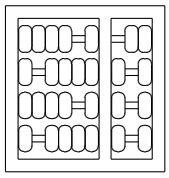


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Algoritmos para Teste de Isomorfismo de Grafos

E. R. Mattos *E. C. Xavier*

Relatório Técnico - IC-PFG-16-08 - Projeto Final de Graduação

December - 2016 - Dezembro

Algoritmos para Teste de Isomorfismo de Grafos

Erick Ricardo Mattos* Eduardo Candido Xavier†

Resumo

O objetivo deste trabalho é estudar a implementação de algoritmos que resolvam o problema de Isomorfismo em Grafos (em inglês *Graph Isomorphism* ou apenas GI). Não é conhecida uma solução polinomial para o problema geral, aquele que não assume alguma propriedade especial para o grafo, bem como não foi provado que este problema é NP-Difícil e, assim, a complexidade deste problema ainda está em aberto.

Foram estudados diferentes métodos para a solução do problema, sendo escolhido para implementação um método proposto por Weisfeiler e Lehman [24], conhecido como k-dim WL, que é capaz de resolver o problema em tempo linear no número de vértices, mas que, no pior caso, ainda é exponencial, tanto em espaço como em tempo.

O método implementado obteve um grande *speedup* em relação ao método trivial, que é tentar todas as possíveis permutações de vértices, quando executados com grafos aleatórios.

Apesar de ser um método com complexidade exponencial, a implementação conseguiu resolver muito casos de teste em baixo tempo computacional e com rapidez, mesmo para grafos com muito vértices. O pior caso do método nos testes foi observado em grafos regulares, como previsto pela análise teórica do algoritmo.

***Palavras chave*:** Algoritmo; Grafo; Isomorfismo; Combinatória;

Introdução

O problema Isomorfismo em Grafos é um problema em aberto na computação, ou seja, não foi encontrado algum algoritmo com complexidade polinomial no tamanho do grafo (aqui considerado como a soma do número de vértices e arestas do grafo) e também não foi provado pertencer as classes NP-Difícil, e consequentemente, a NP-Completo.

Neste projeto foi considerado a implementação de métodos para a resolução deste problema, em especial o método proposto por Weisfeiler e Lehman no fim da década de 60[24]. Este método é capaz de resolver o problema, para algumas classes de grafos, em tempo linear no número de vértices, contudo, a complexidade no pior caso, no caso grafos regulares, ainda é exponencial: tanto em tempo como em espaço.

A implementação do método foi sujeita a testes utilizando três diferentes bancos de grafos: a primeira continha grafos completamente aleatórios, a segunda continha grafos

*Graduando, Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP.
†Professor Associado, Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP.

regulares aleatórios e a ultima base de dados foi formada a partir de bancos de grafos criados para teste de isomorfismo em grafos disponíveis na *Internet*.

Para a construção das bases de grafos aleatórios foi utilizado um método para geração de grafos a partir de uma sequência de graus para os vértices propostas em [5]. Com este método foi possível gerar grafos completamente aleatórios, tanto os regulares como os não. As bases de teste para isomorfismo foram localizadas utilizando os buscadores padrões de páginas Web.

Este relatório está organizado do seguinte modo: a seção 1 contem uma explicação detalhado do problema abordado, bem como uma explicação das notações utilizadas neste projeto. A seção 2 contém uma descrição do processo de busca dos métodos para a solução do problema abordado, assim como a escolha do método implementado, k-dim WL. A seção 3 contém uma descrição dos algoritmos utilizados no desenvolvimento do projeto, tanto os algoritmos para gerar grafos, como os algoritmos para verificar o isomorfismo de grafos. A seção 4 contem mais informações sobre as bases de teste e apresenta os resultados dos testes com o método escolhido. Por fim, a seção 5 contém a discussão dos resultados encontrados.

1 Descrição do problema

1.1 Base teórica

Neste projeto, um grafo G é formado por um conjunto de vértices $V(G)$ e um conjunto de arestas $E(G)$ e é representado por $G = (V, E)$. Uma notação mais compacta é representar $V(G)$ por V_G e $E(G)$ por E_G. O conjunto E_G é uma relação binária de V_G, ou seja, $E_G \subset V_G \times V_G$. A cardinalidade de um conjunto S é representada por $|S|$, assim a notação G_n representa uma grafo genérico com $|V_G| = n$.

Seja $v \in V_G$ um vértice qualquer, então:

- $d(v)$ é o conjunto de vizinhos de v, ou seja, $d(v) = \{x \in V_G : (v, x) \in E_G\}$;
- $e(v)$ é o conjunto de não-vizinhos de v, ou seja, $e(v) = \{x \in V_G : (v, x) \notin E_G\}$;
- A valência ou cardinalidade de v é equivalente a $|d(v)|$ e pode ser denotada por d_v.

Para facilitar, um grafo G_n possui vértices nomeados/rotulados por números inteiros, ou seja, $V_G = \{1, 2, \ldots, n-1\}$. Deste modo, pode-se definir o conceito de matriz de adjacência Adj de um grafo G_n como $Adj \in \mathbb{B}_n$, onde $\mathbb{B}_n$ é o conjunto de matrizes binárias quadradas de ordem n, e:

$$Adj_{u,v} = \begin{cases} 1, & se\,(u, v) \in E_G \\ 0, & se\,(u, v) \notin E_G \end{cases}$$

Um grafo colorido, dado por $G = (V, E, C_1, \ldots, C_r)$, é um grafo no qual cada vértice satisfaz exatamente uma relação de cor, ou seja, para cada $v \in V_G$ pertence exatamente a um C_i com $i \in \{1, \ldots, r\}$, ou seja, $v \in C_i$ e $\forall j \in \{1, \ldots, r\} \setminus \{i\}\ v \notin C_j$.

