentes de grafos: (1) completos, (2) bipartidos completos e (3) binomiais.

Nos grafos completos (1) todas as possibilidades de arestas entre os vértices estão presentes. Para os bipartidos completos (2), são selecionados dois conjuntos com a mesma quantidade de vértices (pares), sendo que cada nó de um conjunto está associado a cada nó do outro conjunto, mas não com os nós do próprio conjunto. Para grafos com totais ímpares de vértices, um conjunto apresentará um nó a mais que o outro. Quanto aos grafos binomiais (3) - também conhecidos como grafos de Erdős-Rényi (1960) - foi considerada uma probabilidade de criação de aresta igual a 50% para cada par de vértices.

Foram gerados grafos de 4 a 13 vértices, sendo que, para cada grafo o algoritmo exato foi executado 30 vezes, registrando-se o tempo médio de execução e o respectivo intervalo de confiança (95%) para uma distribuição normal. A tabela 2 mostra os resultados dos testes para cada tipo de grafo. Conforme vértices são inseridos aumenta-se significativamente o tempo de execução em segundos.

A execução do algoritmo exato do problema MVC para grafos completos é a melhor entre os três tipos, uma vez que somente é possível retirar um vértice antes que o subconjunto de vértices deixe de ser um VC do grafo. Assim, na primeira busca por profundidade já é encontrado um MVC. Embora a busca continue, outras alternativas menores não são possíveis.

Para os grafos bipartidos completos, cada subconjunto é solução do problema. Porém para encontrar tal solução, o algoritmo exato, na maneira que foi projetado, tem que passar por uma grande quantidade de possibilidades para que o subconjunto seja separado como MVC.

Tendo em vista a aleatoriedade dos grafos binominais, já que a bipartição é uma configuração bem específica, é razoável que ele opere em média melhor que o segundo tipo de grafo analisado, porém pior que um grafo completo.

Em todo caso, como ilustra a figura 3, independentemente do tipo do grafo, o tempo gasto pelo algoritmo para encontrar a solução exata do problema cresce de forma não polinomial à medida que o número de vértices aumenta. Sendo assim, para grafos com 30 vértices, por exemplo, a análise por meio do algoritmo exato proposto em tempo aceitável pode não ser mais factível.

Tabela 2 - Teste do algoritmo realizado em três tipos de grafos diferentes, variando a quantidade de vértices.

Quantidade de Vértices	Grafo Completo			Grafo Bipartido Completos			Grafo Binomial		
	Tempo médio (s)	IC (s) - 95%	MVC	Tempo médio (s)	IC (s) - 95%	MVC	Tempo médio (s)	IC (s) - 95%	MVC
4	0,0016	0,0000 - 0,0033	3	0,0016	0,0000 - 0,0033	2	0,0005	0,0000 - 0,0015	2
5	0,0068	0,0040 - 0,0096	4	0,0078	0,0046 - 0,0110	2	0,0026	0,0005 - 0,0047	3
6	0,0344	0,0301 - 0,0386	5	0,0229	0,0186 - 0,0273	3	0,0172	0,0149 - 0,0194	4
7	0,1276	0,1193 - 0,1359	6	0,1328	0,1274 - 0,1383	3	0,0771	0,0727 - 0,0815	4
8	0,3312	0,3184 - 0,3441	7	0,4740	0,4608 - 0,4871	4	0,2495	0,2418 - 0,2572	5
9	0,9344	0,8762 - 0,9926	8	2,4854	2,4356 - 2,5352	4	0,9026	0,8701 - 0,9351	5
10	2,1708	2,1384 - 2,2032	9	8,9359	8,8465 - 9,0254	5	3,3958	3,3471 - 3,4446	6
11	4,7031	4,6601 - 4,7462	10	53,5943	52,9575 - 54,2311	5	7,6474	7,5882 - 7,7066	7
12	9,8354	9,4956 - 10,1752	11	169,7927	168,8348 - 170,7506	6	33,2844	33,1332 - 33,4356	7
13	19,3255	18,8670 - 19,7840	12	1.043,388	1.034,8763 - 1.051,8987	6	39,8958	39,2636 - 40,5281	9

No anexo B encontram-se as ilustrações dos grafos, sobre os quais o algoritmo exato para o problema de MVC é aplicado, juntamente com o tempo médio e o respectivo intervalo de confiança (nível de confiança = 95%).

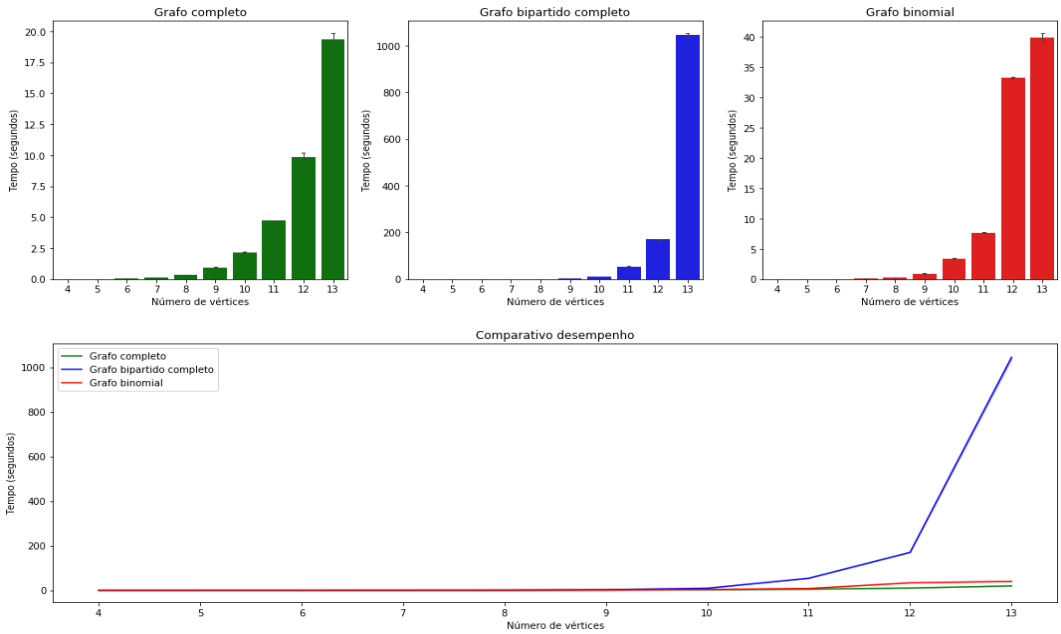


Figura 3 – Gráficos de barra ilustrando o tempo gasto, em segundos, pelo algoritmo exato para encontrar o MVC de grafos (1) Completos, (2) Bipartidos Completos e (3) Binomial (direita à esquerda), com diferentes quantidades de vértices. Abaixo encontra-se o gráfico de linhas, com o tempo de execução para os três tipos de grafos para comparação de performance. Fonte: elaborado pelos autores.

5. ALGORITMOS APROXIMADOS

Conforme verificado, o MVC é um problema NP-Completo, de modo que a obtenção de uma solução ótima seria intratável, i.e., seria improvável encontrar um algoritmo para resolver com exatidão o problema em tempo polinomial. De fato, o algoritmo exato proposto no presente trabalho é executado no tempo de $O(p(n) * n!)$, com n sendo o número de vértices do grafo. A Figura 3 ilustra esse crescimento, em que o código foi executado pelo mesmo computador para resolver o MVC de grafos não-direcionados completos variando-se o valor de n de 4 até 13.

Observa-se que, à medida que o número de vértices cresce, a solução exata deixa de ser factível. Ocorre que, para a maior parte dos problemas NP-completo, apenas instâncias com pequena escala podem ser tratadas usando métodos matemáticos exatos. Diante dessa situação, são comumente usadas alternativas por meio de métodos de aproximação, que podem encontrar uma solução boa o suficiente em um tempo razoável. Esses métodos são implementados por meio de algoritmos aproximados, e podem ser categorizados em heurísticas e meta-heurísticas (AYMEN, 2019; BASSET et al, 2018).

A principal diferença entre essas metodologias de aproximação é que a heurística é mais dependente do problema do que a meta-heurística. Assim, algoritmos heurísticos podem ser aplicados com eficiência a um problema específico, ao mesmo tempo que se tornam insuficientes para outros problemas. Por outro lado, a meta-heurística fornece um *framework* de algoritmo genérico ou um otimizador que pode ser aplicado a quase todos os problemas de otimização (BASSET et al, 2018).

5.1. Métodos aproximados para o problema MVC

Existem diversas heurísticas tradicionais para o problema do MVC. Uma das mais conhecidas emprega um algoritmo guloso que, repetidamente, remove do grafo os vértices de maior grau remanescente, isto é, aqueles nos quais incide o maior número de arestas, adicionando-os, então, ao VC, até que todas as arestas de G estejam nele cobertas. No entanto, ainda que intuitivo, o algoritmo possui uma performance ruim em diversas classes de grafo, não possuindo limites de desempenho fixos (EVANS, 1998).

