CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

UNIDADE DE PÓS-GRADUAÇÃO, EXTENSÃO E PESQUISA

Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos

Victor Alexandre Ploeger Mansueli

ESTUDO DE GENEALOGIAS POR MEIO DA ANÁLISE DE REDES SOCIAIS E DA TEORIA DOS GRAFOS:

proposta de uma nova representação

São Paulo

2018

Victor Alexandre Ploeger Mansueli

ESTUDO DE GENEALOGIAS POR MEIO DA ANÁLISE DE REDES SOCIAIS E DA TEORIA DOS GRAFOS:

proposta de uma nova representação

Dissertação apresentada como exigência parcial para a obtenção do título de Mestre em Gestão e Tecnologia em Sistemas Produtivos do Centro Estadual de Educação Tecnológica Paula Souza, no Programa de Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos, sob a orientação do Prof. Dr. Marcelo Tsuguio Okano.

São Paulo

2018

FICHA ELABORADA PELA BIBLIOTECA NELSON ALVES VIANA
FATEC-SP / CPS – CRB8 8281

Mansueli, Victor Alexandre Ploeger
M289 Estudo de genealogias por meio da análise de redes sociais e da teoria dos grafos: proposta de uma nova representação / Victor Alexandre Ploeger Mansueli. – São Paulo : CPS, 2018.
104 f. : il.

Orientador: Prof. Dr. Marcelo Tsuguio Okano
Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos) - Centro Estadual de Educação Tecnológica Paula Souza, 2018.

1. Sistemas Produtivos. 2. Genealogias. 3. Análise de redes sociais. 4. Teoria dos Grafos. I. Okano, Marcelo Tsuguio. II. Centro Estadual de Educação Tecnológica Paula Souza. III. Título.

VICTOR ALEXANDRE PLOEGER MANSUELI

ESTUDO DE GENEALOGIAS POR MEIO DA ANÁLISE DE REDES SOCIAIS E DA TEORIA DOS GRAFOS:
proposta de uma nova representação

Prof. Dr. Marcelo Tsuguio Okano

Prof. Dr. Napoleão Verardi Galegale

Prof. Dr. Felipe Mancini

São Paulo, 08 de novembro de 2018

Aos meus antepassados europeus que,
após cruzarem entreguerras o Atlântico,
demonstraram a resiliência necessária para
reiniciar a vida em um novo continente.

AGRADECIMENTOS

Agradeço primeiramente ao meu orientador, Prof. Dr. Marcelo Tsuguio Okano, que me acolheu para o Programa, e me guiou durante a elaboração deste estudo. À Unidade de Pós-Graduação, Extensão e Pesquisa do CEETEPS, pelo oferecimento do Curso. Aos funcionários, professores e coordenadores, pelo compartilhamento de experiências e de conhecimento. Aos meus colegas de turma, pelo alívio do peso da jornada. À minha família, Kelly (companheira de vida) e Bono. Aos meus irmãos e pais, Helmina e Hélio (*In memoriam*).

„Man sagt,
die Zeit heilt alle Wunden,
doch was ist,
wenn die Zeit selbst die Krankheit ist?“

WIM WENDERS („Der Himmel über Berlin“, 1987)

RESUMO

MANSUELI, V. A. P. Estudo de genealogias por meio da análise de redes sociais e da teoria dos grafos: proposta de uma nova representação. 103 f. Dissertação (Mestrado Profissional em Gestão e Desenvolvimento da Educação Profissional). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2018.

A Genealogia estuda a ancestralidade de indivíduos, onde se estabelecem parentescos e relacionamentos, e o interesse por essa área tem sido continuamente despertado pelas novas tecnologias de acesso à informação. Famílias podem ser vistas como redes sociais e, como tal, seus componentes estruturais são passíveis de modelagens matemáticas e a Teoria dos Grafos, em particular, auxilia na diagramação de representações genealógicas, provendo três formalizações: *Ore Graphs*, *P-Graphs* e *Bipartite P-Graphs*. Estas, no entanto, possuem algumas limitações quando tratadas isoladamente. Este trabalho propõe uma representação alternativa e complementar: os *K-Graphs* (Grafos de Parentesco). Calcado no método *Design Science*, o novo grafo foi implementado em ambiente computacional, e sua eficácia demonstrada, aplicado em desmembramentos de localidades geográficas, cujas características remetem aos preceitos da Genealogia. Concluiu-se, desta forma, que *K-Graphs* podem ser utilizados em outras áreas do conhecimento, tanto para análise computacional como visual de redes de relacionamentos, configurando uma contribuição empírica deste estudo.

Palavras-chave: Sistemas Produtivos. Genealogias. Análise de Redes Sociais. Teoria dos Grafos.

ABSTRACT

MANSUELI, V. A. P. Estudo de genealogias por meio da análise de redes sociais e da teoria dos grafos: proposta de uma nova representação. 103 f. Dissertação (Mestrado Profissional em Gestão e Desenvolvimento da Educação Profissional). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2018.

Genealogy comprises the ancestry of individuals, where kinship and relationships are established, and the interest in this topic has been continuously heightened by new information access technologies. Families can be seen as social networks and, as such, their structural components are subject to mathematical modeling and Graph Theory, specially, aids in diagramming genealogical representations by providing three formalizations: Ore Graphs, P-Graphs and Bipartite P-Graphs. These, however, have some limitations when treated apart. This work proposes an alternative and complementary representation: the K-Graphs (Kinship Graphs). Based on the Design Science method, the new graph was implemented in a computational environment, and its efficacy demonstrated, applied in the dismemberment of geographic locations, whose characteristics comply with Genealogy precepts. It was concluded that K-Graphs can be applied to other fields of knowledge as well, both for computational and visual analysis of relationship networks, which constitutes an empirical contribution of this study.

Keywords: Productive Systems. Genealogies. Social Network Analysis. Graph Theory.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Poucos de nós tem ciência, de fato, de nossas origens familiares e se propuseram, alguma vez, a pesquisar a própria herança. Este estudo objetiva auxiliar nesta busca, proporcionando uma forma documental de registro de nossas próprias vidas, e as dos que nos antecederam.

Preservar a história de uma família é um ato de reverência e respeito pelos nossos antepassados. É um agradecimento pelo legado proporcionado, que reflete diretamente na nossa condição atual como elos de uma cadeia que remonta à gerações. Ao rastrear as origens dessa cadeia, desvendam-se oportunidades de compreender contextos, situações e circunstâncias que nos permitem explicar muito dos nossos valores e comportamentos atuais.

Pesquisar o que nos liga diretamente a um ancestral proporciona uma conexão com o passado, visto não como uma sequência irremediável de eventos no tempo, mas como um fio condutor de tudo o que somos e representamos hoje. E, atuando como agentes inseridos nessa cadeia temporal, ao documentar formalmente esta pesquisa fornecemos o instrumental necessário para a posteridade.

“Ao contar histórias através do delineamento de nossas genealogias, estamos falando *a* vida (e não *da* vida)” (ROSO, 2010). Falamos, sobretudo, acerca de uma trama única de fatos inter-relacionados, pulsantes, que ecoam e remetem às nossas conexões singulares com o passado. E ao representar essa rica herança por intermédio do esboço de genealogias, preservamos a nossa própria identidade.

Genealogias, entendidas como estudos da ancestralidade de indivíduos, são uma forma de contar estórias: de quem somos, de onde viemos. São sobre pessoas, suas vidas, sobre como suas estórias nos moldam (BALL, 2017).

O seu estudo é multidisciplinar, envolvendo aspectos biológicos e sociológicos. Desenvolve-se no âmbito da história das famílias, e é integralizada por outras ciências, como a Psicologia (ROSO, 2010).

Uma forma de estudar genealogias é por meio de redes sociais, objetos de análise em suas estruturas e relações. A Análise de Redes Sociais[1] (ARS) é uma abordagem de pesquisa distinta dentro das ciências sociais e comportamentais; ela se baseia na premissa da importância dos relacionamentos entre unidades interativas. O conceito de rede enfatiza o fato de que cada indivíduo possui laços com outros indivíduos (WASSERMAN, FAUST, 1994). Ela investiga, portanto, estruturas sociais, tendo emergido como uma técnica essencial na moderna sociologia, e adquirindo relevância em estudos de outras áreas (antropologia, biologia, geografia, história, ciência da computação, entre outras).

Visualizações genealógicas são utilizadas para ajudar a entender e compartilhar a estória do passado com outras pessoas (BALL, 2017). Para Roso (2010), “uma árvore genealógica pode servir como uma retomada de um sistema globalizado de comunicação entre o passado e o presente, entre os mais velhos e os jovens”.

Tradicionalmente, utiliza-se a forma diagramática de árvores para visualização de genealogias (uma analogia natural nos casos em que novas gerações estão posicionadas no topo, e as mais antigas na base, em um “afunilamento” em direção aos antepassados em comum). Derivações podem ser encontradas em mapas de ancestralidade onde um indivíduo aparece à esquerda, com seus ancestrais à direita; ou em mapas de descendência onde um indivíduo se posiciona no topo, na região mais estreita de uma “árvore invertida”, representando o ancestral mais antigo.

Diagramas genealógicos são diagramas de grafos, pois eles contêm pessoas individuais como nós, ligadas por relações de afinidade, parentesco e irmandade (BARNES, HARARY, 1983).

1 Tradução da expressão inglesa “Social Network Analysis” (SNA)

Na conjuntura de genealogias e redes, a Teoria dos Grafos fornece três representações visuais: *Ore Graphs*, *P-Graphs* e *Bipartite P-Graphs* (BATAGELJ, MRVAR, 2008). Todas elas estabelecem relações de parentesco (consanguíneos ou não) através de seus componentes (nós, arestas e arcos). Cada uma delas possui particularidades, limitações, vantagens e desvantagens entre si. Estas características serão exploradas, detalhadas e comparadas no decorrer deste estudo.

1.1 Questão de Pesquisa

Não obstante as similaridades e complementaridades entre as três opções de grafos genealógicos existentes, limitações são observadas isoladamente, e determinados vínculos familiares não são previstos como, por exemplo, representação de gemelaridades, indicação de parentesco entre irmãos e meio-irmãos, e múltiplos vínculos conjugais.

Desta forma, é possível propor uma representação genealógica alternativa, suprindo as deficiências apontadas nas três representações que existem atualmente?

1.2 Objetivos

Este estudo tem os seguintes objetivos:

- Principal

    1. Propor uma nova representação de genealogias na Teoria dos Grafos.

- Específicos

    1. Implementar a nova representação em ambiente computacional;
    2. Demonstrar a eficácia da nova representação em uma outra área de conhecimento.

1.3 Organização

O Capítulo 1, introdutório, situa brevemente a Genealogia no contexto de Redes Sociais e na Teoria dos Grafos, e apresenta os objetivos – principal e específicos – que se busca atingir com este estudo.

O Capítulo 2 destaca a relevância atual da Genealogia à luz das novas tecnologias de informação e comunicação. Discorre sobre os temas tratados, embasados na pesquisa bibliográfica realizada, aprofundando conceitos e expondo os grafos genealógicos existentes.

Os métodos utilizados são apresentados e referenciados no Capítulo 3, assim como a justificação de suas escolhas para atender os objetivos propostos. Partindo do embasamento teórico-conceitual, este estudo também incorporou o desenvolvimento de um artefato, como prova da aplicação prática do conhecimento adquirido.

No Capítulo 4 os resultados do estudo são demonstrados na forma de proposição de um novo grafo genealógico: os *K-Graphs*. São relacionadas suas vantagens e características, além das convenções utilizadas. O exemplo de uma genealogia familiar é fornecido, com quatro gerações, para ilustrar a disposição de indivíduos e seus relacionamentos. Segue-se uma análise acerca de possíveis aplicações do novo grafo para representação de redes de relacionamento em outras áreas, focada nas contribuições empíricas deste estudo. Como prova de conceito, a criação de um artefato (protótipo de programa) visou demonstrar a eficácia dos *K-Graphs* quando aplicado na estruturação e visualização de uma rede de localidades geográficas.

Por fim, o Capítulo 5 apresenta as conclusões e sugestões de pesquisas futuras, além do reconhecimento das limitações deste trabalho. Introduz, ainda, a emergente área da Afinografia, incorporando relacionamentos familiares mais amplos e não convencionais.

2 FUNDAMENTAÇÃO TEÓRICA

A busca pela nossa própria história, construída de geração em geração, está fundamentada na pesquisa de nossos ancestrais, nos relacionamentos entre cada componente da nossa estrutura familiar. E uma forma de preservar essa herança é documentar esta pesquisa por meio de representações genealógicas.

Nossa identidade está intimamente entrelaçada com os genes que compartilhamos a cada geração de nossas famílias. Sistematizar o estudo de nossos progenitores resgata o nosso sentido de pertencimento, a compreender o nosso legado humano.

Para Roso (2010), “o interesse pela genealogia tem sido revigorado nas sociedades em rede pelas mais diversas disciplinas". É inquestionável que a acessibilidade aos repositórios de dados pessoais é um componente crucial no despertar deste interesse. E, nesse aspecto, Sistemas de Informação são agentes que viabilizaram esse processo, por intermédio do uso e da disseminação de tecnologias que permitem a interface entre usuários e bancos de dados genealógicos.

O advento da Internet possibilitou uma conectividade sem precedentes, e pessoas separadas no tempo e/ou espaço que, a princípio, possuíam apenas um sobrenome em comum, fazem uso da “sociedade em rede” para localizar parentes e estabelecer (ou restabelecer) vínculos, resgatando histórias familiares em comunidades *online*.

Da mesma forma, uma miríade de ferramentas são disponibilizadas na Internet para a criação de árvores genealógicas pelos próprios usuários, suscitando um interesse cada vez mais crescente por rastrear, mapear, representar e visualizar mapas de ancestralidade. Entre as principais, pode-se citar:

- Ancestry (https://www.ancestry.com)
- FamilySearch (https://familysearch.org/)
- Geni (https://www.geni.com/)
- MyHeritage (https://www.myheritage.com.br/)
- WikiTree (https://www.wikitree.com/)

Alguns destes *sites* celebram parcerias com instituições que preservam a memória de pessoas envolvidas em processos e fluxos migratórios, como é o caso do Museu da Imigração do Estado de São Paulo, que procura valorizar o encontro das múltiplas histórias e origens, propondo o contato com as lembranças daquelas pessoas que vieram de terras distantes, suas condições de viagem, e adaptação aos novos trabalhos (MUSEU DA IMIGRAÇÃO DO ESTADO DE SÃO PAULO, 2018).