Uma tupla é uma sequência ordenada de elementos homogêneos, ordenada no sentido de que existe uma ordem entre os elementos e não que os elementos estejam em alguma ordenação crescente ou decrescente. De modo similar, uma k-tupla é uma tupla com k

elementos e é representada por $(t_1, t_2, \ldots, t_k)$. Como existe uma ordem nos elementos, a 2-tupla de inteiros $(1, 3)$ é diferente da tupla $(3, 1)$.

Neste projeto são considerados apenas grafos não direcionados: um grafo G é não direcionado se $\forall\,(u, v) \in E_G \rightarrow (v, u) \in E_G$ e a aresta (u, v) e (v, u) são consideradas as mesmas, ou seja, possuem as mesmas propriedades como rótulo, custo, etc. Deste modo, a matriz de adjacência de um grafo não direcionado é simétrica.

Dois grafos G e H são isomórficos se existe uma função bijetora $\varphi : V_G \rightarrow V_H$ tal que se $(u, v) \in E_G$, então $(\varphi(u), \varphi(v)) \in E_H$. Como φ é bijetora, existe uma função também bijetora $\varphi^{-1} : V_H \rightarrow V_G$ que também preserva as arestas de H: se $(u, v) \in E_H$, então $\left(\varphi^{-1}(u), \varphi^{-1}(v)\right) \in E_G$. A notação $G \simeq H$ indica que G e H são isomórficos e, consequentemente, $G \not\simeq H$ indica que G e H não são isomórficos.

1.2 Isomorfismo de grafos

O problema que se deseja estudar é: *Dado dois grafos $G = (V_G, E_G)$ e $H = (V_H, E_H)$, decidir se G e H são isomórficos.*

A complexidade complexidade deste problema ainda está em aberto. Sabe-se que ele está na classe de complexidade NP, mas não existe prova de que este problema está na classe NP-Completo ou algum algoritmo que resolva este problema em tempo polinomial e, assim pertencendo a classe P.

Como apresentado em [20], caso este problema esteja em NP-Completo, haveria um colapso da hierarquia de complexidades, o que sugere que ele não pertença a classe NP-Difícil. Em novembro de 2015, o pesquisador László Babai apresentou um artigo [3] que apresenta uma solução para o problema em tempo *quasi-polynomial*.

Apesar de não se saber a complexidade do problema genérico, existem soluções eficientes para este problema em determinadas classes de grafos com propriedades específicas [4, 7, 13, 15, 19].

Como aplicação prática deste problema destacam-se os estudos em estruturas topológicas como na *mathematical chemistry* [23].

2 Levantamento bibliográfico

A primeira etapa deste projeto foi a busca por métodos que resolvam o problema do isomorfismo em grafos. Como o artigo [3] apresentou um solução com complexidade *quasi-polynomial* e foi escrito por um pesquisador notório na área, a partir dele foi buscado artigos que apresentassem soluções ou heurísticas para o problema.

Baseado nas referências do artigo e nas citações a ele, foi levantado uma base de outros artigos que apresentavam possíveis soluções para o problema:

- *Parallel algorithms for permutation groups and graph isomorphism* por Eugene Luks [16];
- *Hypergraph isomorphism and structural equivalence of Boolean functions* por Eugene Luks [17];

- *The Parameterized Complexity of Geometric Graph Isomorphism* por Vikraman Arvind e Gaurav Rattan [1];
- *Faster isomorphism testing of strongly regular graphs* por Daniel Spielman[22];
- *Polynomial-time Algorithm for Isomorphism of Graphs with Clique-width at most Three* por Bireswar Das, Murali Enduri e I. Vinod Reddy[8];
- *Isomorphism Testing for Graphs of Bounded Rank Width* por Martin Grohe e Pascal Schweitzer[12];
- *Graph Isomorphism Restricted by Lists* por Pavel Klavik, Dušan Knop e Peter Zeman [14];
- *Isomorphism of graphs of bounded valence can be tested in polynomial time* por Eugene Luks [15];
- *Monte-Carlo algorithms in graph isomorphism testing* por László Babai[2];
- *Practical graph isomorphism* por Brendan McKay e Adolfo Piperno[18];
- *Graph Isomorphism and the Lasserre Hierarchy* por Aaron Snook, Grant Schoenebeck e Paolo Codenotti[21];
- *Graph isomorphism problem* por Viktor Zemlyachenko, Nikolai Korneenko e Regina Tyshkevich[26];
- *The Weisfeiler-Lehman Method and Graph Isomorphism Testing* por Brendan Douglas [9];

Ao se analisar os artigos encontrados, foi possível selecionar aqueles que tratavam do problema no caso geral: muito deles abortavam o problema em grafos com uma característica particular.

No final, o artigo selecionado para implementação e teste foi [9]: este artigo descreve um método desenvolvido no final da década de 1960 e resolve o problema geral. A complexidade do algoritmo proposto é, no pior caso, exponencial, mas pode resolver para muitos grafos em tempo polinomial.

A partir do artigo selecionado, foram encontrados outras referências muito importantes: o artigo original [24], em russo, que apesar de não ser usado como base, permitiu encontrar o artigo [6] que apresenta uma descrição formal do algoritmo, além de uma análise sobre a complexidade e a prova de ser exponencial no pior caso. Este método está descrito na seção 3.1.

Após a implementação, foi necessário realizar testes e para tal atividade foi necessário a busca por *datasets* que exploravam o problema de isomorfismos e apresentassem instâncias para teste. Foi encontrado dois bancos com grafos de teste: [11], que apresentou instâncias complexas, mas que não foi explicado como elas foram geradas/obtidas, e [10] que apresentou um *dataset* grande com diversos tipos de instâncias para teste, além do algoritmo que gera tais exemplos de teste.