De modo geral, métodos clássicos de resolução de problemas de otimização são prejudicados por sua incapacidade de resolver funções que não são continuamente diferenciáveis. Os problemas da vida real, no entanto, apresentam questões que têm maior

probabilidade de ter funções objetivas que não são contínuas e, portanto, as aplicações de técnicas clássicas de otimização para esses problemas são limitadas na melhor das hipóteses. Outra questão que incomoda a aceitação das técnicas clássicas é sua tendência de se estabelecer em mínimos ou máximos locais, em vez de em uma solução globalmente ótima. (CHAKRABORTY et al, 2014).

Diversas estratégias foram adotadas buscando contornar esses obstáculos. Uma delas, que tem se mostrado eficaz na implementação de soluções aproximadas para o problema do MVC (KUMAR, 2009), emprega o Algoritmo Genético (GA), cuja validação tem sido objeto de várias análises e estudos submetidos nos últimos anos (BASSET et al, 2018).

5.2. O Algoritmo Genético (GA)

O termo algoritmo genético (GA) foi primeiramente utilizado por John Holland, no livro *Adaptation in Natural and Artificial Systems* de 1975. O GA é uma técnica de otimização baseada na evolução natural. A ideia é manter uma população de soluções candidatas, que evoluirão sobre a pressão seletiva que favorece a sobrevivência de indivíduos que se enquadram em um padrão escolhido, definido por uma *fitness function* – FF (GLOVER e KOCHENBERGER, 2016).

O fundamento do GA opera em uma população de soluções candidatas, denominadas cromossomos. Os cromossomos são fixados com valores binários em cada uma das suas posições características. Esses valores binários ressaltam ou indicam a ausência de características especificas dos cromossomos. Cada cromossomo é verificado através de uma FF. Membros da população com características desejadas são selecionados para produzir novos membros. Esses novos membros, descendentes, irão conter características da geração passadas, sendo que é possível inserir mutações. Assim, são listadas novamente todos os membros, geração progenitora com a sua descendência, através da FF. Existe a possibilidade de membros progenitores mais fracos serem substituídos pelos novos membros gerados. Esse processo é repetido, até uma convergência da população de soluções (OSMAR e LAPORT, 1996).

O GA é constituído de quatro fases: proposta de cromossomos com a avaliação dos mais aptos, através de uma FF; seleção de cromossomos para a reprodução de uma nova geração; aplicação de um cruzamento (*crossover)* e mutação dos cromossomos descendentes gerados; verificação de progresso, i.e., compreender se a nova geração

continua a superar as gerações passadas. O processo será repetido por novas gerações (OSMAR e LAPORT, 1996).

A seleção de progenitores para reprodução é um processo influente para a geração de novas soluções, podendo serem utilizadas diversas técnicas, com a roleta da sorte (RWS) e o torneio (TOS) sendo as mais comumente aplicadas. Durante a fase de cruzamento, parte dos dois cromossomos selecionados são inseridas em um novo cromossomo gerado, podendo esta parte ser selecionada de um ponto, de vários pontos ou uniformemente. Na mutação, partes dos cromossomos são alteradas aleatoriamente, por métodos de embaralhamento, inversão, lançamento de bit e troca. Essa etapa tem por objetivo "escapar" de mínimos ou máximos locais, problema recorrente nos métodos clássicos de aproximação. No entanto, os melhores cromossomos podem ser perdidos durante a criação de novos cromossomos. Para evitar essa situação, um processo de elitismo é usado, i.e., o(s) melhor(es) cromossomo(s) da geração passada são enviados para a nova população (KOTECHA e GAMBHAVA, 2003; BASSET et al, 2018).

Observa-se que, computacionalmente, o GA é um processo de maximização. Em outras palavras, o seu processo garante - ao se substituir, em cada estágio intermediário, a geração antiga pela nova e normalmente melhorada geração - que o ponto máximo global seja encontrado dentre prováveis múltiplos máximos locais no problema abordado. Assim, espera-se que, após serem obtidas diversas soluções intermediárias cada vez melhores, o algoritmo continue se aproximando da solução ótima (CHAKRABORTY et al, 2014).

5.3. Aplicação de GA em problema de MVC

Como explicado anteriormente, dado um grafo $G(V, E)$, sendo V seus vértices e E suas arestas, pode-se encontrar um VC retirando iterativamente vértices e suas respectivas arestas, até que o subconjunto restante (G') seja independente. O subconjunto de vértices S contém todo o conjunto E. S pode ser representado por um certificado binário, onde a presença de um bit seria a ativação do vértice de G. Existe uma abordagem precisa para encontrar um certificado que encontre o VC e seu mínimo, porém seu custo de tempo é muito alto, como demonstrado anteriormente. O GA dá a oportunidade de interagir sobre uma população de certificados de vértices, aplicando um sistema de evolução especializado no princípio de sobrevivência do mais apto, devolvendo uma solução do MVC em tempo polinomial. Tal solução se aproxima da ótima com o aumento

do número de gerações de cromossomos criados (GLOVER e KOCHENBERGER, 2016).

O presente estudo aborda o problema de MVC através de um algoritmo genético híbrido (HGA), que combina técnica de otimização local (LOT) com o sistema clássico do GA. Também é aplicado um operador heurístico de cruzamento de vértices (HVX) para geração de descendentes que se aproxime rapidamente do ótimo global, usando informações levantadas a cada combinação. A mutação é usada após a recombinação para evitar o mínimo local (KOTECHA e GAMBHAVA, 2003).

5.3.1. Operador heurístico de cruzamento de vértice (HVX)

Dado dois cromossomos certificados de soluções que são VC de G, existem diversas maneiras de cruzar suas informações para a produção de um novo cromossomo descendente. Kotecha e Gambhava (2003) propuseram uma abordagem heurística para o processo de cruzamento na reprodução de cromossomos em um algoritmo genético, materializada no algoritmo operador heurístico de cruzamento de vértice (HVX). O algoritmo foi projetado especialmente para o problema de MVC objetivando convergir rapidamente para uma solução ótima, podendo, assim, ser considerado um processo independente.

Quando inserido em um algoritmo genético, o HVX é responsável pela etapa de reprodução de populações no transcorrer de várias gerações. Para isso, recebe dois cromossomos pais P1 e P2, dando origem a um único cromossomo descendente; processo que se repete para todos os pares de cromossomos genitores da população. Por outro lado, para executar o algoritmo de forma independente, é necessário um processo aleatório de geração de cromossomos certificados, em que vértices do grafo são aleatoriamente selecionados e ativados nos cromossomos P1 e P2 até que ambos se tornem VC do grafo G, cobrindo todas as suas arestas.

O processo de cruzamento do algoritmo consiste, basicamente, na classificação dos vértices v dos cromossomos pais P1 e P2 de acordo, primeiramente, com o seu grau ou valência $N(v)$, i .e., o número de arestas que neles incidem, e, posteriormente, com a frequência de ocorrência nos genitores $F(v)$, sendo retirados do grafo G os melhores vértices consecutivamente até que nele não restem mais arestas. O conjunto de vértices resultantes é, por definição, um VC de G. Nos casos de classificação que resultem em empate, é sorteada aleatoriamente um vértice, de modo a evitar sobreposições. Note que,

quando HVX está inserido em um GA, o critério de classificação F(v) tem por objetivo selecionar as características mais recorrentes nos cromossomos genitores, de modo a perpetuá-las nas gerações seguintes. Por outro lado, quando HVX é tratado como uma heurística independente, a abordagem se assemelha à de um algoritmo guloso, uma vez que a estratégia consiste, basicamente, em selecionar os vértices de maior valência, já que os cromossomos genitores sempre serão selecionados aleatoriamente a partir do conjunto de todas as soluções possíveis, sem considerar características evolutivas das populações.

Algoritmo 3: Geração de cromossomos (G)

Entrada: Um grafo $G = \{V, E\}$.

Saída: Um cromossomo P, i.e., um certificado onde cada bit ativo indique a presença do respectivo vértice. Este cromossomo é um subconjunto de G, sendo que cada aresta de G tem ao menos um vértice representante em nele, ou seja, o cromossomo é um VC de G.

1 G_S é uma cópia de G

P é o cromossomo

2 Enquanto a quantidade de arestas em $G_S > 0$:

Escolher dentro da lista de vértices de G_S, um vértice aleatório

Inserir este vértice em P

Retirar este vértice de G_S, junto com suas arestas

3 Retornar o cromossomo P

O algoritmo 4 indica os processos feitos no cruzamento de cromossomos pais, utilizando o grau dos vértices e a frequência com que aparecem nos genitores (KOTECHA e GAMBHAVA, 2003).

Algoritmo 4: Operador heurístico de cruzamento de vértices (HVX) (G, P_1, P_2)

Entrada: Um grafo $G = \{V, E\}$ e cromossomos que serão cruzados P_1 e P_2.

Saída: Um cromossomo P_3 resultado da cominação de P_1 e P_2, que é um VC de G.