Já as ferramentas disponíveis nestes *sites* proveem praticidade de uso, engenhos de busca, compartilhamento de informações, serviços adicionais (como testes de DNA), recursos para exportação de arquivos, entre outras opções. Muitas delas são gratuitas em suas versões básicas. No entanto, todas servem a um mesmo propósito: facilitar a pesquisa da história de nossas famílias.

Além de *sites* de geração automática de árvores genealógicas, configurou-se outro fenômeno proporcionado pela Internet: os encontros ou reuniões de famílias, onde seus membros celebram tradições e fortalecem laços afetivos. Se trata de mais um exemplo das possibilidades de uma sociedade fortemente conectada em rede, onde a Genealogia assume um papel central como integradora de unidades familiares.

Pelo acima exposto, pode-se constatar que o interesse pela busca da nossa própria história foi revigorado com a disseminação das novas tecnologias de informação, e a cada instante surgem novas ferramentas e recursos para o arquivamento, documentação e cruzamento de dados genealógicos.

Neste contexto, a Matemática é uma importante aliada no processo de estruturar, quantificar e correlacionar estas informações de parentesco para análise, e a Teoria dos Grafos, em particular, auxilia na diagramação de representações

genealógicas, fornecendo visualizações intuitivas de seus componentes e relacionamentos.

Este capítulo relaciona os tópicos centrais do embasamento teórico deste estudo. Contextualiza redes sociais, como os métodos utilizados para estudá-las são aplicados na representação de genealogias, introduz a Teoria dos Grafos e, por fim, analisa as opções que esta fornece para formalizar relações de parentesco.

2.1 Redes Sociais

Mrvar (2016), lembra que o conceito de redes extensas pode ser encontrado, por exemplo, nas conexões entre pessoas (relações de parentesco, amizade), no comércio entre organizações e países, em citações e co-autorias, em chamadas telefônicas, nos fluxogramas da Ciência da Computação, na Química (moléculas orgânicas), nas conexões entre palavras em textos ou dicionários, nos transportes, etc.

Já a expressão “rede social” se refere a um conjunto de atores e os laços entre eles. A ARS fornece um meio preciso de definir importantes conceitos sociais, uma alternativa teórica para a hipótese de atores sociais independentes, e um quadro de referência para testar teorias sobre relacionamentos sociais estruturados (WASSERMAN, FAUST, 1994).

A expressão adquiriu uma conotação bem mais abrangente, abarcando não somente grupos localizados em um mesmo espaço geográfico, mas também aqueles oriundos de ambientes virtuais. Deste modo, faz-se necessário delimitar o campo de estudo de genealogias, vistas estritamente sob esta ótica.

Apesar deste estudo estar inserido no âmbito de redes sociais, estas serão tratadas unicamente sob a perspectiva de grupos de indivíduos que possuem algum parentesco entre si, através dos mesmos antepassados, oriundo de vínculos conjugais (casamentos) ou filiais (relação pai-filho). Ou seja, que compartilham, necessariamente, de uma mesma filiação ou linhagem.

Embora, por definição, sejam consideradas “redes”, comunidades virtuais, *sites* de relacionamento e mídias sociais em geral fogem do escopo deste estudo, na medida em que o propósito destes grupos não é, exclusivamente, o estabelecimento e manutenção de vínculos familiares.

Foram adotadas, ainda, as seguintes premissas: descendentes serão considerados como tal apenas sob a estrita denotação biológica do termo, ou seja, por laços de sangue (consanguinidade) com seus progenitores; e somente considerados para representação casamentos entre pessoas de sexo oposto.

Igualmente, casos de perfilhações (adoções) e indivíduos oriundos de endogamias também não foram contemplados no escopo do presente estudo.

2.2 Genealogias

A Genealogia objetiva estabelecer a origem de indivíduos e famílias, por meio do mapeamento de seus ancestrais. Composta por uma série de dados, possibilita a reconstrução da história ao apresentar, usualmente em forma de diagrama, a ascendência de um indivíduo com a indicação de sucessivas gerações. Para Roso (2010), uma definição mais abrangente de genealogia seria o “estudo do parentesco”.

Uma rede genealógica corresponde a um fenômeno essencialmente histórico. Sua trama, definida nos termos de seu reconhecimento pelos que dela são criadores e criaturas, se desdobra no tempo e no espaço. Seus fios provêm de lembranças e de esquecimentos daqueles que a tecem (DAL POZ, SILVA, 2008).

Famílias são compostas por pessoas ligadas entre si por casamentos ou filiações e, como sistemas sociais, sua estrutura relacional é constituída pelos padrões de relacionamento entre seus integrantes.

A presença de informação relacional é uma característica crítica e definidora de uma rede social. Relações de parentesco tem sido estudadas utilizando métodos de rede por muitos anos. Laços podem ser baseados em matrimônios ou

relacionamentos de descendência, e relacionamentos conjugais ou familiares podem ser descritos com o uso de métodos de redes sociais (WASSERMAN, FAUST, 1994).

Parentesco é uma relação social fundamental, a qual é extensivamente estudada por antropologistas e historiadores. Em contraste com as pessoas que montam suas próprias árvores genealógicas, cientistas sociais estão primariamente interessados nas genealogias de comunidades inteiras (NOOY, MRVAR, BATAGELJ, 2011).

Os traços distintivos de redes de parentescos residem menos na forma como seus laços constitutivos são definidos e estabelecidos, do que na forma como estes laços são organizados. Redes de parentesco são caracterizadas pela interação de três princípios fundamentais: filiação, casamento, e gênero (HAMBERGER, HOUSEMAN, WHITE, 2011).

White, Batagelj e Mrvar (1999) observam que os estudos de parentesco e de redes de casamentos em uma grande escala, como embasamento e fundamentação para o estudo de outros processos sociais, podem demonstrar possuir amplas propriedades que irão mudar a nossa concepção de como sociedades, sistemas de mercado, e destacadas instituições sociais são organizadas.

2.2.1 O padrão GEDCOM

O formato de dados GEDCOM é uma linguagem de representação de dados de propósito geral, que visa representar qualquer tipo de informação estruturada em um meio sequencial (ROOTSWEB, 2017).

GEDCOM foi desenvolvido pelo Departamento de História da Família da Igreja de Jesus Cristo dos Santos dos Últimos Dias, visando prover um formato flexível e uniforme para troca de dados genealógicos eletrônicos. GEDCOM é um acrônimo para ***GE****anological* ***D****ata* ***COM****munication*. O seu propósito é promover o compartilhamento de informação genealógica, e o desenvolvimento de uma gama

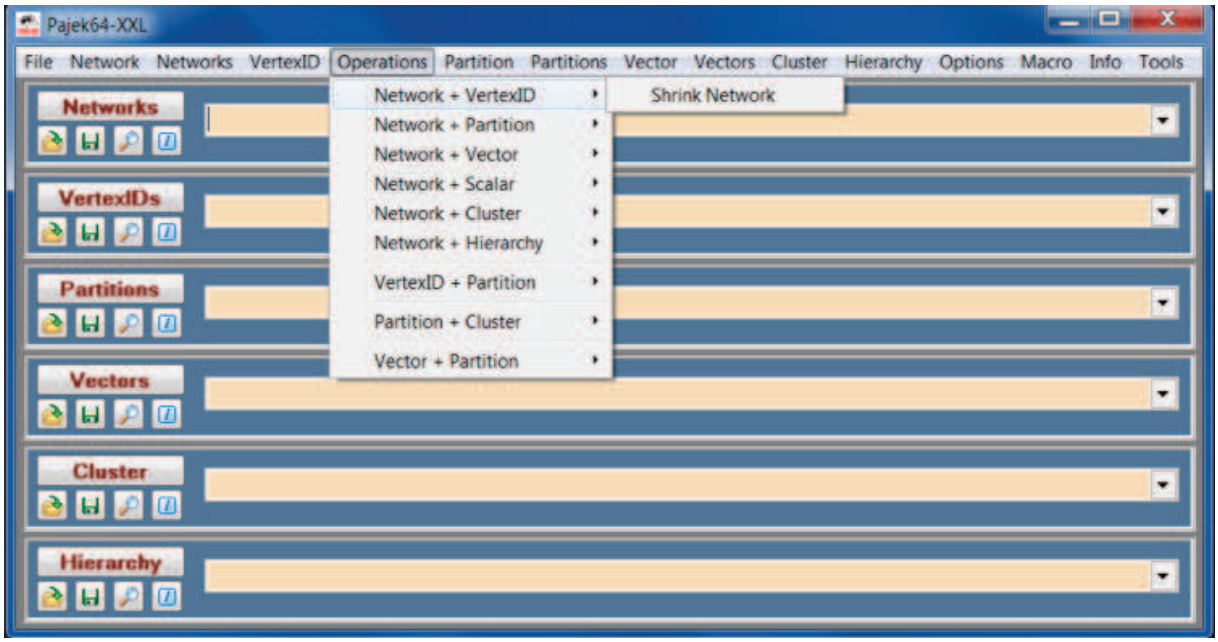
variada de produtos de *software* interoperáveis, para auxiliar genealogistas, historiadores e outros pesquisadores (ROOTSWEB, 2017).

Este formato padrão de dados para genealogias contém recursos para armazenar toda espécie de informação sobre as pessoas e eventos como, por exemplo, seus casamentos. Na Internet estão disponíveis excelentes *softwares* gratuitos e vários bancos de dados genealógicos (NOOY, MRVAR, BATAGELJ, 2011).

Entre os *softwares* que fazem uso desse padrão, *Pajek* é um dos mais conhecidos para análises complexas e visualização de redes densas (PAJEK, 2017).

A janela principal da versão 64-XXL é exibida na Figura 1.

Figura 1 – Janela principal do *software* Pajek



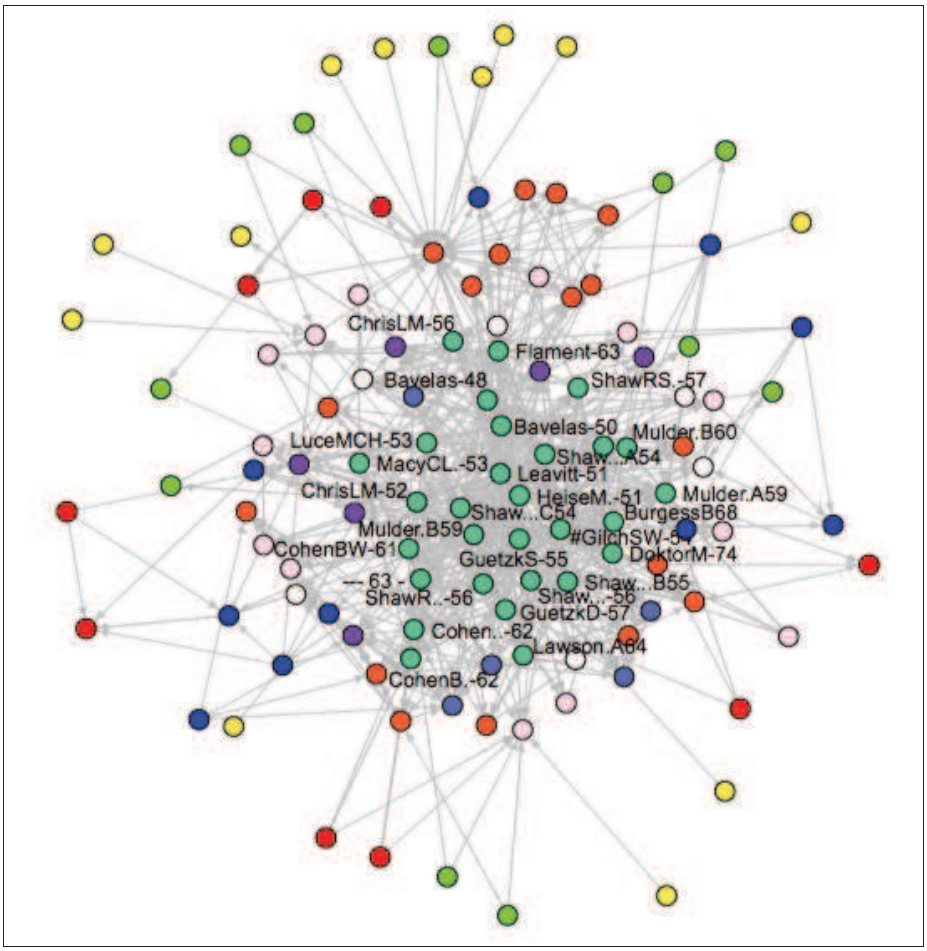
Fonte: Extraído de PAJEK, 2017

O desenvolvimento do programa teve início em 1996, como tese de Andrej Mrvar na Faculdade de Ciência da Computação e Informação na Universidade de Liubliana (Eslovênia). Embora sendo atualizado há mais de 20 anos, e com vários

sistemas integráveis, ainda é considerado o único programa disponível no mercado com capacidade de processar imensas redes (contendo bilhões de vértices) (MRVAR, 2016).

A Figura 2 ilustra um grafo gerado pelo programa: centralidade em uma rede de literatura.

Figura 2 – Exemplo de grafo gerado pelo *software* Pajek



Fonte: Extraído de NOOY, MRVAR, BATAGELJ, 2011

Parte da construção de bancos de dados genealógicos ou de parentesco, como no caso do formato GEDCOM, é a atribuição de um conjunto único de números ou identificadores não apenas para indivíduos, mas também para famílias nucleares, definidas como uniões e sua progênie (caso exista). Para cada indivíduo cujo pai ou pais são conhecidos, pode ser atribuído um identificador FAMC (***FAM****ily*

*of **C**hild*). E para cada indivíduo pode ser atribuído um identificador FAMS (***FAM**ily of **S**pouse*), para cada família nuclear na qual ele ou ela possui um cônjuge ou filho. Sociologistas denominam estas identificações como *família de orientação* versus *família de procriação*. Quando duas pessoas se casam, sua unidade de ligação é codificada sob um identificador FAMS em comum (WHITE, BATAGELJ, MRVAR, 1999).

A Listagem 1 exibe um exemplo do padrão GEDCOM: um trecho de arquivo de extensão “ged”, o qual foi exportado pela ferramenta Geni (GENI, 2017), contendo dados parciais do autor e de um de seus ancestrais.