Devido as limitações destes *datasets* (o primeiro é relativamente pequeno, apenas 40 pares de instâncias, e o segundo apresenta apenas instâncias isomórficas), foi necessário utilizar um algoritmo que gere instâncias de grafos. Após uma busca, foi escolhido implementar o algoritmo proposto por [5] para gerar as instâncias. Este algoritmo está descrito na seção 3.2.

3 Descrição dos algoritmos

Nesta seção, é apresentado uma descrição dos algoritmos implementados tanto para testar isomorfismo, como para gerar instâncias para o teste do método implementado.

3.1 Resolvendo o problema de Isomorfismo em grafos

Como já apresentado, o método para resolver o problema de isomorfismo em grafos escolhido foi, primeiramente, apresentado em [24], mas como esta referência está em russo, o algoritmo implementado foi baseado na descrição dada por [6].

O método escolhido é conhecido pela comunidade por k-dim WL, onde o k assume um valor inteiro. Assim como apresentado em [6], para um determinado grafo $G = (V, E)$, com $n = |V_G|$, e um valor de $k \in \mathbb{N}$, a complexidade do método é dada por $O\left(k^2 n^{k+1} \log n\right)$. Deste modo, caso k seja $O(n)$ este algoritmo será exponencial: este caso, o pior caso, ocorre em grafos super-regulares.

3.1.1 O método k-dim WL

Nesta seção é apresentado o método k-dimensional Weisfeiler-Lehman, ou como também é conhecido k-dim WL, em dois formato: o primeiro é a versão com $k = 1$, ou 1-dim WL, e em seguida a versão para $k > 1$.

O método 1-dim WL ou refinamento de vértice, como também é conhecido, necessita de um grafo colorido previamente $G = (V, E, C_1, \ldots, C_r)$. Agora, seja $W^0 : V_G \to \{1, \ldots, n\}$, com $n = |V_G|$, uma função dada por $W^0 : v \longmapsto i \iff v \in C_i$. Deste modo, é possível definir para uma determinada iteração $t + 1$ o valor de W^{t+1}, que representa a cor de um vértice na iteração $t + 1$, um refinamento de W^t dado por: Seja v um vértice, então:

$$W^{t+1}(v) = \langle W^t(v), y_1, z_1, \ldots, y_n, z_n \rangle$$

Com y_i o número de vértices cujo valor de W^t era igual a i e que também são adjacentes a v, enquanto que z_i é a quantidade de vértices cujo valor de W^t era igual a i e que não são adjacentes a v. Assim, dois vértices estão em uma mesma nova classe de cor se, e somente se, eles estavam na mesma classe de cor antiga e são adjacentes a, exatamente, o mesmo número de vértices em cada categoria de cor antiga.

O refinamento continua até que para algum l tem-se $\forall v \in V_G\ W^{l+1}(v) = W^l(v)$, e neste caso, é escrito $\overline{\mathrm{W}} = W^l$ e $\overline{\mathrm{W}}(G)$ é dito ser um refinamento estável de W^0 para G.

Já o método k-dim WL com $k > 1$ é um pouco mais complexo. Seja G um grafo colorido e V_G^k o conjunto com todas as possíveis tupla de tamanho k formadas com elementos de V_G: este conjunto deve ter uma relação de cor também, ou seja, cada tupla, observe que existem

$|V_G|^k$ tuplas, deve ter exatamente uma classe de cor T_i. Assim, a ordem inicial de W_k^0 é dado pela cor da tuplas: $W_k^0 : V_G^k \to \{1, \ldots, n^k\}$ mapeado por $W_k^0 : u \longmapsto i \iff u \in T_i$.

Também é necessário definir a operação *shift* que para uma determinada função $f : V_G^k \to \{1, \ldots, n^k\}$, uma tupla $u = (x_1, x_2, \ldots, x_k) \in V_G^k$ e um vértice $g \in V_g$, é definida por:

$$shift\left(f, u, g\right) = \langle f\left(u\left(x_1/g\right)\right), f\left(u\left(x_2/g\right)\right), \ldots, f\left(u\left(x_k/g\right)\right)\rangle$$

A notação $u\left(x_i/g\right)$ indica a troca do elemento na posição i, o x_i, por g.

Portanto, $shift\left(W_k^t, u, g\right)$ é uma tupla de tamanho k gerada a partir das cores da iteração t da tupla u trocando seus elementos na ordem por g. Exemplo: Seja $t = 0$, $k = 2$, $u = (1, 2)$ e $g = 3$, então:

$$shift\left(W_2^0, (1,2), 3\right) = \langle W_2^0\left(3,2\right), W_2^0\left(1,3\right)\rangle$$

Assim, é possível definir a cor das tuplas $u \in V_G^k$ para uma iteração $t+1$, W_k^{t+1}, como:

$$W_k^{t+1}\left(u\right) = \langle W_k^t\left(u\right), \text{SORT}\left\{\forall g \in V_G\ shift\left(W_k^t, u, g\right)\right\}\rangle$$

Onde a operação SORT contabiliza e ordena as cores retornadas pela operação *shift*, ou seja, SORT retorna uma sequência de números dada por

$$|g : shift\left(W_k^t, u, g\right) = 1|, |g : shift\left(W_k^t, u, g\right) = 2|, \ldots, |g : shift\left(W_k^t, u, g\right) = n^k|$$

Novamente, o método continua iterativamente até que $\forall u \in V_G^k\ W_k^l\left(u\right) = W_k^{l+1}\left(u\right)$ para algum l, e então $\overline{\text{W}}\left(G\right) = W_k^l$ é uma configuração estável para G.