1 Criar tabela de vértices – VT ($F(v)$ – frequência que o vértice aparece nos cromossomos P_1 e P_2, $N(v)$ – número de arestas conectadas neste vértice) e uma tabela de arestas – ET (lista de arestas para cada vértice)

2 Enquanto $ET <> 0$:

Selecionar v_1, tal que seja o maior em $N(v)\ \forall\ v\ \in VT$.

Se mais de um v possua o mesmo $N(v)$, escolher o v_1, tal que seja o maior em $F(v)$.

Se ocorrer novamente um empate escolher aleatoriamente entre os vértices de maior $N(v)$ e maior $F(v)$.

Retirar de ET o vértice v e sua presença em outros vértices: ET = ET – $\{E(x,y) \rightarrow x = v_1\ ou\ y = v_1\}$.

Incluir $v_1$em P_3

3 Retornar P_3

Embora a reprodução envolva o cruzamento de P_1 e P_2, é possível que vértices ativos em ambos os cromossomos não sejam inseridos na próxima geração. Isso ocorre quando o critério de cobertura de todas as arestas de G é atendido antes de todos os vértices serem selecionados. Nesse caso, temos uma mutação por omissão, já que determinada característica de ambos os genitores não é transmitida para o cromossomo descendente.

5.3.2. *Otimização local (LOT)*

O principal motivo para se inserir um estágio de otimização local em uma abordagem heurística é melhorar a qualidade da solução encontrada e aprimorar o desempenho do algoritmo, ajudando-o a convergir para boas soluções mais rapidamente. Para isso, existem diversas técnicas de otimização que podem ser modificadas para se ajustar ao problema (PERIANNAN, 2007).

No algoritmo genético híbrido, o processo de otimização local é aplicado a cada geração de descendentes, antes dos mesmos serem inseridos na nova população. Em geral, o processo do GA tende a convergir para o mínimo global, enquanto o processo de otimização local busca melhorar a qualidade dos cromossomos, ainda que convergindo para um mínimo local.

Kotecha e Gambhava, no artigo "*A hybrid algorithm for MVC problem",* de 2003, comparam as etapas de seleção, cruzamento e mutação à evolução biológica descrita por Darwin (1), e a etapa de otimização local, que busca inserir informações inteligentes nos cromossomos, à evolução Lamarckiana (2). Dessa forma, enquanto a evolução de Darwin seleciona e perpetua os mais ajustados nas população no transcorrer das gerações, em que

cada cromossomo pode conter diferentes vértices de G, a evolução de Lamarck busca abordar cada um dos seus descendentes e, iterativamente, retirar seus vértices sem violar a restrição de VC. Se possível, a solução passa a ser otimizada, apresentando maior qualidade; se não, isso significa que a combinação de vértices já não permite mais remoções, sendo uma solução ótima (local ou global). O algoritmo 5 indica os processos da otimização local.

Algoritmo 5: Otimização Local (LOT) (G, P)

Entrada: Um grafo $G = \{V, E\}$ e um cromossomo que é uma solução de VC de G.
Saída: Um cromossomo P_f, que pode possuir a mesma quantidade de vértices de P ou menos.

1 Aleatoriamente ordena os vértices em P

2 Para todo vértice v em P:

Seleciona um P_f que não tem o v

Verifica se P_f é um VC de G

Se sim, verifica se P_f contém apenas um vértice

Se sim, retorna P_f

Se não, faz P igual a P_f e retorna a 1

3 Retorna $P_f = P$

Caso seja possível a remoção de vértices, o cromossomo otimizado será inserido na população. Do contrário, o cromossomo original será perpetuado. Observa-se que o presente algoritmo não tem por objetivo encontrar a melhor possibilidade de remoção de vértices em sequência a partir do cromossomo original. Caso assim fosse, configuraria um algoritmo exato, com tempo de execução não polinomial. Como é selecionado apenas um ponto de partida para o processo de remoção, o tempo de execução do algoritmo é, no pior caso, de $O(n^2)$, com n sendo o número de vértices do cromossomo.

5.3.3. Algoritmo Genético Híbrido (HGA)

Para aplicar os conceitos anteriormente descritos no algoritmo genético híbrido é necessário, primeiro, definir a *fitness function* (FF) e o processo de mutação.

A FF irá inferir o quanto um cromossomo é apto para prosseguir para próximas gerações e produzir novos cromossomos. No presente estudo, além dos cromossomos mais aptos - ranqueados através da FF - fazerem parte dos genitores que produzirão, por

meio de cruzamento (HVX), os cromossomos descendentes, os mesmos serão também inseridos na nova geração, com uma taxa de elitismo de 50% (GLOVER e KOCHENBERGER, 2016). Assim, no transcorrer de cada geração, a nova população gerada será composta pelos descendentes dos cromossomos da população anterior (50%), bem como pelos seus cromossomos mais aptos (50%). Para o processo de seleção, uma FF direta para um GA que busca a solução de um MVC é a contagem dos vértices presentes no cromossomo analisado. Quanto menor a contagem, mais apto o cromossomo está para ser escolhido (KOTECHA e GAMBHAVA, 2003).

A mutação tem por objetivo impedir o assentamento das soluções em mínimos locais, por meio da inserção de características diferentes nos descendentes da população. Para isso, é ativado ou desativado um bit (vértice) do cromossomo gerado. No caso de mutações com acréscimo de bit, o cromossomo passará a ser menos apto do que anteriormente, dado o critério de seleção da FF. A situação, contudo, será temporária, já que os vértices redundantes serão removidos na etapa de otimização. Quanto às mutações com exclusão de bits, estas somente ocorrerão caso o cromossomo mutado ainda seja um VC. Após este processo é então aplicada a LOT.

O HGA deve receber dois parâmetros: o tamanho da população (*P*) e o número de gerações (*t*). Esses parâmetros são calibrados de acordo com a complexidade do grafo a ser analisado. Quanto maior os seus valores, mais iterações serão efetuadas, demandando mais tempo para o algoritmo retornar uma solução. Contudo, maior será a chance de ele convergir para o mínimo global, isto é, a solução de MVC exata do grafo.

Com os parâmetros de entrada, é necessário, primeiramente, determinar a população de cromossomos na primeira geração. Para isso recorre-se ao algoritmo 3, de modo a gerar, aleatoriamente, cromossomos certificados, processo que ocorrerá repetidamente até que a população alcance o tamanho do parâmetro *P* informado. Em seguida os cromossomos serão ranqueados através da FF e organizados em pares para a reprodução por meio do cruzamento com o HVX. A metade mais apta da população inicial passará a constituir a nova geração junto dos descendentes gerados, após estes passarem pelo processo de mutação e depois de otimização. O processo então se repete por *t* gerações, convergindo até a solução ótima para o problema de MVC, como representado no algoritmo 6.

Algoritmo 6: Algoritmo Genético Híbrido (HGA) (G, pop, ger)

Entrada: Um grafo $G = \{V, E\}$, o tamanho da população de cromossomos P e o número de gerações que serão transcorridas *t*.

Saída: O cromossomo P_i, sendo o mais apto da população após *t* gerações

1 Gerar *P* cromossomos aleatórios certificados como VC de G, usando o algoritmo 3.

2 Através da FF, que verifica quantos vértices ativos tem cada cromossomo, ordenar a lista da população.

3 Para cada geração indicada (ger):

Selecionar 50% primeiros cromossomos da geração anterior (mais aptos)

Inserir em P_i

Para cada cromossomo inserido em P_i, começando do primeiro e indo até o penúltimo (iteração por pares):

Resultado 1 = HVX (G, P_n, P_{n+1})

Resultado 2 = Mutação (G, Resultado 1)

Resultado 3 = LOT (G, Resultado 2)

Inserir 'Resultado 3' em P_i

Aplicar a FF em P_i

4 Retornar o primeiro cromossomo de P_i

UĞURLU, no seu artigo "*A new hybrid genetic algorithm for vertex cover problem"* de 2013, utiliza populações de até 100 cromossomos iterando através de 10 gerações, argumentando que devido a abordagem de otimização local e a heurística de cruzamento e mutação o HGA converge rápido para a solução global. Kotecha e Gambhava (2003) descrevem a importância da mutação e do LOT para a solução do problema, demonstrando que em diversos grafos a solução do MVC é encontrada mais vezes com tais métodos, quando comparado com abordagens que não os aplicam.

6. DESEMPENHO DO ALGORITMO APROXIMADO

No intuito de avaliar o desempenho do HGA no tocante a sua capacidade de convergência para o ótimo global em tempo factível, foram consideradas as 20 primeiras instâncias do *Benchmarks with Hidden Optimum Solutions for Graph Problems* (BHOSLIB) apresentadas na competição de problemas booleanos de satisfação (SAT) de 2004, instâncias essas traduzidas de SAT gerados aleatoriamente na área de transição de

fase de acordo com o modelo RB, proposto por Xu e Li (2000), e que foram provadas de serem difíceis tanto na teoria quanto na prática (XU et al., 2007).