Listagem 1 – Exemplo da estrutura de dados do padrão GEDCOM, com os identificadores FAMC e FAMS destacados

```
0 @I4796223@ INDI
 1 NAME Victor Alexandre Plöger /Mansueli/
  2 GIVN Victor Alexandre Plöger
  2 SURN Mansueli
 1 SEX M
 1 BIRT
  2 DATE 10 AUG 1975

...

 1 _EMAIL victormansueli@gmail.com
 1 FAMC @F40304511690700111121@

...

0 @I283082953730003009@ INDI
 1 NAME Ludwig /Plöger/
  2 GIVN Ludwig
  2 SURN Plöger
 1 SEX M
 1 DEAT
 1 BIRT
  2 DATE 12 JUN 1870
  2 ADDR
   3 CITY Buer
   3 CTRY Germany
 1 FAMS @F40304511857000111135@

...
```

Fonte: Exportado de GENI, 2017

2.3 Teoria dos Grafos

Em termos históricos, já na década de 1930 redes sociais eram representadas na Teoria dos Grafos, área da Matemática cuja origem remonta ao século XVIII (CHARTRAND, LESNIAK, ZHANG, 2010).

Grafos tem sido largamente utilizados na ARS como um meio de representar formalmente relações e quantificar importantes propriedades sócio-estruturais (WASSERMAN, FAUST, 1994).

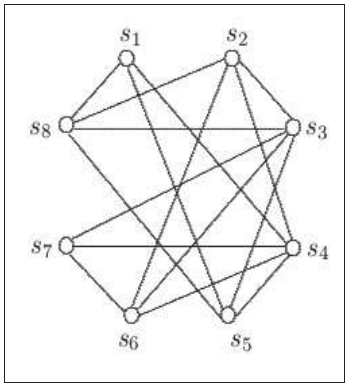
Diagramas deste tipo estão em uso prático em várias partes do mundo, e descrições orais destas configurações de relações estão entre nós por milhares de anos. Uma vez que a Teoria dos Grafos foi vista como relevante para a análise de redes sociais, diagramas genealógicos, como grafos, se tornaram um terreno óbvio para aplicação desta Teoria (BARNES, HARARY, 1983).

Um grafo é um modelo para uma rede social, com uma relação não-direcional dicotômica: isto é, um laço ou está presente ou está ausente entre cada par de atores. Relações não-direcionais incluem, entre outras coisas, algumas relações de parentesco como “é casado(a) com” ou “é um parente consanguíneo de” (WASSERMAN, FAUST, 1994).

Em sua formalização matemática, um grafo é um conjunto V finito de objetos chamados “vértices” (também “pontos” ou “nós”), juntamente com um conjunto possivelmente vazio A, de subconjuntos de 2-elementos de V chamado arestas (também “linhas” ou “elos”) (CHARTRAND, LESNIAK, ZHANG, 2010).

A Figura 3 exemplifica o diagrama de um grafo, com seus pares de vértices e arestas. Neste grafo hipotético, os vértices foram nomeados para identificação dos atores envolvidos, e as arestas representam os relacionamentos existentes entre cada par de vértices.

Figura 3 – Exemplo de um grafo, com 8 vértices identificados



Fonte: Extraído de CHARTRAND, LESNIAK, ZHANG, 2010

Além de suas aplicações em Engenharia e Ciência da Computação, existe uma lista virtualmente infinita de problemas que podem ser solucionados com a Teoria dos Grafos. Alguns exemplos onde grafos são utilizados incluem linguística, estruturas sociológicas, economia, cibernética, inteligência artificial, reconhecimento de padrões, genética, etc. Em algumas destas aplicações, o uso de grafos atende apenas o propósito trivial de representação visual. Existem muitos casos, entretanto, onde resultados importantes e não tão óbvios são obtidos através do uso aprofundado da Teoria dos Grafos (NARSINGH, 1974).

É oportuno ressaltar que, embora este estudo enfoque a Teoria dos Grafos no contexto de genealogias, as aplicações desta Teoria, como demonstrado, se estendem claramente para diversas áreas do conhecimento, e a representação, aqui proposta, de conexões entre atores participantes de uma rede social familiar poderia, da mesma forma, ser utilizada em outras possíveis aplicações, onde os princípios da Genealogia possam ser respeitados, e as estruturas hierárquicas mantidas.

2.3.1 Propriedades de Grafos

Algumas propriedades de grafos são pertinentes quando utilizados na representação de genealogias. Elas permitem inferir informações relacionais entre os componentes de uma rede de parentesco, identificar e quantificar núcleos familiares, gerações e religações, além de conceitos como adjacência, incidência e propriedades estruturais de coesão.

Propriedades são usadas para determinar a conectividade de grafos, definir a distância entre díades (pares), e identificar vértices e arestas críticos. Particularmente, “caminhos” nos permitem calcular a distância entre dois vértices. Outros tipos de caminhos são “trilhas”, “percursos”, “*tours*” e “ciclos”. A partir destas propriedades são definidas outras como distância geodésica, diâmetro e excentricidade (WASSERMAN, FAUST, 1994).

Outras propriedades relevantes em grafos genealógicos são a sua densidade (proporção de linhas existentes em relação às possíveis), e o grau de seus vértices (quantidade de linhas incidentes em cada um deles).

Em representações genealógicas, o cálculo da distância entre indivíduos (vértices) através dos caminhos disponíveis é de fundamental importância na definição das gerações e graus de parentesco.

2.3.2 Grafos Genealógicos

As próximas seções descrevem as formalizações existentes na Teoria dos Grafos para representações de genealogias: *Ore Graphs*, *P-Graphs* e *Bipartite P-Graphs*.

a) *Ore Graphs*

São grafos nomeados em homenagem ao matemático norueguês Øystein Ore. Segundo Hamberger, Houseman e White (2011), constituem a representação mais convencional de redes de parentesco, onde vértices representam indivíduos,

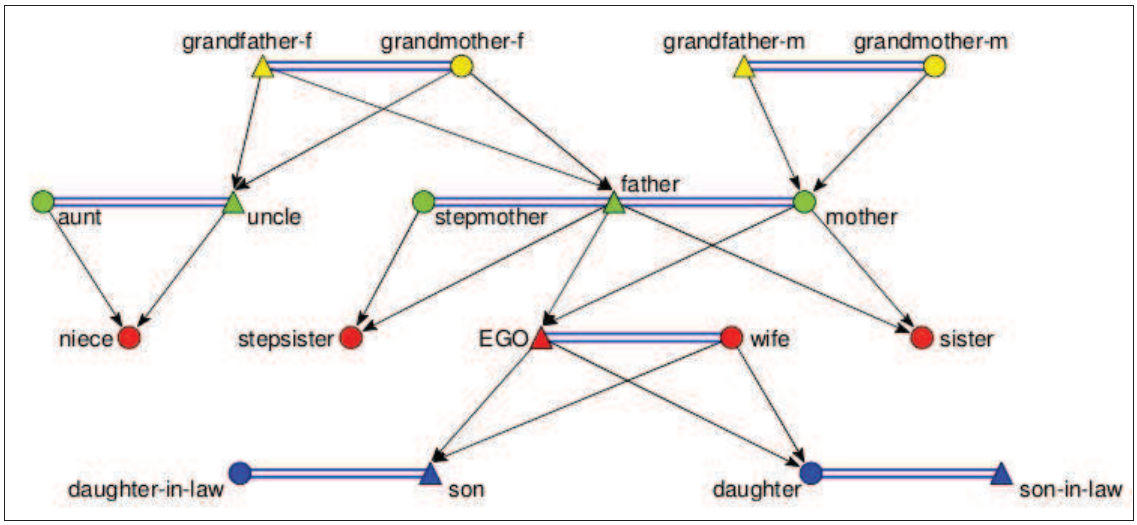
arcos representam laços filiais, e arestas representam casamentos.

Neste sociograma, homens são representados por triângulos, mulheres por elipses, casamentos por linhas (duplas), e relações pai-filho, por arcos. Em contraste com árvores genealógicas, pais *e* mães estão conectados com seus filhos em um *Ore Graph* (NOOY, MRVAR, BATAGELJ, 2011).

Uma das limitações de um *Ore Graph* é a ausência de conexões entre relações de irmandade. Ou seja, seria necessário retroceder até os pais de um indivíduo para identificar outros vértices originados a partir deles, e que representam os seus irmãos. Ou, de maneira semelhante, retroceder até os seus avós (paternos e maternos), obter os filhos destes e identificar, entre eles, quais são os seus tios. Além de dificultar a identificação visual dos vértices pertencentes a uma mesma geração, estes passos adicionais degradariam a eficiência de algoritmos de busca.

As Figuras 4 e 5 exemplificam este grafo genealógico. Em ambos o vértice "father" é representado com dois relacionamentos conjugais: "stepmother" e "mother". Os vértices são nomeados tendo EGO[2] como ponto de referência.

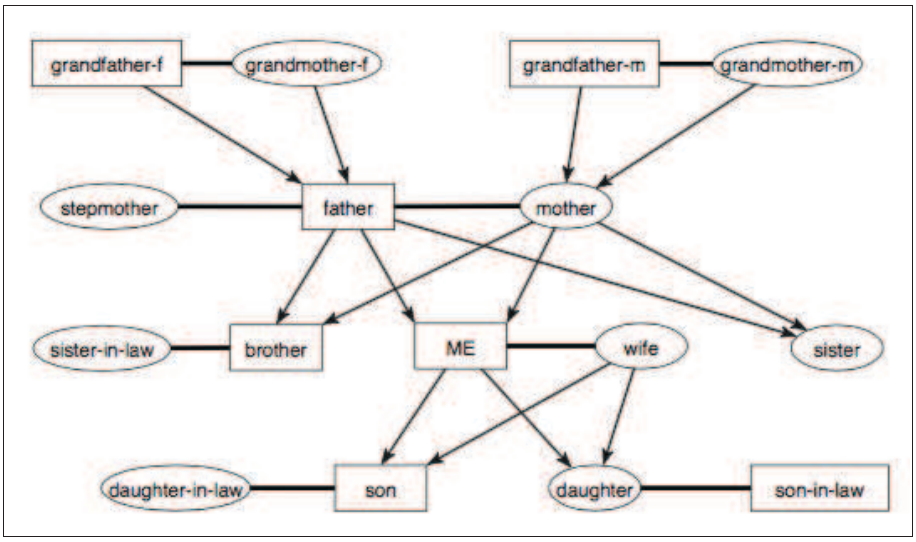
Figura 4 – Exemplo de um *Ore Graph*, onde o relacionamento entre "father" e "stepmother" gerou a descendente "stepsister"



Fonte: Extraído de NOOY, MRVAR, BATAGELJ, 2011

2 Pessoa em particular, na genealogia baseada em termos, a partir da qual os outros relacionamentos são relativos (FISCHER, 2017)

Figura 5 – Outro exemplo de *Ore Graph*, onde o relacionamento entre “father” e “stepmother” não gerou descendentes



Fonte: Extraído de MRVAR, BATAGELJ, 2004

Na Figura 4 pode-se constatar que o relacionamento entre “father” e “mother” gerou, além de EGO e “sister”, também o vértice identificado como “brother”.

b) *P-Graphs*

Em um *P-Graph* (do inglês *Parentage Graph*), casais *e* indivíduos solteiros são vértices, e arcos (que podem ser nomeados) apontam dos filhos para os pais. O tipo de arco demonstra se o descendente é masculino (arco cheio), ou feminino (arco tracejado) (NOOY, MRVAR, BATAGELJ, 2011).

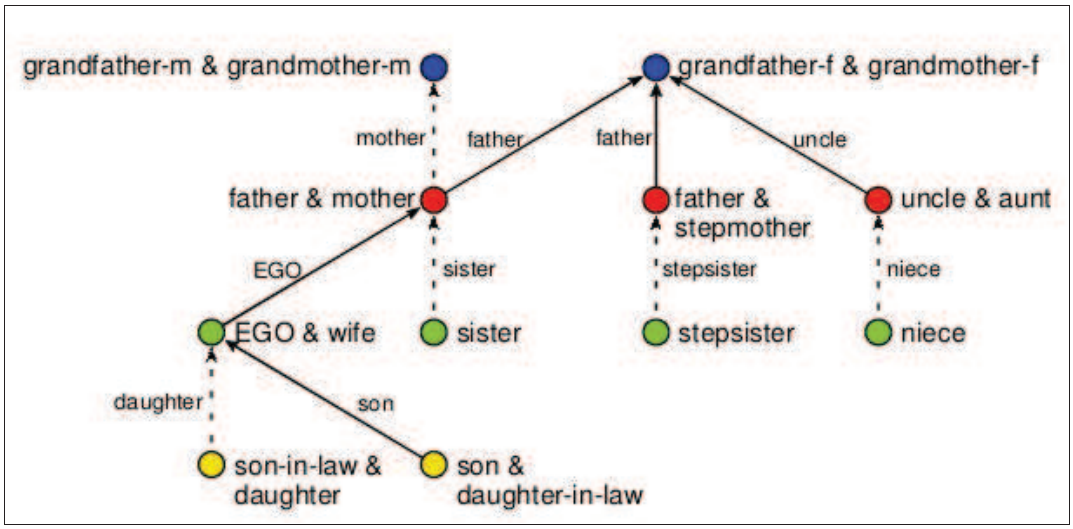
Nestes grafos a distinção gráfica ocorre entre os gêneros dos filhos, caracterizados pelo tipo de arco que os unem aos seus pais, sendo que todos os vértices são representados utilizando-se o mesmo símbolo. E uma vez que um relacionamento conjugal é estabelecido, ambos indivíduos passam a ser representados em um único vértice.

Neste tipo de grafo, cada pessoa é representada por um [único] arco, exceto no caso de um novo casamento. Como cada casamento é um vértice separado, homens e mulheres que casam novamente são representados por dois ou mais

arcos (NOOY, MRVAR, BATAGELJ, 2011).