Na pártica, é mantida uma *hash table* cujo valor de busca é o *hash* do valor de W_k^t, mesmo para $k = 1$, e o valor associado é um índice inteiro: caso o valor de $W_k^{t+1}\left(u\right)$ já exista nesta tabela, então a nova cor de u, ou v quando $k = 1$, é o índice associado ao *hash* de $W_k^{t+1}\left(u\right)$, caso contrário, é inserido uma nova entrada nesta tabela com um índice não utilizado ainda.

Como apresentado e devidamente provado em [6], $G \not\simeq H \to \overline{\text{W}}\left(G\right) \not\equiv \overline{\text{W}}\left(H\right)$, mas para se obter a informação que $G \simeq H$ é necessário testar se a configuração obtida representa de fato uma função de isomorfismo: caso contrário, é necessário retestar o método para um valor de k maior. Quando $k = n$, caso se mantenha a equivalência entre as configurações, mas esta não representa uma função isomórfica, nem nenhuma configuração obtida para algum k menor, então é possível concluir que os grafos não são isomórficas.

Portanto, é possível obter o seguinte algoritmo para testar o isomorfismo entre G e H:

1. Inicie $k = 1$
2. Seja W_G a configuração estável de G obtida pelo método k-dim WL com o valor atual de k
3. Seja W_H a configuração estável de H obtida pelo método k-dim WL com o valor atual de k
4. Se $W_G \not\equiv W_H$, então devolva `Falso`

5. Se $W_G \equiv W_H$ e estas configurações representam um isomorfismo, então devolva `Verdadeiro`
6. Se $k = |V_G|$, então devolva `Falso`
7. Incremente k por 1
8. Volte para o passo 2

Uma configuração é compatível com outra, se elas apresentam as mesmas classes de cor (para isso é necessário utilizar as mesmas *hash tables*) e o mesmo número de integrantes em cada classe. Duas configurações representam um isomorfismo, se o mapeamento de tuplas com todos os elementos iguais, entre as configurações, representam uma função isomórfica válida.

3.1.2 Coloração inicial

O método, como descrito na seção anterior, supõe que existe uma coloração inicial seja para os vértices, quando $k = 1$, seja para as tuplas, quando $k > 1$, mas não apresenta nenhum algoritmo para se obter tais colorações. Nesta seção será apresentado métodos para se obter tais colorações.

Para a versão 1-dim WL, é necessário que os vértices tenham uma coloração inicial: um método trivial para isto é agrupar os vértices com mesma valência em uma categoria, como cada vértice possui um valor para sua valência, a restrição de se ter exatamente uma relação de cor se mantém válida.

Já para a versão k-dim WL com $k > 1$, é necessário colorir tuplas de tamanho k e, novamente, tem-se uma separação por casos:

- Se $k = 2$, então é possível separa as tuplas em três categorias:
  - Caso os elementos da tupla seja idenênticos, ou seja, o mesmo vértice está na posição 1 e 2 da tupla, então tem-se uma categoria de cor.
  - Caso exista uma aresta entre os vértices da tupla, ou seja, o elemento 1 é adjacente ao elemento 2 da tupla, tem-se uma nova cor.
  - O terceiro caso é quando os elementos da tupla não são o mesmo e também não são adjacentes.
- Para $k > 2$, tem-se um algoritmo mais complexo: as tuplas são separadas por categorias de isomorfismo: Duas tuplas $S_1 = (x_1, \ldots, x_k)$ e $S_2 = (y_1, \ldots, y_k)$ são isomórficas se, e somente se, as seguintes propriedades valem com $\forall i, j \in \{1, \ldots, k\}$
  - $x_i = x_j \iff y_i = y_j$
  - $(x_i, x_j) \in E_G \iff (y_i, y_j) \in E_G$
  - $x_i \in C_t \iff y_i \in C_t$, com C_t a classse de cor do vertíce x_i

  Essa definição de isomorfismo foi obtida de [9].

3.2 Gerando instâncias de teste

Como já dito, para gerar algumas instâncias foi necessário implementar um gerador de grafos aleatório. Para isto foi implementado o método apresentado em [5]: este algoritmo recebe como entrada uma sequência de inteiros e produz um grafo cujos vértices possuem valência dada pela sequência fornecida. Como esses grafos são aleatórios, é possível gerar dois grafos com uma mesma sequência de valências para os vértices e que não são isomórficos: existe uma probabilidade, relativamente baixa, de se gerar o mesmo grafo e neste caso eles seriam isomórficos.

O método proposto, recebe uma sequência de inteiros não-negativos $d_1, d_2, \ldots, d_n$ e produz um grafo $G = (V, E)$ com $|V_G| = n$ e $|E_G| = m$ para $\sum_{i=1}^{n} d_i = 2m$, o algoritmo supõe que a sequência $d_1, d_2, \ldots, d_n$ é *graphical*, ou seja, existe pelo menos um grafo com esta sequência de valências. Assim o método segue:

1. Inicie o conjunto E sem elementos,
2. Inicie $\overrightarrow{\mathrm{d}}$ e $\overline{\mathrm{d}}$ com $(d_1, d_2, \ldots, d_n)$
3. Enquanto alguma aresta possa ser adicionada em E, repita:
   (a) Escolha dois vértices $v_i, v_j \in V_G$ com probabilidade proporcional a

$$\overrightarrow{\mathrm{d}}_i \overrightarrow{\mathrm{d}}_j \left(1 - \frac{\overline{\mathrm{d}}_i \overline{\mathrm{d}}_j}{4m}\right)$$

   De modo que $i \neq j$ e $(v_i, v_j) \notin E_G$

   (b) Decremente por 1 os valores de $\overrightarrow{\mathrm{d}}_i$ e $\overrightarrow{\mathrm{d}}_j$
4. Se $|E| < m$ então reporte `Falha`, caso contrário devolva $G = (V, E)$

Quando o maior grau da sequência, d_{max}, é da ordem de $O\left(m^{1/4-\epsilon}\right)$ para algum $\epsilon > 0$, a complexidade deste algoritmo é $O\left(md_{max}\right)$ [5]. Contudo, quando cresce o número de vértice e de arestas no grafo, a complexidade deste algoritmo cresce tornando-o inviável para grafos muito densos.