Ressalta-se que o benchmark BHOSLIB possui, ao total, 40 instâncias com grafos variando de 450 a 1534 vértices, além de dois grafos complementares de 4.000 vértices, e tem sido amplamente utilizado na literatura como um ponto de referência para novas heurísticas associadas ao problema de MVC, com suas instâncias disponíveis no endereço http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm (CAI, SU e SCHEN, 2010).

Como referência, foram comparados os resultados do algoritmo HGA com os do seu algoritmo heurístico de reprodução, o HVX, que converge rapidamente para soluções ótimas, ainda que usualmente locais. Como o operador de cruzamento de vértices não possui elementos evolutivos, caraterísticos de GA (evolução de Darwin) e de LOT (evolução de Lamarck), espera-se que suas soluções não sejam tão próximas do mínimo global. Contudo, pode ser considerado como ponto de referência para o desempenho do algoritmo genético híbrido, tanto no tempo de execução quanto na solução alcançada. Os resultados obtidos, na mesma máquina utilizada para verificar o desempenho do algoritmo exato, constam da tabela 3.

Tabela 3 – Desempenho dos algoritmos HGA e HVX nas 20 primeiras instâncias do benchmark BHOSLIB.

Grafo	Vértices	Arestas	MVC	HGA		HVX	
				Solução	Tempo (segundos)	Solução	Tempo (segundos)
frb30-15-1	450	17.827	420	427	431,61	431	1,41
frb30-15-2	450	17.874	420	429	393,27	432	1,41
frb30-15-3	450	17.809	420	427	409,06	429	1,41
frb30-15-4	450	17.831	420	427	292,19	430	1,31
frb30-15-5	450	17.794	420	424	341,50	429	1,34
frb35-17-1	595	27.856	560	567	691,64	569	2,89
frb35-17-2	595	27.847	560	568	717,53	570	3,09
frb35-17-3	595	27.931	560	568	585,28	572	2,97
frb35-17-4	595	27.842	560	568	580,56	571	3,28
frb35-17-5	595	28.143	560	568	756,84	570	2,94
frb40-19-1	760	41.314	720	733	770,92	738	5,97
frb40-19-2	760	41.263	720	729	1.086,56	734	6,17
frb40-19-3	760	41.095	720	730	1.200,70	731	5,61
frb40-19-4	760	41.605	720	730	1.092,08	734	5,70
frb40-19-5	760	41.619	720	729	1.126,56	732	5,70
frb45-21-1	945	59.186	900	912	1.970,86	917	10,23
frb45-21-2	945	58.624	900	911	1.531,69	914	10,47
frb45-21-3	945	58.245	900	914	1.633,78	916	10,77
frb45-21-4	945	58.549	900	910	1.980,36	912	10,80
frb45-21-5	945	58.579	900	914	1.806,23	915	10,42

Dada a complexidade dos grafos do *benchmark* BHOSLIB, não foi possível estabelecer valores elevados para os parâmetros de população *P* e geração *t* do algoritmo HGA, o que impediu a convergência das soluções para o ótimo global. Em verdade, mesmo para valores reduzidos considerados ($P = 20$ e $t = 5$) o algoritmo levou uma quantidade significativa de tempo para convergir para a solução, especialmente quando comparado ao algoritmo heurístico HVX, que alcançou resultados similares em tempo substancialmente menores.

Contudo, para grafos de menor complexidade, como os registrados por Ashay Dharwadker em "*The Vertex Cover Algorithm*" (2011) e usados para validar o algoritmo exato proposto neste trabalho, o HGA conseguiu convergir rapidamente para o mínimo global, enquanto o HVX, por sua vez, apresentou resultados mais dispersos, por vezes se assentando em ótimos locais em vez de convergir para o MVC do grafo. Dessa forma, à medida que a complexidade do grafo aumenta, o HGA passa a ser uma solução menos factível para o problema de MVC do que outras abordagens heurísticas, como o HVX.

Kumar (2009), obteve resultados similares em sua pesquisa, em que foram comparados vários algoritmos como abordagens para o problema de cobertura mínima de vértices, a saber: *Branch and Bound*, Algoritmo Aproximado, Algoritmo Guloso, Algoritmo Genético, *Primal-Dual*, e Algoritmo de Alom. No estudo, o autor registra as seguintes observações:

- *Branch and Bound*: É um algoritmo exato que encontra garantidamente a cobertura mínima do vértice. Sua complexidade cresce exponencialmente rápido com o tamanho do problema;
- Algoritmo Aproximado: Fornece diferente soluções, porém todas elas próximas da ótima com, no máximo, o dobro do seu tamanho.
- Algoritmo Guloso: É fácil encontrar situações em que esse algoritmo falha em produzir uma solução ótima.
- Algoritmo Genético: Fornece melhores resultados quando combinado com técnicas de otimização local. Para problemas maiores, o crescimento do número de avaliações requeridas pelo algoritmo genético se torna mais rápido.
- *Primal-Dual*: O algoritmo termina assim que a solução primária é viável. A solução dual final é usada como um limite inferior para o valor da solução ótima por meio da dualidade fraca.

- Algoritmo de Alom: Busca sempre retornar a solução ótima do grafo. A complexidade é a mesma do Algoritmo Aproximado. Para grafos maiores esse algoritmo pode não conseguir retornar uma solução ótima.

7. APLICAÇÕES PRÁTICAS

O problema de MVC atraiu pesquisadores e profissionais devido ao NP-completude e também porque muitos problemas difíceis da vida real podem ser formulados como instâncias do MVC. As aplicações práticas ocorrem em várias áreas, como por exemplo: comunicações, engenharia civil e elétrica (KUMAR, 2009), rede de segurança, escalonamento e projeto VLSI. O MVC também está intimamente relacionado ao problema de encontrar um clique máximo. Isso tem uma gama de aplicações em bioinformática e biologia, como a identificação de sequências de proteínas relacionadas e para encontrar polimorfismo genético no genoma humano (WAGNER et al, 2017).

Para ilustrar uma aplicação do MVC, considere o problema de instalar proteções em um museu onde os corredores convergem nas quinas. A tarefa é colocar um número mínimo de guardas para que haja pelo menos um guarda no final de cada corredor (KUMAR, 2009). Um outro exemplo é o caso de policiamento de ruas de um bairro, onde determinadas esquinas deverão receber os postos de vigilância de modo que os principais quarteirões recebam um posto em pelo menos uma de suas esquinas.

Lista-se outras aplicações para o problema de encontrar a cobertura mínima de vértices (GUSEV, 2020):

i. Cobertura de vértices em redes de transporte: Seja o grafo não direcionado uma rede de transporte. Cada vértice é uma encruzilhada, uma aresta é uma estrada. Se câmeras de vigilância forem implantadas nos vértices da MVC, então cada trecho da estrada será monitorado por uma câmera e os custos de aquisição de câmeras serão minimizados. As câmeras de vigilância devem fornecer cobertura total da rede de transporte. Se as câmeras existentes capturarem inteiramente a rede de transporte, não haverá mais necessidade de alocação de orçamento para a compra de novas câmeras.

ii. Cobertura de vértices em uma sociedade: Seja cada vértice do grafo uma pessoa. Se houver um conflito entre duas pessoas, então existe uma aresta entre os vértices que

representam essas pessoas. Denote que no subconjunto inverso ao MVC, não há conflitos diretos entre pessoas no conjunto, e a dimensionalidade deste conjunto é máxima.

iii. Vertex Cover em processos de negócios: Seja cada vértice do grafo um estado de um certo processo. Se a transição de um estado para outro for possível, então existe uma aresta entre os vértices correspondentes. Algum trabalho é executado durante a transição de um estado do processo para outro. Podem surgir erros durante a execução do trabalho. As verificações são necessárias para controlar os erros. Se tais verificações forem feitas em vértices da MVC, o número de verificações será minimizado. Além disso, se um processo estiver em dois estados, pelo menos uma verificação será executada.

iv. Cobertura de vértices em redes de computadores: Seja o grafo G uma rede de servidores. Cada vértice do grafo G é um servidor. Há uma borda entre dois vértices se seus servidores correspondentes estiverem interconectados. É demonstrado que a propagação de um vírus em uma rede depende da topologia do grafo G. O manuseio oportuno dos servidores na cobertura mínima do vértice pode mitigar ou impedir a propagação do vírus.