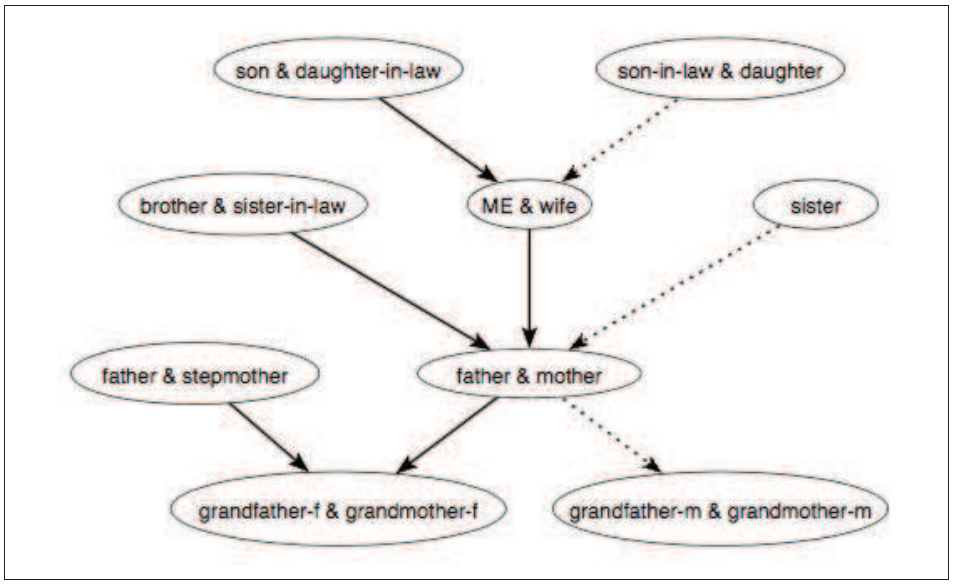
As Figuras 6, 7 e 8 apresentam exemplos de *P-Graphs* onde pode-se evidenciar um caso de casamento múltiplo (“father & mother” e “father & stepmother”). Na Figura 6 os ancestrais estão dispostos no sentido de cima para baixo, com este padrão invertido nas Figuras 7 e 8.

Figura 6 – Exemplo de um *P-Graph*, com arcos nomeados, onde o relacionamento “father & stepmother” gerou a descendente “stepsister”



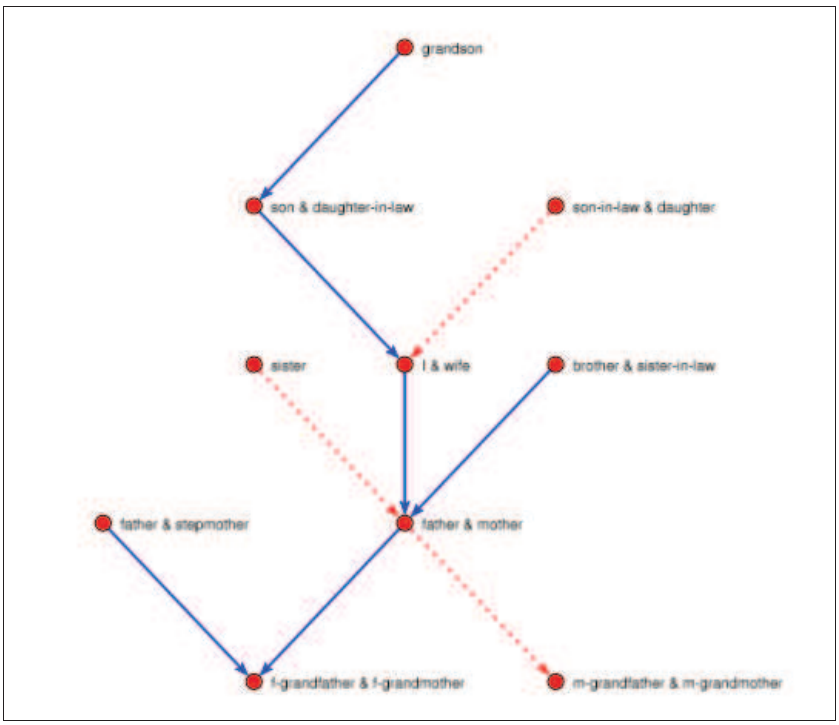
Fonte: Extraído de NOOY, MRVAR, BATAGELJ, 2011

Figura 7 – Outro exemplo de *P-Graph*, com arcos não nomeados, onde o relacionamento “father & stepmother” não gerou descendentes



Fonte: Extraído de MRVAR, BATAGELJ, 2004

Figura 8 – Um *P-Graph* com arcos não nomeados, com o descendente “grandson” no topo



Fonte: Extraído de BATAGELJ, MRVAR, 2008

P-Graphs representam casais e filhos não casados como vértices, ao passo que elos entre pai-filho são os arcos conectando vértices, tanto dentro como entre diferentes famílias nucleares (WHITE, BATAGELJ, MRVAR, 1999).

Mrvar e Batagelj (2004) apontam que, em comparação com *Ore Graphs*, uma das vantagens deste tipo de grafo é a existência de menos vértices e linhas. Outro benefício é a ausência de linhas cruzadas. Entretanto, em casos de múltiplos casamentos, o mesmo indivíduo é replicado em mais de um vértice e/ou arco.

Em um *P-Graph*, é impossível distinguir entre um tio casado e um novo casamento de um pai, ou entre meia-irmãs e sobrinhas (MRVAR, BATAGELJ, 2004). White (2004), por sua vez, aponta que *P-Graphs* não distinguem meio-irmãos porque, quando o mesmo pai está em dois casamentos diferentes, o relacionamento de meia-irmandade assemelha-se com aquele de primos (WHITE, 2004).

A impossibilidade de distinção de graus de parentesco em um *P-Graph* ocorre quando múltiplos casamentos são verificados. Podemos encontrar esta situação ilustrada na Figura 7 onde, considerando-se que os vértices não sejam nomeados, o relacionamento ”father & stepmother” poderia bem ser o de um “uncle & aunt”, já que neste caso pertenceriam a uma mesma geração (representada pelos vértices em um mesmo plano horizontal).

Da mesma forma que em *Ore Graphs*, White e Jorion (1996) ainda lembram que conexões extras não-direcionais entre irmãos não são introduzidas.

White, Batagelj e Mrvar (1999), por sua vez, lembram que neste tipo de grafo as relações são entre famílias, mas ressalvam que a convenção de direcionamento dos arcos de filho para pai não é tão intuitiva.

c) *Bipartite P-Graphs*

Bipartite P-Graphs são redes bimodais onde indivíduos e casais são representados por vértices. Portanto não há linhas de casamentos, mas existem arcos que apontam de indivíduos para casais, e de casais para indivíduos. Devido

ao fato de casamentos serem representados por vértices, é possível representar relações de irmandade mesmo se os pais dos irmãos são desconhecidos. Ademais, casamentos podem ser facilmente particionados, por exemplo, de acordo com suas datas (HAMBERGER, HOUSEMAN, WHITE, 2011).

Um *Bipartite P-Graph* possui dois tipos de vértices – vértices representando casais (retângulos) e vértices representando indivíduos (círculos para mulheres e triângulos para homens) – portanto cada cônjuge está envolvido em dois tipos de vértices (ou até mais se ele/ela está envolvido em múltiplos casamentos). Arcos novamente apontam dos filhos para seus pais (MRVAR, BATAGELJ, 2004).

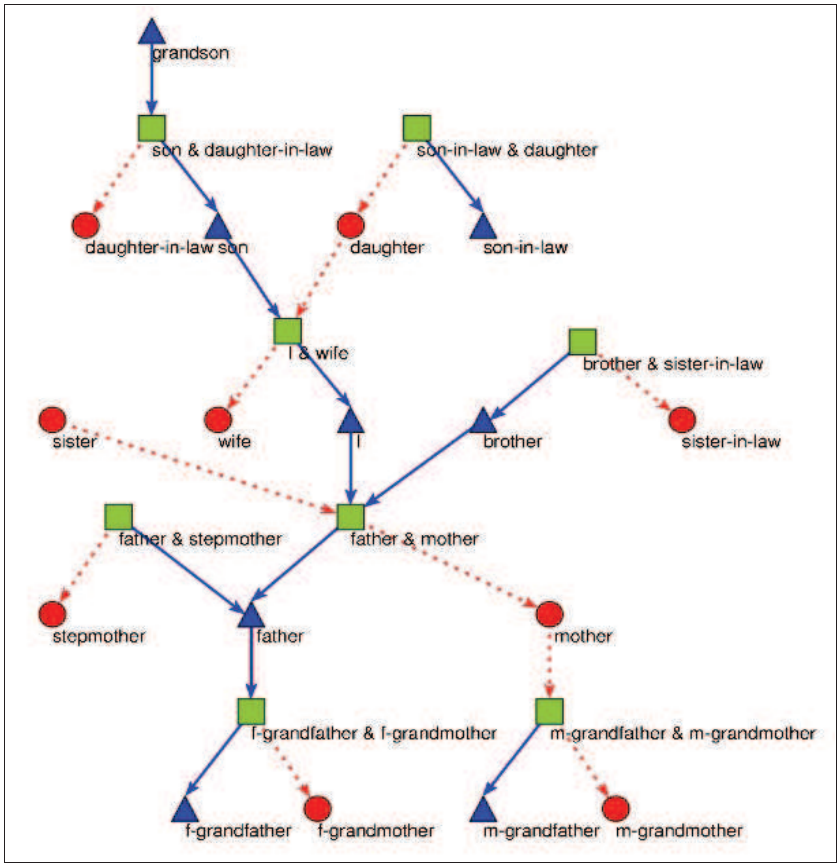
Bipartite P-Graphs apresentam uma vantagem adicional: podemos distinguir entre um tio casado e um outro casamento de um pai. Esta propriedade nos permite, por exemplo, encontrar casamentos entre meio-irmãos e meia-irmãs (MRVAR, BATAGELJ, 2004).

White (2004) aponta que meia-irmandades são distinguidas no formato *Bipartite P-Graphs*, no qual indivíduos correspondem a um conjunto de vértices, e casais, a outro.

Todavia, *Bipartite P-Graphs* tem como desvantagem a quantidade consideravelmente maior de vértices e linhas, comparados aos *P-Graphs.* Além disso, as distâncias dos percursos nestes grafos não correspondem aos graus de parentesco (NOOY, MRVAR, BATAGELJ, 2011).

A Figura 9 exemplifica um *Bipartite P-Graph*.

Figura 9 – Exemplo de um *Bipartite P-Graph*



Fonte: Extraído de BATAGELJ, MRVAR, 2008

Os componentes extras que existem nos *Bipartite P-Graphs* prejudicam a compreensibilidade e legibilidade da representação, na medida em que aumenta o número de matrimônios.

Cabe aqui destacar as diferentes convenções adotadas nas representações visuais dos três tipos de grafos genealógicos, apresentados nas figuras utilizadas como exemplo nesta seção.

Verificou-se que nos *Ore Graphs* as arestas e arcos não são nomeados, o mesmo ocorrendo para os arcos nos *Bipartite P-Graphs*. Já no caso dos *P-Graphs* a nomeação dos arcos é opcional.

Como *P-Graphs* e *Bipartite P-Graphs* não possuem arestas (apenas arcos

unidirecionais), conceitualmente poderíamos classificá-los como dígrafos. Conforme a definição de Chartrand, Lesniak e Zhang (2010), "um grafo direcionado ou dígrafo D é um conjunto finito e não-vazio de objetos chamados vértices, juntamente com um conjunto (possivelmente vazio) de pares ordenados de vértices distintos de D chamados arcos ou arestas direcionadas".

A simbologia utilizada para a distinção dos vértices por gênero varia conforme o tipo de grafo. Nos *Ore Graphs* observou-se que uma figura geométrica diferente é utilizada conforme o sexo do indivíduo. Nos *P-Graphs* e nos *Bipartite P-Graphs* o tipo de arco (cheio ou tracejado) identifica o sexo no vértice de origem. Já nos *Bipartite P-Graphs*, especificamente, duas figuras distintas (círculos e triângulos) identificam o sexo, e quadrados replicam os indivíduos casados. Todavia, constatou-se que a utilização de cores distintas serve tanto ao propósito de representar diferentes gerações, como também o gênero.

O Quadro 1, comparativo, sintetiza aspectos positivos e negativos dos três grafos apresentados.

Quadro 1 – Comparação entre os grafos genealógicos

Grafo	Aspectos positivos	Aspectos negativos
Ore Graph	Conexões diretas entre pais e filhos; Não ocorre replicação de indivíduos em vértices, para casos de múltiplos casamentos.	Ausência de conexões diretas entre irmãos; Linhas cruzadas.
P-Graph	Conexões diretas entre pais e filhos; Menor quantidade de vértices e linhas; Ausência de linhas cruzadas.	Ausência de conexões diretas entre irmãos; Replicação de indivíduos em vértices e arestas, para casos de múltiplos casamentos.
Bipartite P-Graph	Conexões diretas entre pais e filhos; Ausência de linhas cruzadas.	Ausência de conexões diretas entre irmãos; Replicação de indivíduos em vértices, para casos de múltiplos casamentos; Maior quantidade de vértices e linhas.

Fonte: Elaborado pelo autor

3 MÉTODOS

Em linhas gerais, a abordagem metodológica do presente estudo é considerada qualitativa, de cunho eminentemente exploratório e teórico-conceitual, já que se baseia primariamente em revisões bibliográficas de livros e artigos.

Para a construção dos artefatos propostos nos objetivos específicos, o método *Design Science* foi o escolhido.

3.1 Teórico-Conceitual

O referencial teórico foi obtido pelo método bibliométrico, que assegurou o acesso tanto às obras seminais, como às publicações mais recentes no tema, e que indicaram correspondência direta com os assuntos tratados neste estudo. A bibliografia resultante da aplicação deste método proveu os pilares nos quais se sustenta toda a conceituação utilizada e que, em última análise, se configura no próprio arcabouço para se atingir o objetivo principal proposto.

A bibliometria é um método largamente utilizado em pesquisas acadêmicas para o levantamento, seleção, coleta, armazenamento, e análise (referencial, de citações e estatística) de publicações científicas (livros, periódicos, artigos, anais de congressos, etc.).

No método bibliométrico, descritores (termos) são utilizados a fim de se restringir os resultados de buscas. Eles podem ser combinados de diversas formas, aumentando ainda mais a eficiência do processo de filtragem. Os descritores utilizados na pesquisa deste estudo estão relacionados no Quadro 2.

Quadro 2 – Descritores utilizados na pesquisa

Social Network Analysis
Graph Theory
Trees
Genealogy
Kinship Network Analysis
Ore Graph
P-Graph

Fonte: Elaborado pelo autor

Optou-se pela adoção de uma ferramenta para automatizar e facilitar o processo de bibliometria. Foi selecionado o *software* gerenciador de referências bibliográficas “JabRef” (JABREF, 2018), e sua escolha respeitou os critérios de custo (código aberto), portabilidade (capacidade de execução em distintas plataformas), facilidade de uso, e bases de conhecimento disponibilizadas para consulta.

O Quadro 3 relaciona as bases de conhecimento (“databases”), disponíveis para a opção “Web search”.