Assim era gerado uma sequencia *graphical* para os graus (esse sequência é dita *graphical*, pois era gerado valores de no máximo $|V_G| - 1$, limite da valência de um vértice, e a soma dos valores da sequência é sempre par, a soma dos graus dos vértices de um grafo é sempre o dobro do número de arestas e assim é um valor par) e com esta sequência é gerado dois grafos usando o algoritmo acima: é esperado que estes grafos seja não isomórficos. Para gerar instâncias isomórficas, foi necessário permutar os vértices de um dos grafos gerados.

O *dataset* apresentado em [10], apesar de apresentar um algoritmo para geração, não foi executado tal algoritmo, pois o próprio *dataset* continha instâncias que foram consideradas o suficiente.

O *dataset* apresentado em [11] não apresenta como foi gerado, apenas informa que são instâncias difíceis para o problema de isomorfismo.

4 Avaliação dos testes computacionais

Como já informado, os testes foram separados em três categorias:

- Instâncias aleatórios: são grafos gerados através do algoritmo apresentado em [5], esse conjunto de instâncias para teste apresentam grafos com tamanhos entre 10 e 90 vértices e número aleatório de arestas, para cada número de vértice foram geradas 15 instâncias de grafos isomórficos e 15 instâncias de grafos não isomórficos. Devido ao tempo computacional para gerar grafos maiores, o limitante no número de vértices foi 90;

- Instâncias Difíceis: Este conjunto de teste foi obtido de [11] e apresenta 39 instâncias sendo 8 isomórficas e o restante não isomórficas.

- Instâncias categóricas: Este conjunto de teste foi obtido do *dataset* de [10] e está organizado na seguintes categorias:

  - Grafos conectados aleatórios: São 1000 pares de grafos aleatórios com tamanho variando entre 20 e 1000 vértices, com densidade de arestas, η, assumindo os valores 0.01, 0.05 e 0.1;
  - *Regular Mesh*: Grafos regulares em formato de *mesh* 2D, 1000 instâncias de grafos com tamanhos entre 16 e 1024 nós, 3D, 800 instâncias de grafos com tamanhos entre 27 e 1000 nós, e 4D, 500 instâncias de grafos com tamanhos entre 16 e 1296 nós;
  - *Modified Mesh*: São as *mesh* da categoria anterior, mas com grau de irregularidade, ρ, assumindo os valores 0.2, 0.4 e 0.6.

  Como o *dataset* é para testar isomorfismos em grafos em em sub-grafos, apenas as instâncias que foram projetadas para testar isomorfismo em grafos foi utilizada. Novamente, das 1000 instâncias para cada tamanho, apenas as 15 primeiras foram testadas.

4.1 Resultados computacionais para o método *k*-dim WL

Como já informado, os conjuntos de testes que possuíam mais de uma instância para cada tamanho, as instâncias aleatórias e categóricas, foram executados 15 vezes com grafos de mesmo tamanho, em vértice, mas com diferentes conjuntos de arestas. Isso foi necessário para se obter um intervalo de confiança aceitável.

Para o cálculo do intervalo de confiança foi utilizado uma confiança de 95%, obtida através da Distribuição t de Student, pois esta se adéqua melhor as estatísticas obtidas dado o fato que os erros de medição são desconhecidos e devem ser estimados a partir dos dados medidos[25].

A medição dos tempos foi feita através da biblioteca *std::chrono* da linguagem C++, a medida de tempo obtida não considera o tempo de leitura dos grafos, apenas o tempo entre os preparativos de se executar o método e o resultado informado por este.

Os testes foram executados em uma máquina com:

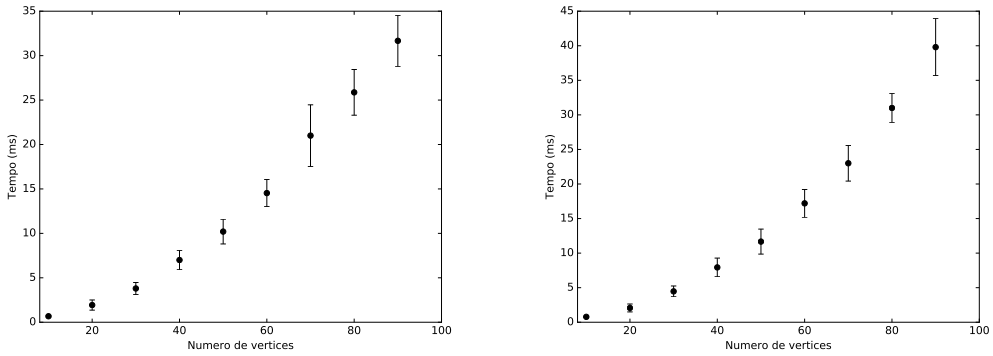
- Sistema operacional *macOS Sierra* na versão 10.12.1, com *kernel* na versão *Darwin 16.1.0*;
- Processador Intel Core i5 2,8 GHz com 2 *cores*;
- 8GB de memória RAM DDR3 com 1600 MHz de velocidade;
- 500GB de SSD;
- GCC/G++ *Apple LLVM version 8.0.0 (clang-800.0.42.1)*, *flags Wall, O3, g, std= c++0x* e bibliotecas *lm, lpthread*;

Durante a execução dos testes, apenas as atividades padrões do Sistema Operacional estavam sendo executadas.