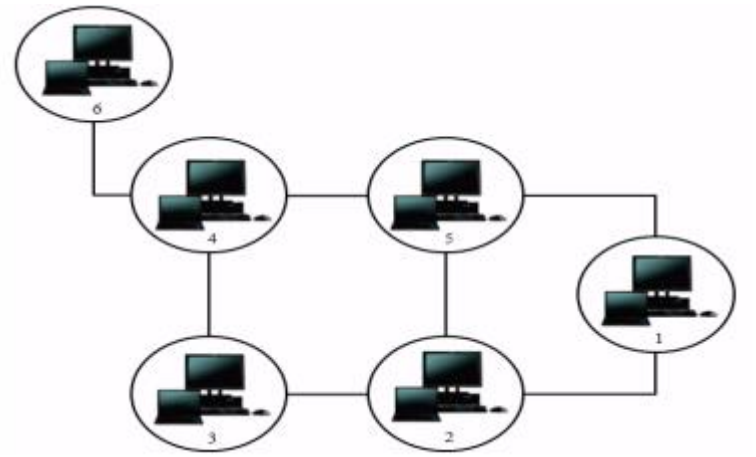


Figura 3 -O conjunto {2,4,5} é um mínimo *vertex cover* (MVC) na rede de computadores

Deve-se notar que, por algumas razões práticas, a MVC pode ser inferior em suas propriedades à VC não mínima. A MVC pode mudar se algumas arestas forem adicionadas ou removidas do gráfico. É por isso que o foco ao considerar o modelo matemático no qual a topologia do grafo será variável deve ser colocado em uma VC que

não seja mínima. Essa cobertura, não mínima, muito provavelmente permaneceria uma cobertura se a topologia dos grafos fosse transformada. Em redes de transporte, entretanto, estradas principais surgem ou desaparecem raramente, então a tomada de decisão baseada na centralidade do vértice em uma MVC é aceitável (GUSEV, 2020).

Com o intuito de aplicar o MVC a uma situação do mundo real, o presente estudo aborda um problema encontrado no policiamento de ruas, que é a necessidade de ações para reduzir a criminalidade em regiões ditas como críticas. Em reportagem do site Perimetral Segurança (2021), a polícia Civil de Belo Horizonte mostra as principais rotas dos criminosos, apontando que o combate aos assaltos deve se concentrar em nove zonas quentes espalhadas por seis regiões da cidade.

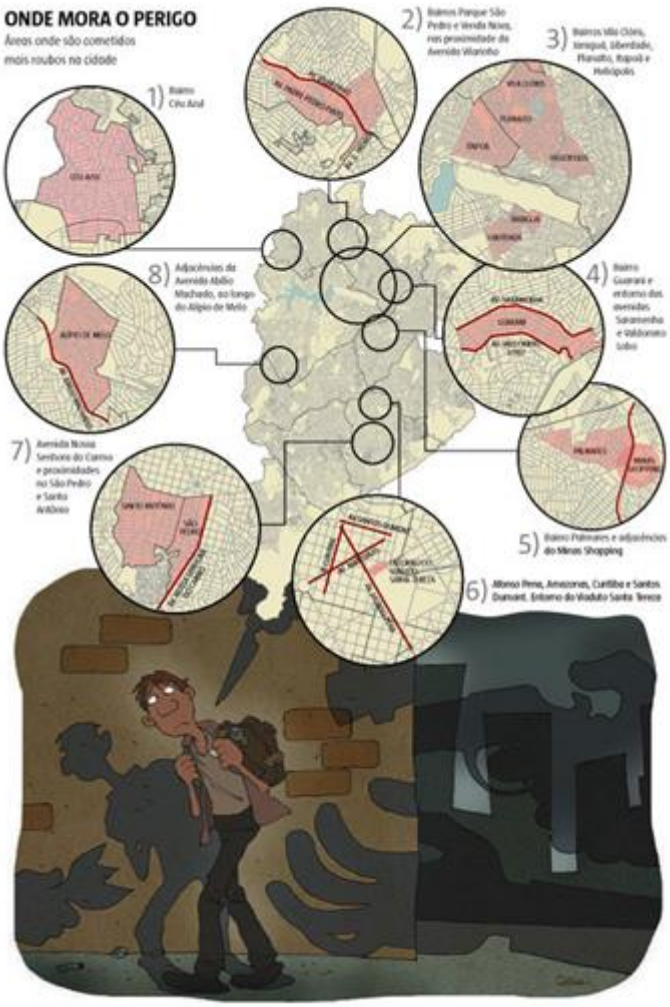


Figura 4 -Polícia mostra as rotas dos criminosos em Belo Horizonte
Fonte: *http://perimetralseguranca.com.br/blog/policia-mostra-rotas-dos-criminosos-em-belo-horizonte/*

Para a aplicação prática do MVC, foi escolhido o bairro Guarani, considerado o um dos mais violento dentre os 14 bairros considerados mais perigosos pelos dados

disponibilizados pela polícia civil de Belo Horizonte. Recorrentemente são registradas ocorrências de assaltos, roubos, agressões e outros crimes no local.

Diante da necessidade de ações para reduzir a criminalidade nesse bairro, i.e., torná-lo mais seguro, propõe-se a instalação de câmeras de vigilância em pontos específicos. O objetivo é utilizar o MVC para identificar o mínimo de pontos nos quais as câmeras de vigilância devem ser instaladas, de modo que todas as ruas do bairro sejam cobertas.

Segundo estatística do IBGE, obtida no Censo de 2010, a população do bairro Guarani em Belo Horizonte (MG) é de aproximadamente 7.718 habitantes. Considerando o contorno fornecido no *Google Maps*, pode-se atribuir a esse bairro 145 segmentos de rua compreendidos entre 94 esquinas e cruzamentos. Para filmar todas as ruas é necessário identificar quais os pontos em que é necessária a instalação de câmeras de vigilância de forma que todas os seus segmentos sejam cobertos. O ideal é que seja o menor número de pontos possível, não só por questões orçamentárias, mas também para facilitar o monitoramento. No caso, as esquinas e cruzamentos são considerados os vértices e os segmentos de ruas entre eles são as arestas do grafo, como ilustrado na figura 5. Assume-se ainda que a extensão de qualquer aresta é passível de ser coberta pelo alcance das câmeras.

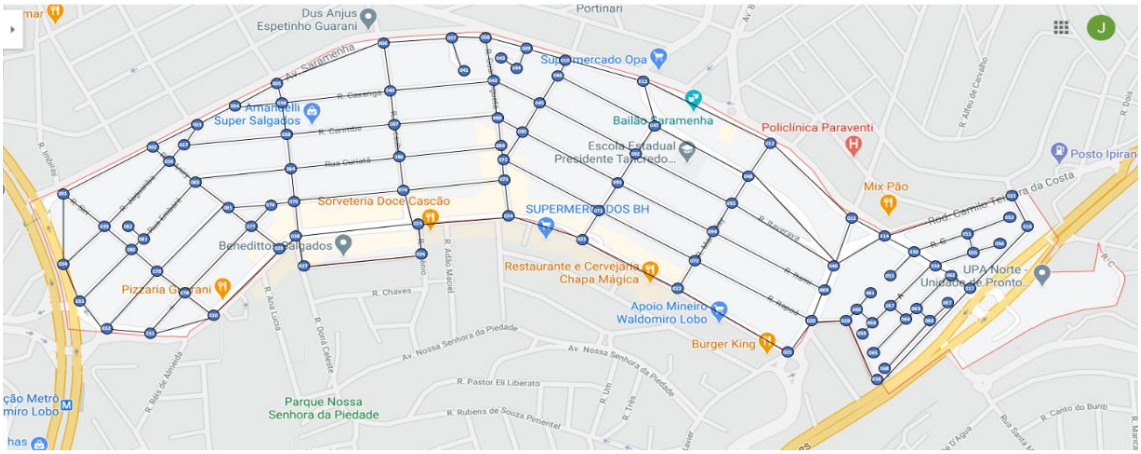


Figura 5 – Grafo construído sobre o bairro do Guarani, em Belo Horizonte-MG, com os vértices representando suas esquinas e cruzamentos e as arestas os segmentos de ruas compreendidos entre eles
Fonte: elaborado pelos autores

Como se trata de mais de 94 vértices, não seria possível, com o algoritmo exato proposto no presente estudo, encontrar a solução em tempo factível, já que este se mostra inviável a partir de poucas dezenas de nós (vértices). O algoritmo levaria dias ou até semanas para encontrar a solução exata, o que inviabilizaria futuras revisões do projeto. Por esse motivo foram utilizados os algoritmos aproximados aqui abordados.

Usando um espaço amostral de 100 soluções e um nível de confiança de 95% considerando uma distribuição normal, obteve-se, com o algoritmo heurístico HVX, uma média de 54 vértices em um intervalo de confiança de 53,87 e 54,33. O algoritmo meta-heurístico HGA, por sua vez, retornou rapidamente, nas mesmas condições e com os parâmetros população $P = 50$ e geração $t = 10$, uma média de 50 vértices em um intervalo de confiança de 50,11 e 50,27.

Observa-se que o intervalo de confiança para o HGA foi consideravelmente estreito, indicando uma forte convergência para a solução de 50 vértices. Constatou-se ainda, aumentando os parâmetros P e t para 100 e 20, respectivamente, que o algoritmo sempre converge para essa solução, possivelmente indicando a obtenção do ótimo global, isto é, a solução exata para o problema de cobertura de vértices mínima no bairro do Guarani.