Quadro 3 – “Databases” disponíveis para consulta no *software* JabRef

ACM Portal
ArXiv
DBLP
DOAJ
GVK
Google Scholar
IEEEXplore
INSPIRE
Medline
Springer

Fonte: Elaborado pelo autor

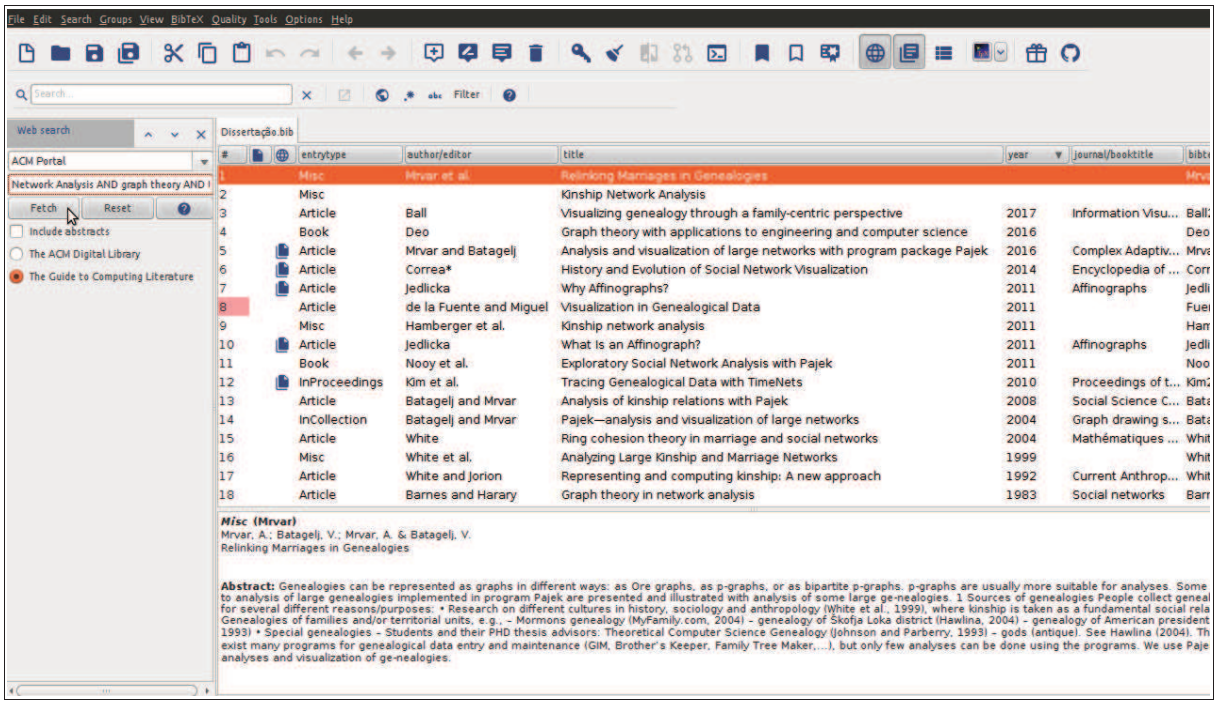
Visando maximizar os resultados relevantes para a pesquisa, os descritores foram combinados de todas as formas com os operadores “or” e “and”, na formação das *strings* de busca. Não houve critério de seleção quanto à data de publicação dos resultados, quanto à nacionalidade dos autores, ou ao veículo de publicação dos trabalhos.

Como critérios de inclusão, foi determinado que somente obras de língua inglesa seriam consideradas e que, em seu conteúdo, genealogias – ou relações de parentesco – fossem tratadas, em maior ou menor grau, no âmbito de redes sociais e/ou da Teoria dos Grafos.

Uma vez finalizada a pesquisa inicial, procedeu-se a uma triagem dos resultados, baseando-se no ano de publicação, referências (citação, cocitação), conteúdo do título e do resumo, a fim de se obter um inventário representativo das publicações mais significativas. Ao final 18 entradas foram selecionadas, além de servirem, elas mesmas, como fonte de referência complementar (livros, dissertações e teses).

A Figura 10 exibe o conteúdo do arquivo com extensão “bib”, aberto no *software*, e criado para armazenar as entradas para manipulação (resultados selecionados nas buscas).

Figura 10 – Arquivo “bib” no *software* JabRef, contendo as 18 entradas selecionadas, classificadas em ordem decrescente de ano



Fonte: JabRef

As entradas seguem o formato “BibTeX”, e o seu conteúdo textual está transcrito no Quadro 4.

Quadro 4 – Conteúdo textual das entradas armazenadas no formato “BibTeX”

Tipo de Entrada	Autor/Editor	Título	Ano	Título do Periódico/Livro
Misc	Mrvar, Andrej and Batagelj, Vladimir	Relinking Marriages in Genealogies	-	-
Misc		Kinship Network Analysis	-	-
Article	Ball, Robert	Visualizing genealogy through a family-centric perspective	2017	Information Visualization
Book	Deo, Narsingh	Graph theory with applications to engineering and computer science	2016	-
Article	Mrvar, Andrej and Batagelj, Vladimir	Analysis and visualization of large networks with program package Pajek	2016	Complex Adaptive Systems Modeling
Article	Correa, Carlos D.	History and Evolution of Social Network Visualization	2014	Encyclopedia of Social Network Analysis and Mining
Article	Jedlicka, Davor	Why Affinographs?	2011	Affinographs
Article	de la Fuente, Jesus Miguel	Visualization in Genealogical Data	2011	-
Misc	Hamberger, Klaus and Houseman, Michael and Douglas, R White	Kinship network analysis	2011	-
Article	Jedlicka, Davor	What Is an Affinograph?	2011	Affinographs
Book	Nooy, Wouter De and Mrvar, Andrej and Batagelj, Vladimir	Exploratory Social Network Analysis with Pajek	2011	-
InProceedings	Kim, Nam Wook and Card, Stuart K. and Heer, Jeffrey	Tracing Genealogical Data with TimeNets	2010	Proceedings of the International Conference on Advanced Visual Interfaces
Article	Batagelj, Vladimir and Mrvar, Andrej	Analysis of kinship relations with Pajek	2008	Social Science Computer Review
InCollection	Batagelj, Vladimir and Mrvar, Andrej	Pajek—analysis and visualization of large networks	2004	Graph drawing software
Article	White, Douglas R	Ring cohesion theory in marriage and social networks	2004	Mathématiques et sciences humaines
Misc	White, Douglas R. and Batagelj, Vladimir and Mrvar, Andrej	Analyzing Large Kinship and Marriage Networks	1999	-
Article	White, Douglas R and Jorion, Paul	Representing and computing kinship: A new approach	1992	Current Anthropology
Article	Barnes, John A and Harary, Frank	Graph theory in network analysis	1983	Social networks

Fonte: Elaborado pelo autor

O inventário resultante (conjunto das entradas) subsidiou as referências bibliográficas utilizadas na Fundamentação Teórica (Capítulo 2) deste estudo.

3.2 *Design Science*

Como parte deste estudo se dedica à construção de artefatos, optou-se pela adoção do método *Design Science* para se atingir os objetivos específicos propostos. Segundo Lacerda *et al.* (2013) o desenvolvimento, como o processo de construção de um artefato, pode utilizar diferentes abordagens como algoritmos computacionais, representações gráficas e protótipos.

Design Science, como o outro lado do ciclo de pesquisa em Sistemas de Informação, cria e avalia artefatos de Tecnologia da Informação (TI). Uma base matemática para o projeto permite vários tipos de avaliações quantitativas de um artefato de TI, incluindo provas de otimização, simulação analítica, e comparações quantitativas em projetos alternativos. *Design science* é inerentemente um processo de solução de problemas. É ativa com respeito à tecnologia, se empenhando na criação de artefatos tecnológicos que afetam pessoas e organizações (HEVNER, MARCH, PARK, 2004).

Uma vez definida a nova representação de genealogias na Teoria dos Grafos (Capítulo 4), procedeu-se à sua implementação em ambiente computacional, com o propósito de contribuir com a Análise de Redes Sociais, no que concerne ao estudo de relações de parentesco em Sistemas de Informação. Para se atingir este objetivo, as diretrizes do método mencionado guiaram a geração de artefatos na forma de componentes de *software*, constantes dos Apêndices A e B.

Estes componentes de *software* estão na forma de código, em uma linguagem de programação orientada a objetos. Estão organizados em classes (Apêndice A) que modelam a estrutura e a aparência de um *K-Graph*, e utilizam, portanto, uma nomenclatura específica da Genealogia. Já os algoritmos de busca (Apêndice B), embora dispostos nos métodos destas classes, foram segregados para demonstrar as opções de busca em um *K-Graph*, explicitando a lógica

envolvida na manipulação de indivíduos e relacionamentos.

Estendendo as contribuições deste estudo, a Seção 4.2.1 detalha a criação de um protótipo, na forma de um programa (sistema), visando demonstrar a eficácia do novo grafo quando aplicado em outra área de conhecimento.

Hevner, March e Park (2004) descrevem um sistema prototípico como um exemplo de um artefato onde se aplicam métodos de avaliação (experimental, por exemplo), e que pode estender a base de conhecimento ou aplicar conhecimento existente de maneira inovativa.

O protótipo foi desenvolvido visando aplicar a estrutura de um *K-Graph* na solução de um problema de desmembramento de localidades geográficas, representadas na Teoria dos Grafos.

4 RESULTADOS E ANÁLISE

Este capítulo demonstra os resultados do estudo na forma de proposição de um novo grafo genealógico. Em seguida se analisam possíveis aplicações do novo grafo para representação de redes de relacionamento em outras áreas. Se propõe, ainda, a criação de um artefato (protótipo de programa) para comprovar a eficácia da nova representação, aplicando-a em uma rede de localidades geográficas.

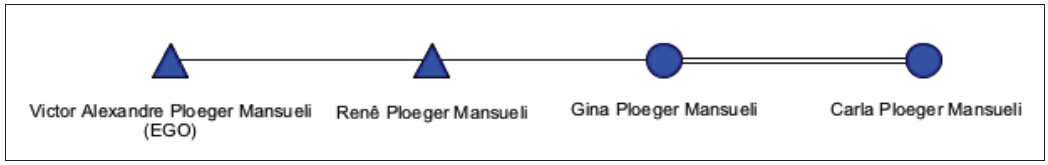
4.1 Uma Nova Representação: *K-Graphs*

Grafos de Parentesco ou *K-Graphs* (K do inglês *Kinship*) são uma nova proposta representacional de genealogias, que se baseia na disposição dos componentes estruturais dos grafos existentes, complementando-os ao agregar as principais vantagens de cada representação. Visou-se suprimir, desta mesma forma, as limitações observadas quando utilizados isoladamente.

Este novo sociograma foi concebido como uma variação dos *Ore Graphs*, adotado como referência por não apresentar replicação de indivíduos (característica presente nos *P-Graphs* e *Bipartite P-Graphs*). Os vértices em um *K-Graph* representam apenas um indivíduo – mesmo em casos de múltiplos casamentos – e este vértice não se repete na estrutura.

O principal aspecto que diferencia um *K-Graph* dos três grafos genealógicos aqui expostos são as relações de irmandade. Estas são estabelecidas através de conexões diretas entre irmãos, com arestas originando-se a partir do primogênito de um casal. Desta maneira uma estirpe é traçada de forma contínua e linear, com classificação temporal (pela data de nascimento), composta por vértices de uma mesma geração (Figura 11).

Figura 11 – Exemplo de uma linha de descendência (estirpe) em um *K-Graph*, com 4 vértices, dispostos em ordem cronológica pela data de nascimento



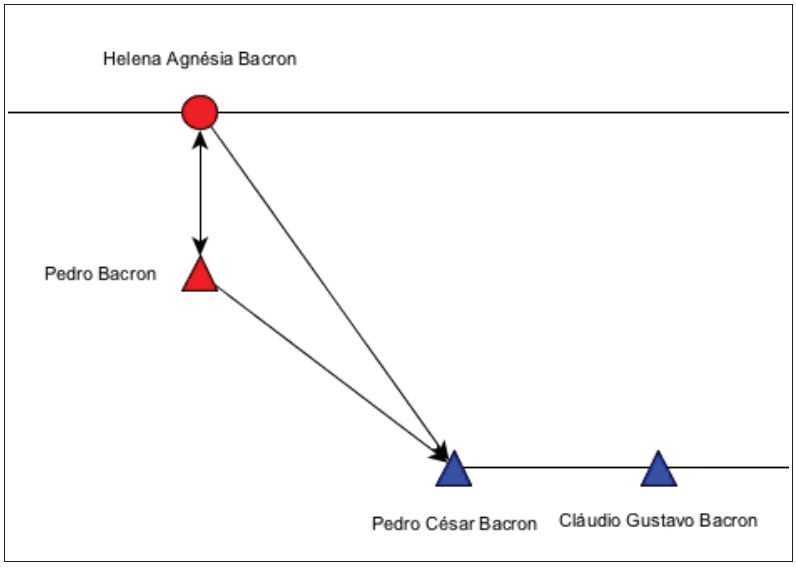
Fonte: Elaborado pelo autor

4.1.1 Vantagens e Características

Esta seção explora as particularidades dos *K-Graphs*, em contraposição aos aspectos negativos descritos no Quadro 1 (Capítulo 2), destacando-se suas vantagens e características.

Embora baseada nos *Ore Graphs*, a nova representação notabiliza-se pela ausência de cruzamentos de arcos, o que ocorre no grafo genealógico tradicional quando existe mais de um filho por casal (em um *K-Graph* apenas o primeiro filho é conectado aos seus pais, por meio de um arco direcional partindo de cada um deles). A Figura 12 ilustra esta vantagem.

Figura 12 – Exemplo de filiação em um *K-Graph*: apenas o primeiro vértice (primogênito) é conectado aos seus progenitores, por meio de dois arcos direcionais (de pais para filho)

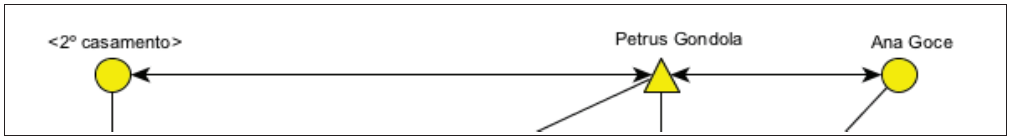


Fonte: Elaborado pelo autor

Além de prever múltiplos casamentos (através da associação, com um arco bidirecional, de um novo vértice de gênero oposto com um vértice existente), *K-Graphs* não estão sujeitos à análises equivocadas de parentesco, pois a unicidade dos vértices é garantida (indivíduos não são replicados).