Devido a natureza exponencial do método, foi determinado um tempo máximo de execução, um *time out*, para todas as instâncias individuais: o valor escolhido foi 5 minutos.

4.1.1 Instâncias aleatórios

Os resultados para as instâncias aleatórias foi obtido:



(a) Instâncias isomórficas (b) Instâncias não isomórficas

Figura 1: Resultados para instâncias formadas por grafos aleatórios

Observe que, apesar do crescimento no número de vértices o tempo computacional de execução continua baixo, crescendo lentamente quando comparado ao fato de ser exponencial a complexidade: para as instâncias testadas é possível que em nenhuma foi obtido um grafo super-regular que explore a complexidade exponencial.

Também é observado que o comportamento da curva é similar entre as instâncias isomórficas e não isomórficas, apesar de que a segunda necessita de mais tempo de execução. Mas dadas as barras de erro, os tempos são equivalentes.

4.1.2 Instâncias difíceis

Devido a grande variedade de grafos neste *dataset*, não é possível gerar gráficos. Contudo, das 39 instâncias, o método implementado apenas resolveu 20 em tempo inferior a 5 minutos:

Tabela 1: Tempo de execução para as instancias difíceis [11]

Instância	**Número de vértices**	**Tempo (ms)**
6	20	1508
7	25	246195
8	25	60689
10	11	57
12	28	2492
14	77	105190
20	13	129
21	34	6726
22	34	6361
23	34	6296
24	25	1285
25	14	166
29	16	690
30	25	4697
31	18	2
32	36	32437
33	40	41283
37	21	1890
38	8	5
39	3	115

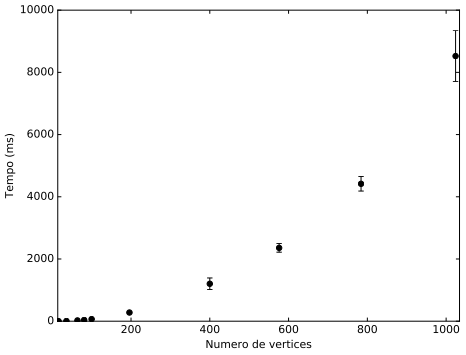
Mesmo estas instâncias foram executadas em tempo superior a apresentada na referência [11], contudo, como o algoritmo implementado para resolver este problema não foi apresentado, não é possível comparar as execuções.

4.1.3 Instâncias Categóricas

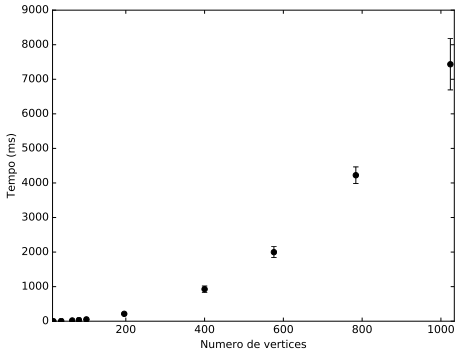
Como as instâncias estão separadas por categorias, é possível analisar o comportamento do método implementado em diferentes situações.

A primeira categoria a ser analisada é a execução em grafos regulares, *Regular Mesh*, neste conjunto de teste, nenhuma execução terminou antes de atingir 5 minutos: nem as *mesh* 2D, 3D e 4D para poucos vértices. Como os grafos são regulares, e possivelmente super-regulares, já era esperado que eles atingissem o pior caso do algoritmo e como uma das etapas é gerar todas as tuplas de um tamanho k, mesmo com 16 vértices já é possível ter que gerar tuplas com tamanho considerável e assim necessitando muito tempo e memória.

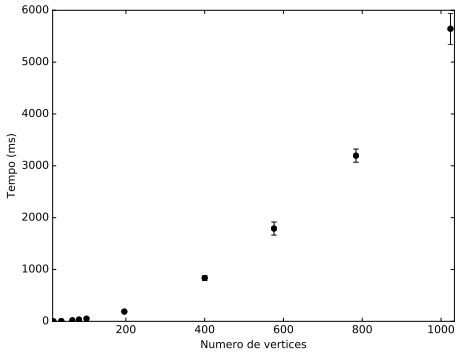
Já as *mesh* irregulares, *Modified Mesh*, foram executadas em baixo tempo computacional. Para as *mesh* 2D os resultados são:



(a) $\rho = 0.2$



(b) $\rho = 0.4$



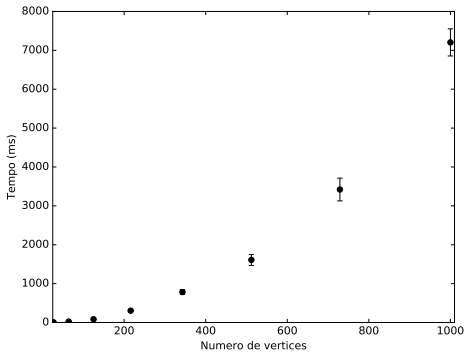
(c) $\rho = 0.6$

Figura 2: Resultados para instâncias *Modified Mesh* 2D

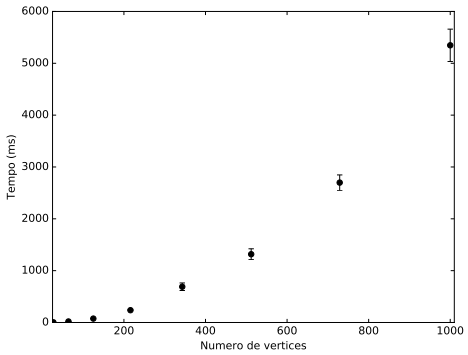
É notória a diminuição no tempo de execução conforma menos regular o grafo se torna, com o aumento do valor de ρ, isso já é esperado, pois quanto menos regular o grafo se torna, ou seja, quanto maior a quantidade de vértices com valências diferentes, menor deve ser o valor de k para se chegar a conclusão de que os grafos são ou não são isomórficos. Também existe uma grande diferença entre o tempo de execução destas instâncias e as instâncias aleatórias, apresentadas na figura 1: o tempo de execução aqui é maior, justamente pelo fato da instâncias aleatórias terei uma variação maior nos valores de valência.

Nas instâncias 3D, figura 3, e 4D, figura 4, há redução no tempo de execução com o aumento do ρ que é semelhante ao apresentado na *mesh* 2D. Também é observado que o tempo de execução diminui entre o conjunto 2D e 3D, enquanto que do conjunto 3D para 4D aumenta, e assim, aparentemente, não existe ligação entre este fator e o tempo de execução.

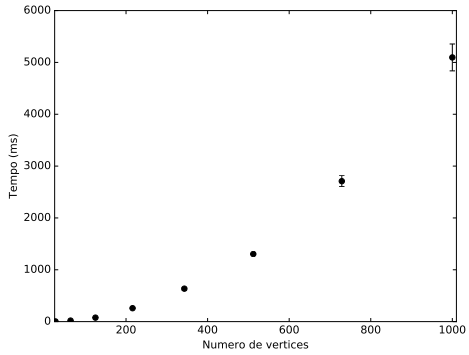
Apesar da figura 4 aparentar ter poucos pontos, isso se deve a esparsidade no tamanho dos grafos providos pelo *dataset*.



(a) $\rho = 0.2$



(b) $\rho = 0.4$



(c) $\rho = 0.6$

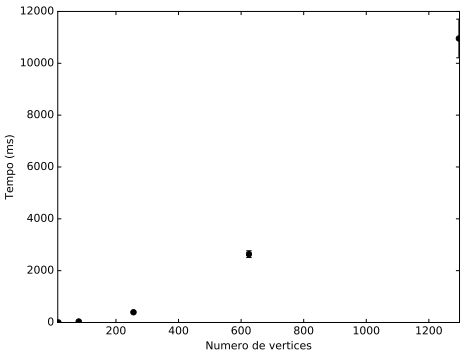
Figura 3: Resultados para instâncias *Modified Mesh* 3D

Para o conjunto com grafos conectados aleatórios tem-se a figura 5 que apresentam uma queda no tempo de execução, mas apresentam uma curva no mesmo formato dos conjuntos das *mesh*. Observe que o tempo decaiu com o aumento do η, mas variou pouca coisa para os valores 0.05 e 0.1, o que indica uma estabilidade com o aumento da densidade de arestas.

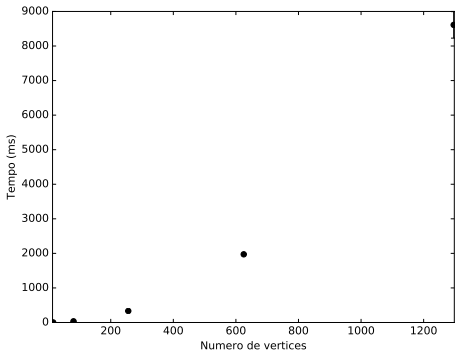
Apesar da quantidade de memória utilizada na execução das instâncias não ter sido medida, é esperado que ela também cresça exponencialmente com o aumento de k: o tamanho das tuplas cresce linearmente com k e a quantidade de tuplas crece exponencialmente com k, assim se k cresce será necessário utilizar mais espaço para armazenar as tuplas e para manter os valores da nas *hash tables*.

4.2 Resultados computacionais para o método força bruta

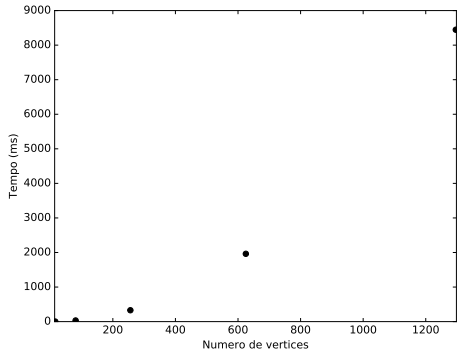
Utilizando o método força bruta, testar todas as permutações possíveis, também é exponencial, o número de permutações possíveis é $|V_G|!$. Como testar todas as permutações é muito custoso: no caso de se confirmar que dois grafos não são isomórficos, é necessário testar todas as permutações, para então se confirmar, as instâncias com mais de 10 vértices



(a) $\rho = 0.2$



(b) $\rho = 0.4$



(c) $\rho = 0.6$

Figura 4: Resultados para instâncias *Modified Mesh* 4D

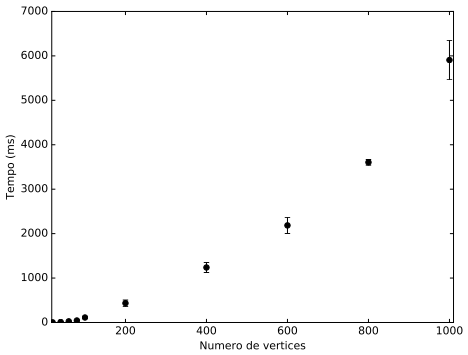
já não terminavam com menos de cinco minutos.