Dessa forma, pode-se concluir que o algoritmo genético híbrido HGA se mostrou a melhor abordagem para o problema, retornando rapidamente um ótimo - possivelmente global - de 50 pontos para instalação de câmeras de vigilância de modo a cobrir todas as suas ruas. Observa-se, ainda, que o MVC encontrado representa aproximadamente metade de suas esquinas e cruzamentos. A figura 6 ilustra os pontos (em verde) a serem instaladas as câmeras, de acordo com a solução encontrada pelo algoritmo HGA.

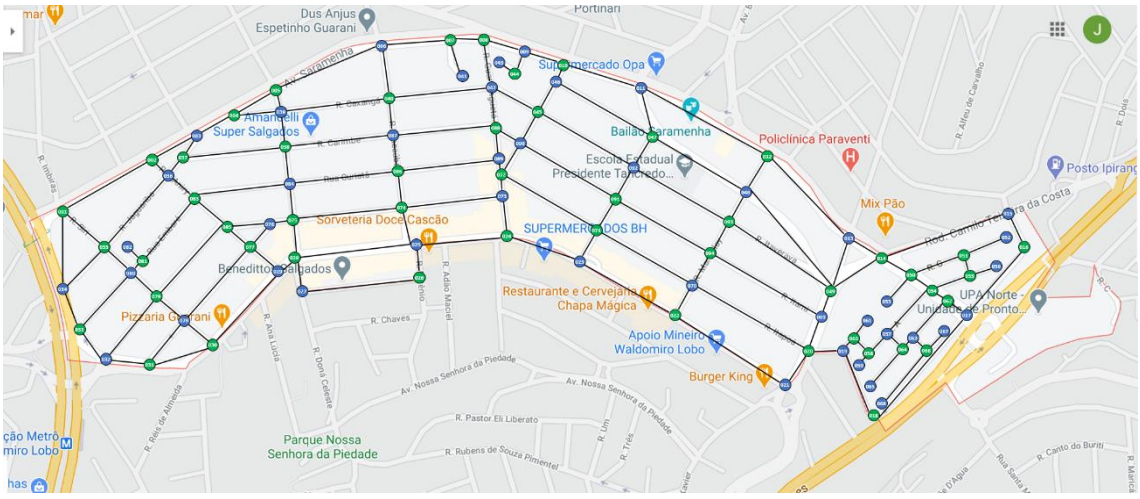


Figura 6 – Grafo construído sobre o bairro do Guarani, em Belo Horizonte-MG, com os vértices representando suas esquinas e cruzamentos e as arestas os segmentos de ruas compreendidos entre eles. Os 50 vértices em verde representam os pontos a serem instaladas câmeras de vigilância de modo a cobrir todas as ruas do bairro.
Fonte: elaborado pelos autores

8. CONCLUSÃO

Diante dos fatos analisados, verifica-se que o algoritmo exato implementado para o problema de cobertura mínima de vértices fornece soluções ótimas. Contudo, sua performance vai depender do tipo de grafo avaliado e do seu número de vértices, tornando-se infactível com poucas dezenas de nós. Visando contornar esse problema foram empregados algoritmos aproximados usando heurísticas (algoritmo HVX) e meta-heurísticas (algoritmo HGA), que se mostraram efetivos na obtenção de soluções ótimas ou quase ótimas em tempo polinomial.

O algoritmo HVX converge rapidamente para uma solução quase ótima, tendendo, no entanto, a se assentar em mínimos locais. O algoritmo HGA, por sua vez, consegue convergir para o mínimo global, com o desempenho dependendo dos parâmetros de quantidade da população e número de gerações, bem como da complexidade do grafo avaliado. Para grafos com grande quantidade de vértices, o HGA leva uma quantidade de tempo substancialmente maior do que o HVX para encontrar uma solução apenas um pouco melhor. Dessa forma, o algoritmo HVX se torna mais factível que o HGA à medida que a complexidade do grafo avaliado aumenta.

Para aplicações de menor complexidade, como o caso do bairro do Guarani, em Belo Horizonte - Minas Gerais, com suas esquinas e cruzamentos representados por vértices e os segmentos de rua entre eles por arestas, referentes a um grafo G = (94, 145), o HGA se mostrou a melhor abordagem, já que rapidamente retornou a solução – possivelmente - ótima para o problema, ao passo que o algoritmo exato se mostrou infactível e o algoritmo HVX convergiu - ainda que de forma rápida - para ótimos locais.

BIBLIOGRAFIA

ABDEL-BASSET, Mohamed; ABDEL-FATAH, Laila; SANGAIAH, Arun Kumar. Metaheuristic algorithms: A comprehensive review. Computational intelligence for multimedia big data on the cloud with engineering applications, p. 185-231, 2018.

BHASIN, Harsh; AHUJA, Geetanjli. Harnessing genetic algorithm for vertex cover problem. International Journal on Computer Science and Engineering, v. 4, n. 2, p. 218, 2012.

CHAKRABORTY, Udit; KONAR, Debanjan; CHAKRABORTY, Chandralika. A GA based Approach to Find Minimal Vertex Cover. 2014.

DHARWADKER, Ashay. The vertex cover algorithm. Institute of Mathematics, 2011. Disponível em: < http://www.dharwadker.org/vertex_cover/>. Acesso: 24 de março de 2021.

ERDŐS, P.; RÉNYI, A. On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci, v. 5, n. 1, p. 17-60, 1960.

EVANS, Isaac K. Evolutionary algorithms for vertex cover. In: International Conference on Evolutionary Programming. Springer, Berlin, Heidelberg, 1998. p. 377-386.

GLOVER, F. W.; KOCHENBERGER, G. A., eds. Handbook of metaheuristics. Vol. 57. Springer Science & Business Media, 2006.

GUSEV, Vasily V. The Vertex Cover game: Application to transport networks. National Research University Higher School of Economics. 2020.

HAN, Keun-Hee; KIM, Chan-Soo. Applying Genetic Algorithm to the Minimum Vertex Cover Problem. The KIPS Transactions: PartB, v. 15, n. 6, p. 609-612, 2008.

KARP, Richard M. Reducibility among combinatorial problems. In: Complexity of computer computations. Springer, Boston, MA, 1972. p. 85-103.

KOTECHA, Ketan; GAMBHAVA, Nilesh. A Hybrid Genetic Algorithm for Minimum Vertex Cover Problem. In: IICAI. 2003. p. 904-913.

KUMAR, K. V. R.; MAAROJU, Narendhar; GARG, Deepak. Complete Algorithms on Minimum Vertex-Cover. CIIT International Journal of Software Engineering and Technology, 2009.

KUMAR, K.V.R. Choosing the Efficient algorithm for Vertex Cover Problem. Computer Science and Engineering Department Thapar University. June, 2009.

LEBZA, Rezzag; AYMEN, Mohamed. Meta-heuristic based approach for Minimum Vertex Cover Problem. 2019. Tese de Doutorado. UNIVERSITE MOHAMED BOUDIAF-M'SILA FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE DEPARTEMENT D'INFORMATIQUE-Option: IDO.

LEWIS, Harry R. Computers and intractability. A guide to the theory of NP-completeness. 1983.

OPENDSA. Chapter 28: Limits to Computing. 28.17. Reduction of Independent Set to Vertex Cover. 2017.

PERIANNAN, Muthubharathi. An ant-based algorithm for the minimum vertex cover. 2007. Dissertação de Mestrado. Universidade do Estado da Pensilvânia.

TAYLOR, David. NP – Completeness: A Few Simple Reduction: NP-Complete Reductions: Clique, Independent Set, Vertex Cover, and Dominating Set. Algorithms with Attitude. 2020.

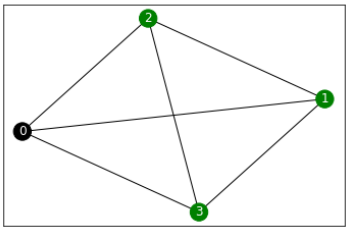
UĞURLU O. A NEW HYBRID GENETIC ALGORITHM FOR VERTEX COVER PROBLEM. Anadolu Üniversitesi Bilim Ve Teknoloji Dergisi A-Uygulamalı Bilimler ve Mühendislik. 2013 Jun 24;14(3):277-82.

WAGNER, Markus; FRIEDERICK, Tobias; LINDAURER, Marius. Improving local search in a minimum vertex cover solver for classes of networks. 2017.