Para identificar uma nova relação conjugal, um único arco bidirecional é conectado entre dois vértices, não sendo necessária a criação de um novo vértice para representar um casal, como ocorre nos *P-Graphs* e nos *Bipartite P-Graphs*. A Figura 13 exemplifica este cenário.

Figura 13 – Exemplo de múltiplo casamento em um *K-Graph*: vértices conectados por meio de um arco bidirecional



Fonte: Elaborado pelo autor

4.1.2 Convenções

K-Graphs utilizam uma topologia própria para identificar os relacionamentos em uma rede de parentesco. As diferentes arestas configuram este tipo de grafo como híbrido, contendo linhas, linhas duplas, e arcos (uni e bidirecionais).

Cores discriminam vértices de uma mesma geração, e o uso de figuras geométricas distintas determinam o gênero. Abaixo elencam-se as simbologias e convenções que distinguem os *K-Graphs* dos grafos tradicionais:

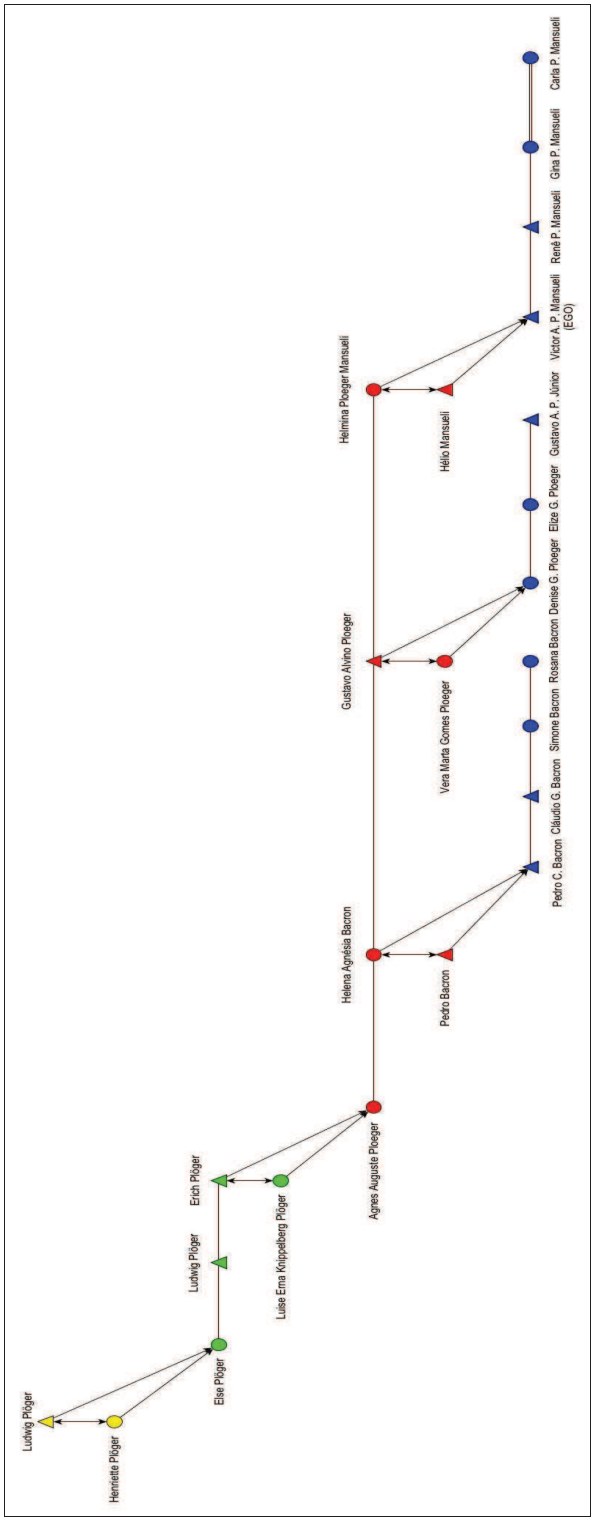
- Disposição espacial: ascendentes acima, descendentes abaixo; componentes dispostos do centro para a direita, a partir dos vértices dos antepassados mais antigos (“raízes”);
- Gêneros: discriminados pelas figuras geométricas dos vértices (triângulos cheios para masculino, círculos cheios para feminino);
- Nomeações: apenas os vértices são identificados com nomes de indivíduos (arestas não são nomeadas);

- Gerações: identificadas pelas cores dos vértices;
- Relacionamentos conjugais (pares de vértices de casais): identificados por arcos bidirecionais;
- Filiações: primogênitos conectados a seus pais por dois arcos direcionais (de pais para filho);
- Irmandades: irmãos conectados entre si por linhas (primogênitos e caçulas com apenas um vértice adjacente; irmãos “do meio” conectados por dois vértices adjacentes);
- Gemelaridades: gêmeos conectados entre si por linhas duplas;
- Linearidade: irmãos representados horizontalmente;
- Temporalidade: vértices que identificam irmãos são dispostos conforme a ordem cronológica das datas de nascimento.

4.1.3 Representação Gráfica

A Figura 14 ilustra um *K-Graph* composto pela ascendência materna do autor deste trabalho (identificado pelo termo EGO no sociograma), limitada a quatro gerações (bisavós, avós, pais e filhos). Os componentes foram dispostos do centro para a direita. Para a composição visual foi utilizado o *software* de edição de grafos “yEd”, versão 3.17 (YED, 2018).

Figura 14 – Exemplo de um *K-Graph*



Fonte: Elaborado pelo autor

Na Figura 14 podemos evidenciar informações da estrutura familiar representada. Estas informações, por sua vez, permitem inferir relações de parentesco. Alguns exemplos:

- 4 gerações foram representadas, em um total de 24 indivíduos (11 masculinos e 13 femininos);
- Os bisavós maternos de EGO estão situados no topo do diagrama (vértices amarelos);
- O avô materno de EGO (caçula) possui 2 irmãos (a irmã primogênita e o irmão “do meio”);
- De seu lado materno, EGO possui 3 tios (2 tias e 1 tio) e 7 primos (4 primas e 3 primos);
- EGO é primogênito e possui 3 irmãos (1 irmão e 2 irmãs gêmeas);
- Das 3 primeiras gerações (mais antigas), 3 indivíduos não tiveram descendentes (Else Plöger, Ludwig Plöger (filho) e Agnes Auguste Ploeger).

Ao analisar a representação de grafos genealógicos na estrutura proposta pelos *K-Graphs*, conclui-se que esta usufruiu dos benefícios existentes nas representações tradicionais. Ao mesmo tempo, unificou a existência de vértices e extinguiu o cruzamento de arcos, resultando em um formato com maior legibilidade.

Foi demonstrado que as três formalizações existentes (*Ore Graphs*, *P-Graphs* e *Bipartite P-Graphs*) possuem algumas limitações quando tratadas isoladamente, e estas foram supridas pela nova proposta: relações de irmandade foram estabelecidas através de arestas (o que também possibilitou a supressão dos cruzamentos de arcos observados nos *Ore Graphs*); a possibilidade de representação de múltiplos casamentos foi prevista, e; uma correspondência única entre indivíduos e vértices foi assumida.

K-Graphs ainda poderiam representar relações não tradicionais de parentesco através de outras simbologias. Adoções, por exemplo, poderiam ser identificadas por triângulos e círculos sem preenchimento (filhos e filhas, respectivamente).

Todavia, *K-Graphs* não estão restritos à representações genealógicas. A próxima seção explora possíveis aplicações, onde a representação do grafo proposto neste estudo poderia ser utilizada para analisar e descrever atores e laços

em modalidades diversas de associações em redes, contribuindo, desta forma, com outras áreas de pesquisa.

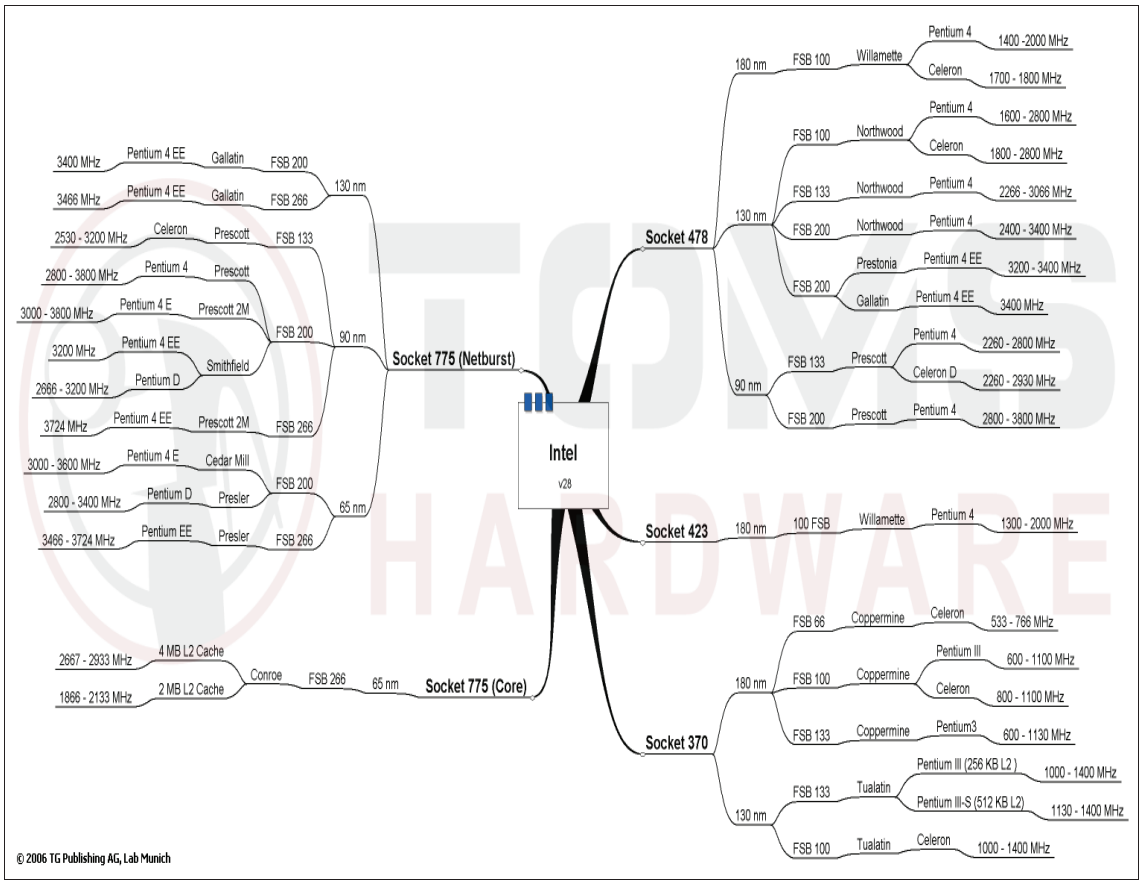
4.1.4 Outras Aplicações

Não obstante o foco da nova proposta representacional tenha sido o campo da Genealogia, *K-Graphs* podem ser aplicados em análises de redes diversas, condicionando-se estas a preceitos genealógicos como ascendência, descendência primordialidade e unicidade.

K-Graphs poderiam, por exemplo, ser aplicados na ilustração das relações em uma cadeia logística de fornecimento de materiais ou serviços (“Supply Chain”), assim como na representação de famílias (linhas e modelos) de produtos ao longo do tempo, onde estes possuam características únicas, e que tenham origem a partir da combinação de dois produtos anteriores.

A Figura 15 ilustra uma destas linhas de produtos, onde o novo grafo genealógico poderia ser aplicado como recurso representacional: desenvolvimento de CPUs da Intel® (do *Socket* 370 ao LGA 775).

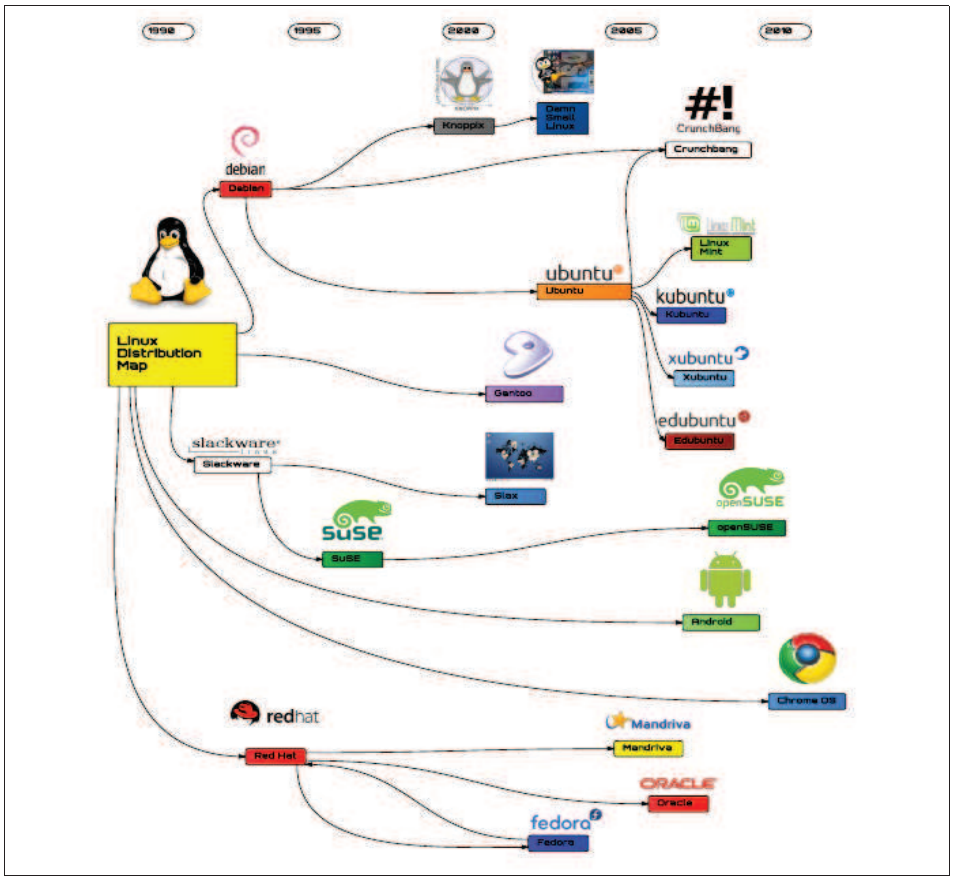
Figura 15 – Exemplo de uma família de produtos: processadores Intel®



Fonte: TOM'S HARDWARE (2018)

Outro possível uso para um *K-Graph* seria a representação de versões de Sistemas Operacionais, como distribuições da família GNU/Linux, onde cada uma delas se origina a partir de incrementos realizados em algum momento em uma distribuição prévia (Figura 16).