5 Conclusão

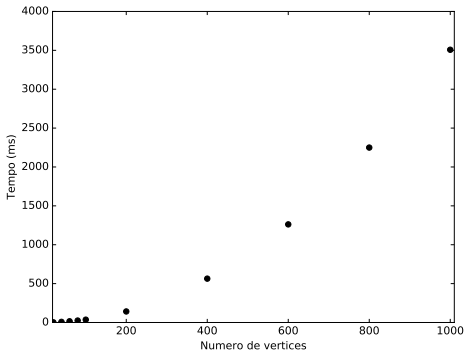
O objetivo deste projeto era explorar o problema de isomorfismo em grafos através de métodos, ainda que com complexidade de tempo e/ou espaço exponenciais, para resolver este problema.

Apesar de ser um problema com uma descrição simples e objetiva, não se sabe por completo sua complexidade e assim, apesar de todos os métodos propostos para resolução do problema não serem polinomiais, muitos métodos abordam particularidades de alguns grafos para resolver este problema em tempo polinomial.

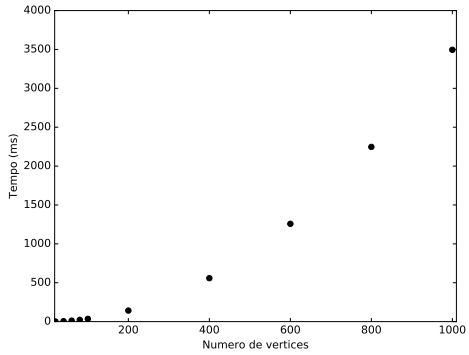
A escolha do método k-dim WL apareceu apropriada, pois é provado que para uma grande variedade de grafos, o método apresenta uma solução em tempo polinomial, diferente do método trivial de se tentar todas as permutações que é exponencial para todos os grafos, mas que para algumas categorias de grafos ainda possui complexidade exponencial de tempo



(a) $\eta = 0.01$



(b) $\eta = 0.05$



(c) $\eta = 0.1$

Figura 5: Resultados para instâncias de grafos conectados aleatórios

e memória.

Os resultados obtidos se comportaram como esperado: para grafos aleatórios ou com propriedades específicas, o resultados destas instâncias foram obtidos com rapidez, mesmo o número de vértices crescendo significativamente. Enquanto que para as instâncias de grafos regulares, o resultado não foi obtido em tempo hábil mesmo para grafos pequenos, seguindo o caminho do método de força bruta, fato que já era esperado devido a análise de complexidade do método.

Enfim, a implementação do método escolhido apresentou o comportamento esperado sendo muito rápida mesmo para grafos grande quando não regulares, e apresentando comportamento exponencial para grafos regulares: mesmo não obtendo uma reposta em menos de 5 minutos, é suposto que o algoritmo levaria tempo exponencial para responder. Mesmo sendo um método eficiente, talvez seja possível criar um algoritmo hibrido que para grafos não regulares utilize este método para resolver, enquanto que para os grafos regulares, um outro método possa ser utilizado.

Referências

[1] Vikraman Arvind and Gaurav Rattan. The parameterized complexity of geometric graph isomorphism. In *International Symposium on Parameterized and Exact Computation*, pages 51–62. Springer, 2014.

[2] László Babai. Monte-carlo algorithms in graph isomorphism testing. *Université tde Montréal Technical Report, DMS*, 1979.

[3] László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

[4] László Babai, D Yu Grigoryev, and David M Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 310–324. ACM, 1982.

[5] Mohsen Bayati, Jeong Han Kim, and Amin Saberi. A sequential algorithm for generating random graphs. *Algorithmica*, 58(4):860–910, 2010.

[6] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[7] Charles J Colbourn and Kellogg S Booth. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM Journal on Computing*, 10(1):203–225, 1981.

[8] Bireswar Das, Murali Krishna Enduri, and I Vinod Reddy. Polynomial-time algorithm for isomorphism of graphs with clique-width at most three. *arXiv preprint arXiv:1506.01695*, 2015.

[9] Brendan L Douglas. The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

[10] Pasquale Foggia, Carlo Sansone, and Mario Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, pages 176–187, 2001.

[11] Graphs for GI testing. `http://funkybee.narod.ru/graphs.htm`. Acesso: 10-11-2016.

[12] Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. *arXiv preprint arXiv:1505.03737*, 2015.

[13] John E Hopcroft and Jin-Kue Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184. ACM, 1974.

[14] Pavel Klavik, Dušan Knop, and Peter Zeman. Graph isomorphism restricted by lists. *arXiv preprint arXiv:1607.03918*, 2016.

[15] Eugene M Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1):42–65, 1982.

[16] Eugene M Luks. Parallel algorithms for permutation groups and graph isomorphism. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 292–302. IEEE, 1986.

[17] Eugene M Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 652–658. ACM, 1999.

[18] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[19] Gary Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 225–235. ACM, 1980.

[20] Uwe Schöning. Graph isomorphism is in the low hierarchy. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 114–124. Springer, 1987.

[21] Aaron Snook, Grant Schoenebeck, and Paolo Codenotti. Graph isomorphism and the lasserre hierarchy. *arXiv preprint arXiv:1401.0758*, 2014.

[22] Daniel A Spielman. Faster isomorphism testing of strongly regular graphs. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 576–584. ACM, 1996.

[23] Nenad Trinajstic et al. *Chemical graph theory*. CRC press, 1992.

[24] B Weisfeiler and A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[25] Wikipedia. Student's t-distribution — wikipedia, the free encyclopedia, 2016. [Online; accessed 22-November-2016].

[26] Viktor N Zemlyachenko, Nickolay M Korneenko, and Regina I Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.