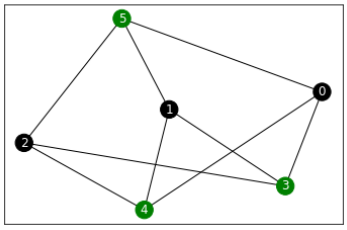
ANEXOS

ANEXO A - GRAFOS CONHECIDOS PARA VALIDAÇÃO DO ALGORITMO EXATO

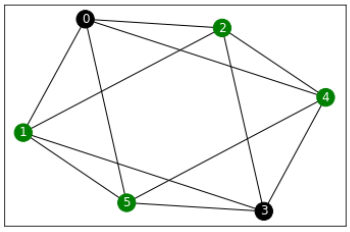
Tetraedro - MVC: 1, 2, 3



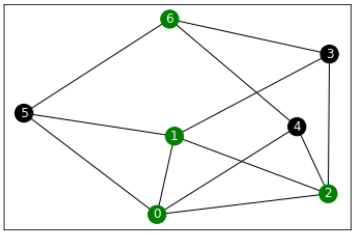
Grafo bipartido de Kuratowski K3,3 - MVC: 3, 4, 5



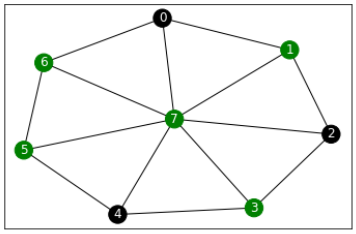
Octaedro - MVC: 1, 2, 4, 5



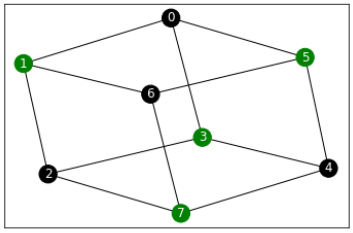
Grafo de Bondy-Murty G1 - MVC: 0, 1, 2, 6



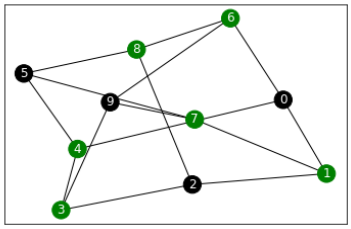
Grafo de roda W8 - MVC: 1, 3, 5, 6, 7



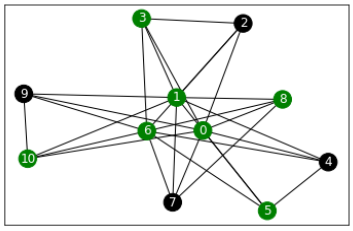
Cubo - MVC: 1, 3, 5, 7



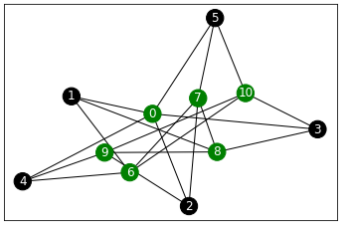
Grafo de Petersen - MVC: 1, 3, 4, 6, 7, 8



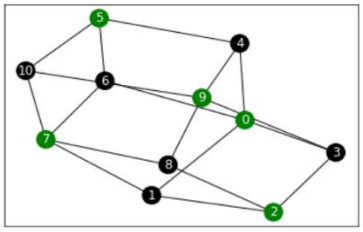
Grafo de Bondy-Murty G2 - MVC: 0, 1, 3, 5, 6, 8, 10



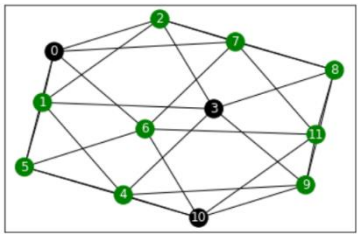
Grafo de Grötzsch - MVC: 0, 6, 7, 8, 9, 10



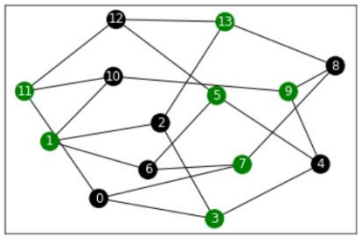
Grafo de Herschel - MVC: 0, 2, 5, 7, 9



Icosaedro - MVC: 1, 2, 4, 5, 6, 7, 8, 9, 11

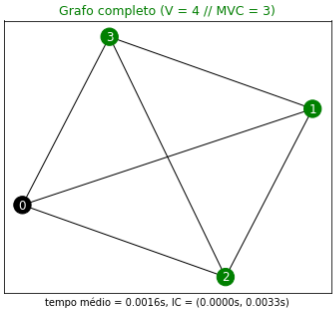


Bondy-Murty G3 - MVC: 1, 3, 5, 7, 9, 11, 13

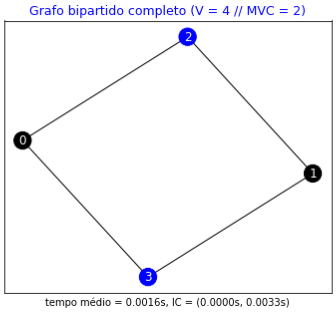


ANEXO B – GRAFOS UTILIZADOS PARA AVALIAR O TEMPO DE EXECUÇÃO DO ALGORITMO EXATO

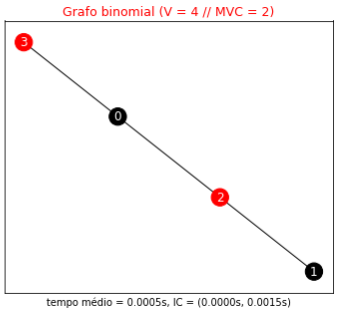
Verde - Grafo Completo; Azul - Grafo Bipartido Completo; Vermelho - Grafo Binomial



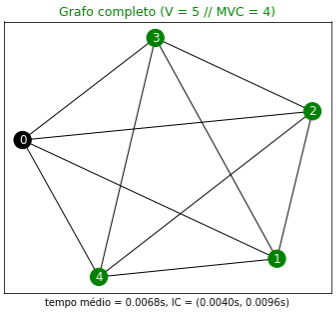
tempo médio = 0.0016s, IC = (0.0000s, 0.0033s)



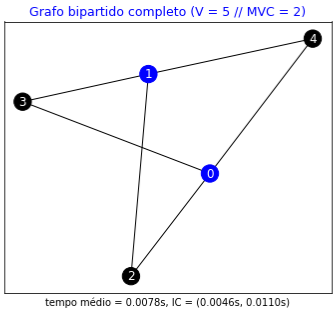
tempo médio = 0.0016s, IC = (0.0000s, 0.0033s)



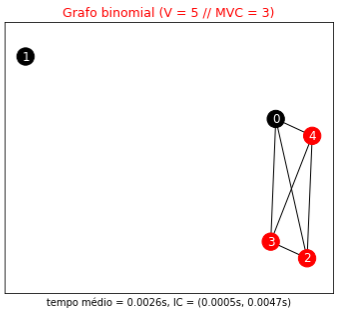
tempo médio = 0.0005s, IC = (0.0000s, 0.0015s)



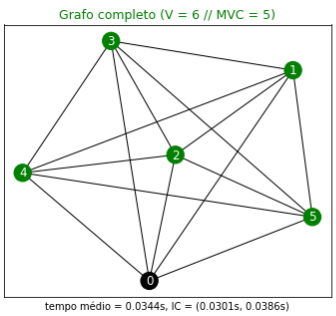
tempo médio = 0.0068s, IC = (0.0040s, 0.0096s)



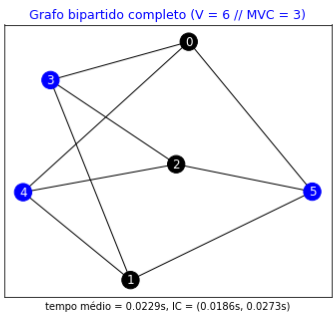
tempo médio = 0.0078s, IC = (0.0046s, 0.0110s)



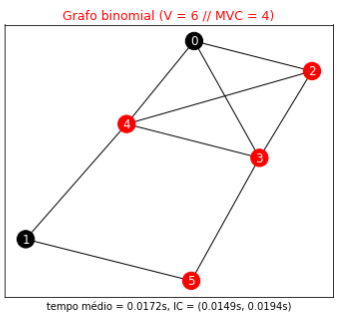
tempo médio = 0.0026s, IC = (0.0005s, 0.0047s)



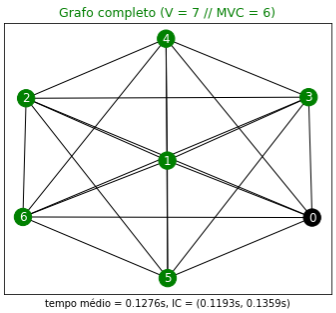
tempo médio = 0.0344s, IC = (0.0301s, 0.0386s)



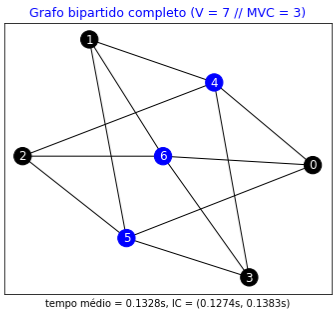
tempo médio = 0.0229s, IC = (0.0186s, 0.0273s)



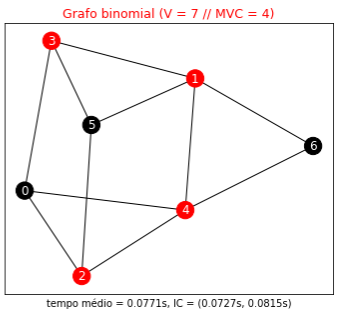
tempo médio = 0.0172s, IC = (0.0149s, 0.0194s)



tempo médio = 0.1276s, IC = (0.1193s, 0.1359s)