Figura 16 – Família das principais distribuições do Sistema Operacional Linux



Fonte: NETWORK ENGINEER (2018)

4.2 Prova de Conceito

Para ilustrar a aplicação de *K-Graphs* em uma outra área de conhecimento, demonstrando que são suscetíveis de serem explorados de outras formas úteis (além de representações genealógicas), esta seção apresenta uma aplicação para divisões geográficas.

Divisões geográficas assemelham-se a genealogias em suas estruturas hierárquicas. Elas compartilham de um “ancestral” em comum (continentes, países, por exemplo), e se desmembram em unidades menores, agrupadas para efeitos de organização e classificação política, administrativa, econômica, social, etc. Todas se originam em um determinado momento, a partir de subdivisões que se assemelham à composições familiares.

Poderíamos, para efeito de analogia, considerar as 5 Regiões nas quais se divide o Brasil (Centro-Oeste, Norte, Nordeste, Sudeste e Sul) como descendentes da junção do País com um determinado evento no tempo. Da mesma forma seria considerada a divisão das Regiões em Estados ao longo do Século XIX, com as últimas alterações ocorrendo na Constituição de 88 (27 Unidades Federativas, compostas de 26 Estados mais o Distrito Federal). E subsequentes desmembramentos em distritos, municípios, bairros e ruas.

Pesquisas foram realizadas para se obter uma base dados de localidades a nível nacional, sendo que vários estudos de geociências se encontram disponíveis no endereço eletrônico do Instituto Brasileiro de Geografia e Estatística (IBGE). Contudo, a sua maioria é composta por censos demográficos, e não foi possível localizar uma fonte de dados da estrutura territorial brasileira contendo informações temporais (data de fundação, por exemplo), sendo estas informações derivadas como atributos essenciais na organização dos vértices em um *K-Graph*. Igualmente, bases de dados dos Correios foram descartadas por apresentar uma alta granularidade e, como constatado no caso do IBGE, pela ausência de campos que representem datas.

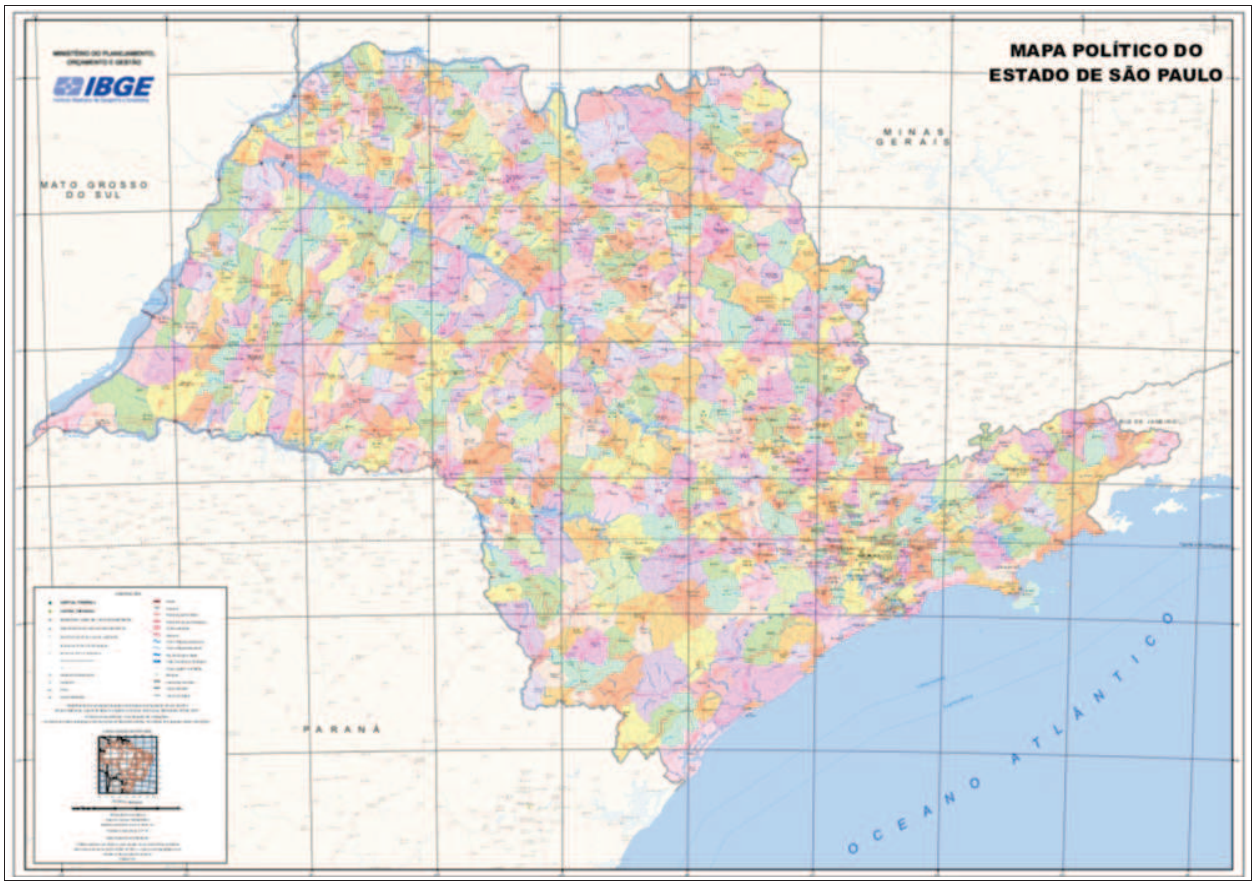
Optou-se, ainda, por limitar a amostragem de localidades ao Estado de São Paulo, para efeito de simulação da aplicabilidade da estrutura de um *K-Graph* em dados geográficos. Para tanto, procedeu-se a uma pesquisa no endereço eletrônico da Fundação Sistema Estadual de Análise de Dados (Seade).

A Seade, vinculada à Secretaria de Planejamento e Gestão do Estado de São Paulo, é um centro de referência nacional na produção e disseminação de análises e estatísticas socioeconômicas e demográficas. Para isso, realiza pesquisas diretas e levantamentos de informações produzidas por outras fontes, compondo um amplo acervo, disponibilizado gratuitamente, que permite a caracterização de diferentes aspectos da realidade socioeconômica do estado, de suas regiões e municípios e de sua evolução histórica (FUNDAÇÃO SEADE, 2018).

O Estado de São Paulo subdivide-se em 645 municípios, distribuídos em 42 regiões de governo, 14 regiões administrativas (RA) e três regiões metropolitanas: de São Paulo, da Baixada Santista (que tem a conformação espacial da RA de Santos) e de Campinas (contida na RA do mesmo nome). O conjunto de transformações socioeconômicas ocorridas nos últimos 50 anos no Estado foi acompanhado por um intenso processo de redistribuição da população, do que resultou uma concentração populacional regionalmente diferenciada (SEADE [1], 2018).

A Figura 17 foi extraída da página de mapas do IBGE, e ilustra a divisão político-administrativa do Estado de São Paulo (IBGE, 2018).

Figura 17 – Mapa Político de SP



Fonte: Extraído de IBGE (2018)

4.2.1 Protótipo de Programa

Como fonte de dados para o desenvolvimento do protótipo deste estudo foi utilizado um exemplo disponibilizado na Internet pela Seade, em 22/11/2016, em uma de suas páginas de “Press Releases” contendo informações do desmembramento dos 645 municípios paulistas (SEADE [2], 2018).

Os dados para amostra foram extraídos de um arquivo do tipo planilha eletrônica (tabdesmsp_dez2017.xlsx), integrante do código-fonte e dados para *download*[3] na página “Desmembramentos dos Municípios Paulistas”.

A estrutura e conteúdo da planilha se mostrou conveniente para os propósitos da aplicação, com uma amostragem de 657 ocorrências, informação temporal (ano de criação da localidade), assim como aderência ao preceito de descendência (desmembramento), vinculando as localidades com o seu respectivo “pai”. Desta forma, constatou-se que características propostas nas representações genealógicas dos *K-Graphs* puderam ser reproduzidas no contexto de localidades geográficas:

- Relacionamentos conjugais (junções): compostos por uma localidade e um desmembramento;
- Irmandades: localidade(s) originada(s) a partir de uma mesma localidade-pai;
- Gemelaridades: localidades-irmãs com o mesmo ano de criação;
- Temporalidade: vértices que identificam localidades-irmãs dispostos conforme a ordem cronológica do ano de criação.

As próximas seções estão organizadas de acordo com as etapas seguidas no processo de desenvolvimento do protótipo.

4.2.1.1 Banco de Dados

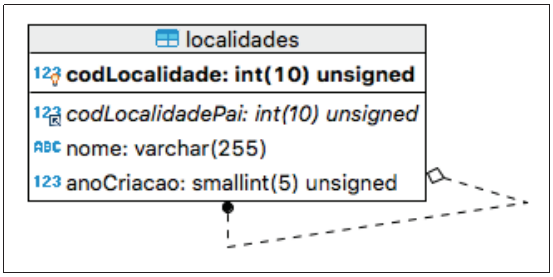
Para armazenar os dados da planilha foi utilizado o Banco de Dados *Open Source* “MySQL”, edição *Community Server*, versão 5.7.18.

3 *Link* disponível em http://www.seade.gov.br/visualizacao/desmembramentosp/desmembramentosp_dez2017.zip

A tabela “localidades” foi modelada e criada de forma a mapear a estrutura das colunas na planilha. Para possibilitar a vinculação de uma localidade com suas “filhas”, um auto-relacionamento foi definido no campo “codLocalidadePai”. Este auto-relacionamento foi configurado na forma de uma Chave Estrangeira (*Foreign Key* – FK), assegurando que apenas localidades-pai válidas seriam referenciadas.

A ferramenta utilizada para manipular o Banco de Dados foi o *software* DBeaver, versão 5.0.2 (DBEAVER, 2018). A Figura 18 exibe o Diagrama Entidade-Relacionamento (ER) da tabela.

Figura 18 – Diagrama ER da tabela “localidades”, com um auto-relacionamento na coluna “codLocalidadePai"



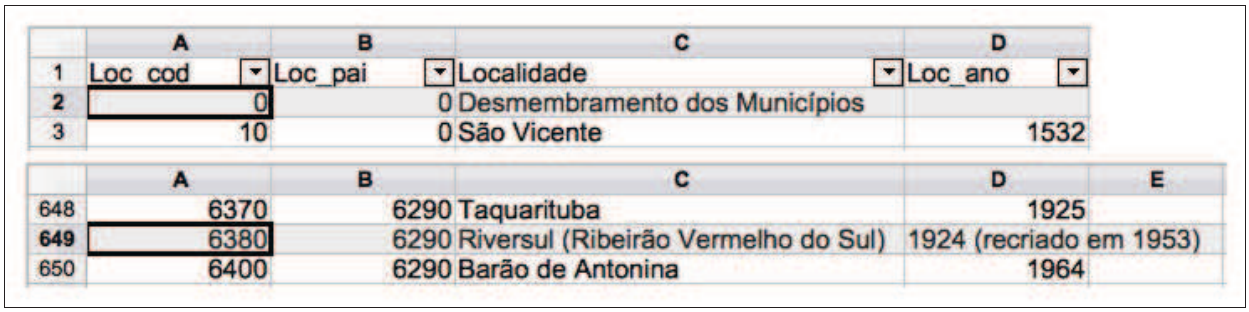
Fonte: Extraído de DBeaver (2018)

a) Normalização

A Normalização é uma atividade de modelagem de dados que visa garantir a integridade de um banco de dados, reduzindo a sua redundância e aumentando o seu desempenho.

Após a análise dos dados disponíveis na planilha, alguns ajustes se mostraram necessários antes de se proceder à carga da tabela. Duas linhas foram alteradas para a importação, e seu conteúdo original pode ser visto na Figura 19 (linhas destacadas em cinza).

Figura 19 – As linhas 2 e 649 da planilha, com os dados originais



	A	B	C	D
1	Loc_cod	Loc_pai	Localidade	Loc_ano
2	0	0	Desmembramento dos Municípios	
3	10	0	São Vicente	1532

	A	B	C	D	E
648	6370	6290	Taquarituba	1925	
649	6380	6290	Riversul (Ribeirão Vermelho do Sul)	1924 (recriado em 1953)	
650	6400	6290	Barão de Antonina	1964	

Fonte: Elaborado pelo autor a partir de SEADE [2], 2018

Três células foram atualizadas na linha 2:

1. Coluna “Loc_pai”: o valor “0” foi alterado para “NULL”;
2. Coluna “Localidade”: o valor “Desmembramento dos Municípios” foi alterado para “SP”;
3. Coluna “Loc_ano”: foi preenchido o valor “1532”.

Após esta atualização a primeira linha da planilha identificou a localidade “raiz”, representada pelo Estado de SP, com pais “nulos”, e com ano de criação igual ao ano de sua primeira localidade-filha (São Vicente, 1532). Este último ajuste de data fez-se necessário devido à obrigatoriedade da existência de valor na respectiva coluna da tabela.

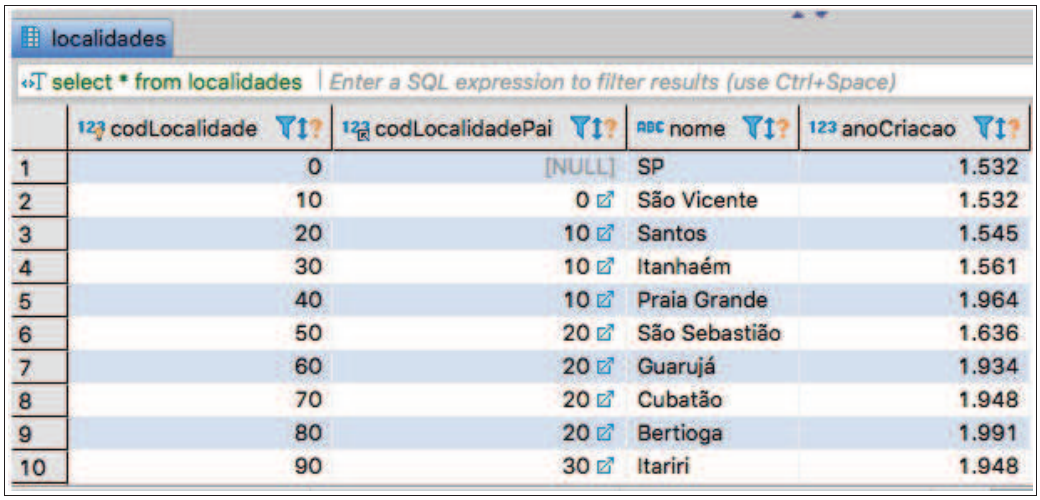
A linha 649 da planilha apresentava uma observação na coluna “Loc_ano”, invalidando o tipo de dado para o qual a coluna seria mapeada: “1924 (recriado em 1953)“. Neste caso o conteúdo foi alterado para “1953”. Esta linha se referia à localidade “Riversul (Ribeirão Vermelho do Sul)”.