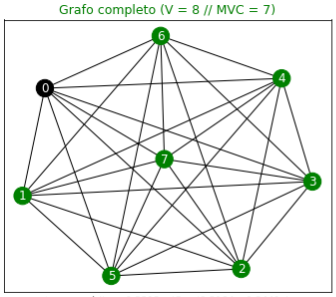


tempo médio = 0.1328s, IC = (0.1274s, 0.1383s)



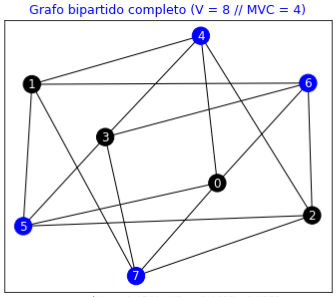
tempo médio = 0.0771s, IC = (0.0727s, 0.0815s)

Grafo completo (V = 8 // MVC = 7)



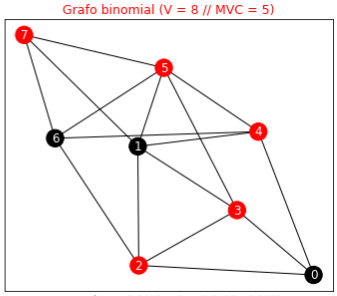
tempo médio = 0.3312s, IC = (0.3184s, 0.3441s)

Grafo bipartido completo (V = 8 // MVC = 4)



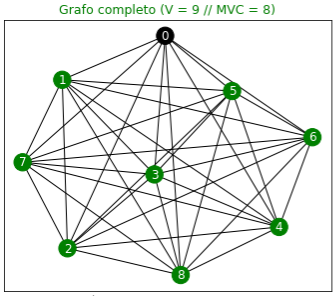
tempo médio = 0.4740s, IC = (0.4608s, 0.4871s)

Grafo binomial (V = 8 // MVC = 5)



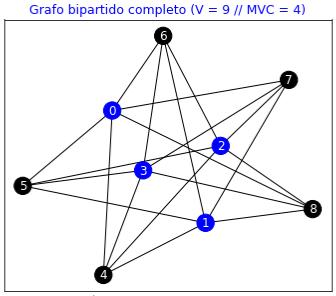
tempo médio = 0.2495s, IC = (0.2418s, 0.2572s)

Grafo completo (V = 9 // MVC = 8)



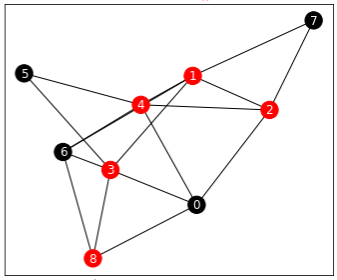
tempo médio = 0.9344s, IC = (0.8762s, 0.9926s)

Grafo bipartido completo (V = 9 // MVC = 4)



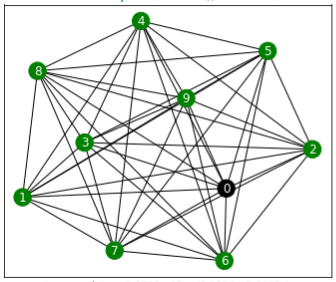
tempo médio = 2.4854s, IC = (2.4356s, 2.5352s)

Grafo binomial (V = 9 // MVC = 5)



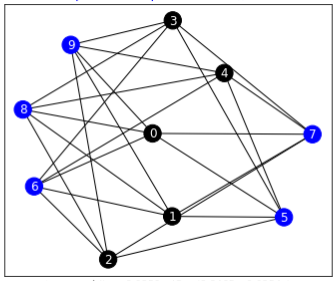
tempo médio = 0.9026s, IC = (0.8701s, 0.9351s)

Grafo completo (V = 10 // MVC = 9)



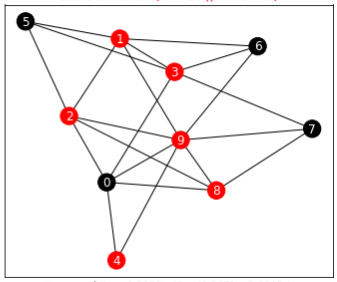
tempo médio = 2.1708s, IC = (2.1384s, 2.2032s)

Grafo bipartido completo (V = 10 // MVC = 5)



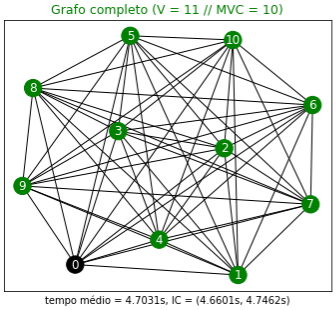
tempo médio = 8.9359s, IC = (8.8465s, 9.0254s)

Grafo binomial (V = 10 // MVC = 6)



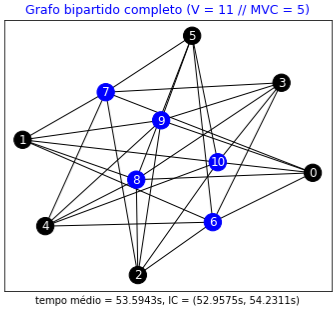
tempo médio = 3.3958s, IC = (3.3471s, 3.4446s)

Grafo completo (V = 11 // MVC = 10)



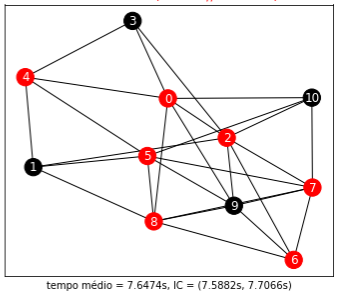
tempo médio = 4.7031s, IC = (4.6601s, 4.7462s)

Grafo bipartido completo (V = 11 // MVC = 5)



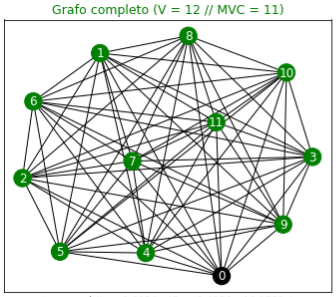
tempo médio = 53.5943s, IC = (52.9575s, 54.2311s)

Grafo binomial (V = 11 // MVC = 7)



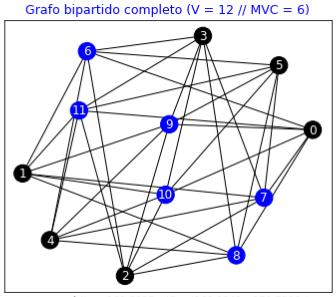
tempo médio = 7.6474s, IC = (7.5882s, 7.7066s)

Grafo completo (V = 12 // MVC = 11)



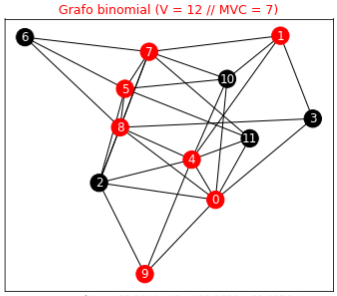
tempo médio = 9.8354s, IC = (9.4956s, 10.1752s)

Grafo bipartido completo (V = 12 // MVC = 6)

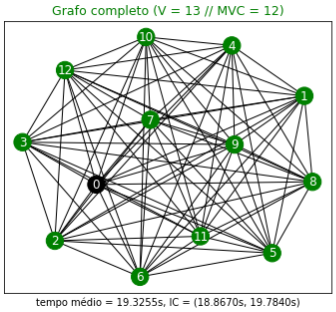


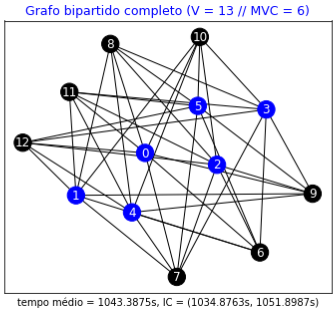
tempo médio = 169.7927s, IC = (168.8348s, 170.7506s)

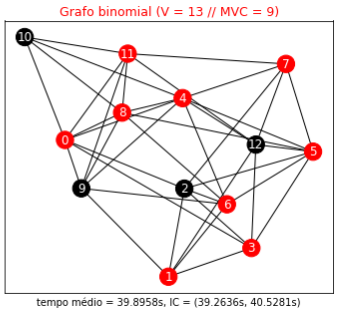
Grafo binomial (V = 12 // MVC = 7)



tempo médio = 33.2844s, IC = (33.1332s, 33.4356s)


Grafo completo (V = 13 // MVC = 12)
tempo médio = 19.3255s, IC = (18.8670s, 19.7840s)


Grafo bipartido completo (V = 13 // MVC = 6)
tempo médio = 1043.3875s, IC = (1034.8763s, 1051.8987s)


Grafo binomial (V = 13 // MVC = 9)
tempo médio = 39.8958s, IC = (39.2636s, 40.5281s)