Importante ainda salientar que não foi revisado o conteúdo da coluna “Localidade”, contendo os nomes das localidades. Foram mantidos, portanto, os casos encontrados em que informações adicionais constavam entre parênteses, como "(parte 1)" e "(extinto)", por exemplo. Por último, convencionou-se que a sigla “SP” seria utilizada para referenciar o Estado, enquanto “São Paulo” identificaria a cidade (capital).

b) Carga

A importação dos dados da planilha gerou 657 registros, ordenados pela coluna “codLocalidade”. A Figura 20 exibe o resultado da seleção das primeiras 10 linhas (a primeira linha representa a localidade-raiz, SP, a partir da qual surgiram todos os desmembramentos).

Figura 20 – Conteúdo da tabela “localidades”: 10 primeiros registros (ordenados pelo código da localidade)



localidades

select * from localidades | Enter a SQL expression to filter results (use Ctrl+Space)

	codLocalidade	codLocalidadePai	nome	anoCriacao
1	0	[NULL]	SP	1.532
2	10	0	São Vicente	1.532
3	20	10	Santos	1.545
4	30	10	Itanhaém	1.561
5	40	10	Praia Grande	1.964
6	50	20	São Sebastião	1.636
7	60	20	Guarujá	1.934
8	70	20	Cubatão	1.948
9	80	20	Bertioga	1.991
10	90	30	Itariri	1.948

Fonte: Extraído de DBeaver (2018)

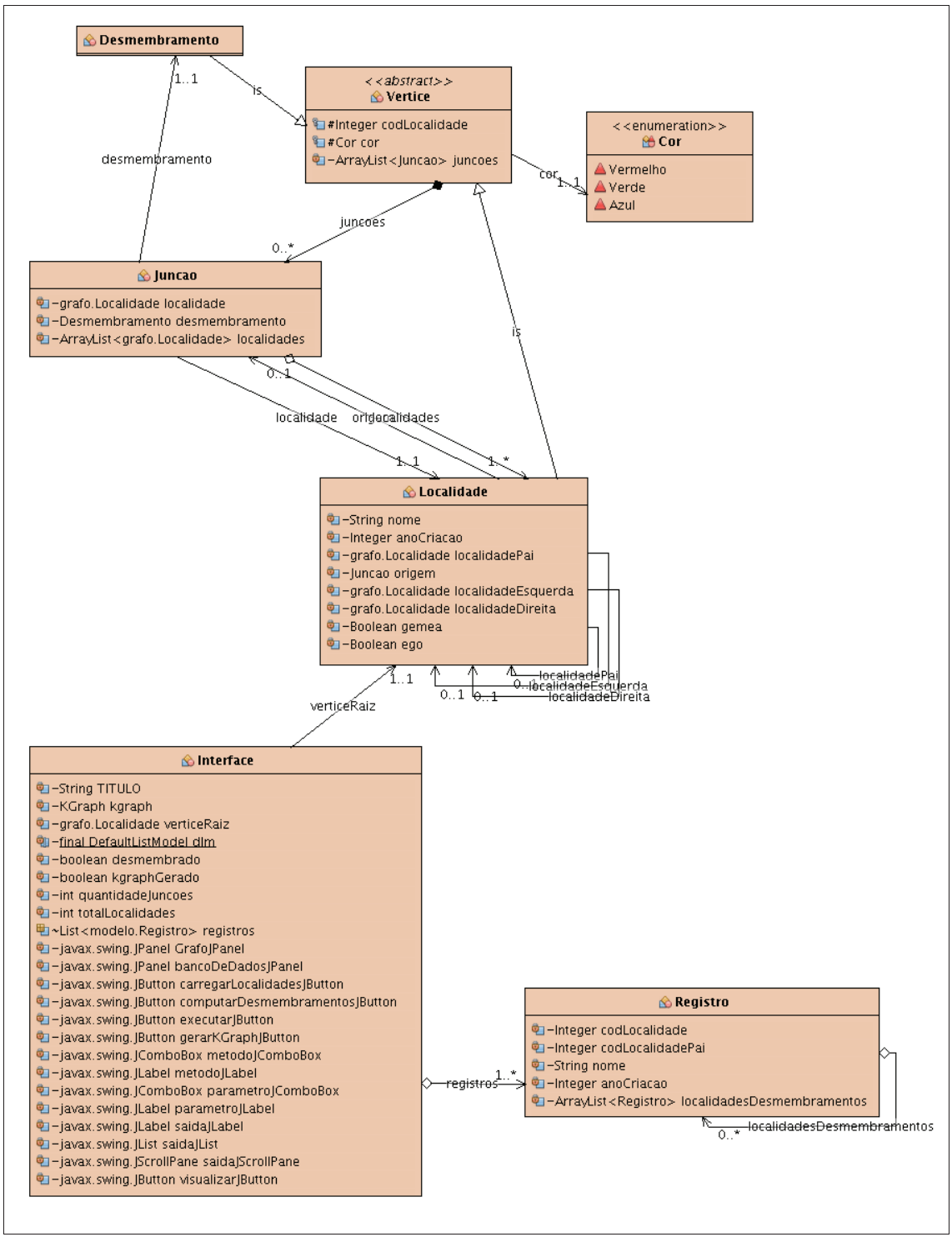
4.2.1.2 Interface Gráfica

Para implementar o protótipo utilizou-se como ferramenta de desenvolvimento de *software* a IDE (*Integrated Development Environment*) NetBeans, versão 8.0.2, com suporte à construção de aplicações do tipo “Desktop” por meio da biblioteca de componentes visuais “Swing”.

A Linguagem de Programação adotada foi Java (SE Runtime Environment (build 1.8.0_131-b11)). Entre outros critérios para a sua escolha, destacam-se o seu paradigma de orientação a objetos, e a grande variedade de tecnologias (“frameworks”, bibliotecas, *Application Programming Interfaces* (APIs)) disponíveis para integração com a Plataforma.

O Diagrama de Classes UML (*Unified Modeling Language*) do protótipo é exibido na Figura 21, contendo os atributos/propriedades e associações.

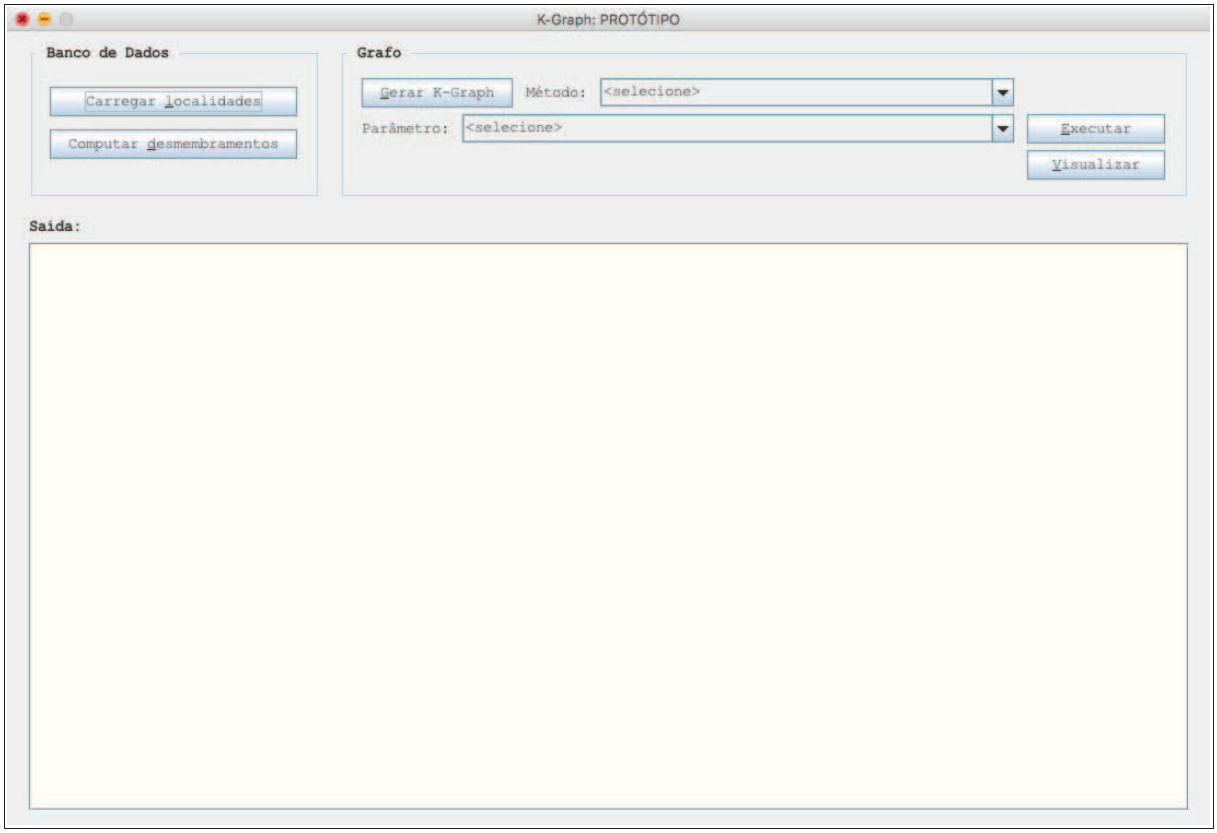
Figura 21 – Diagrama de Classes UML do protótipo



Fonte: Elaborado pelo autor

A Figura 22 exibe a tela principal (interface), com os componentes visuais organizados em dois grupos: “Banco de Dados” e “Grafo”. O componente “Saída” é reservado para exibição (“output”) do processamento dos botões “Carregar localidades”, “Computar desmembramentos”, “Gerar K-Graph” e “Executar”.

Figura 22 – Interface do protótipo



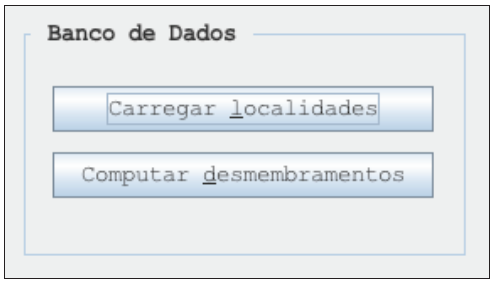
Fonte: Elaborado pelo autor

a) Grupo “Banco de Dados”

Para estabelecimento de comunicação com o Banco de Dados foi utilizada a API “JDBC” (*Java Database Connectivity*).

Este grupo é composto por dois botões (Figura 23), que mapeiam os registros do banco de dados para objetos em memória.

Figura 23 – Botões do grupo “Banco de Dados”

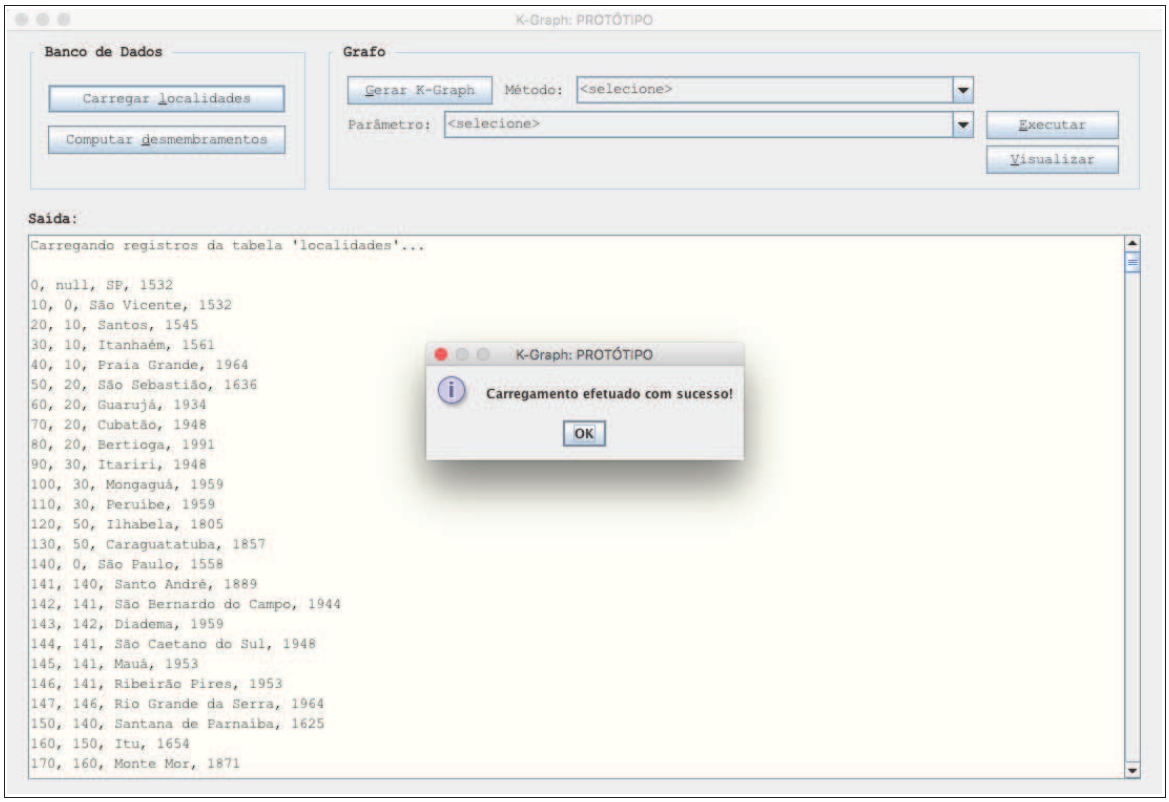


Fonte: Elaborado pelo autor

a.1) Botão **Carregar localidades**

Executa uma instrução SQL (*Structured Query Language*), selecionando todos os registros da tabela “localidades”, ordenados pela coluna “codLocalidade”, instanciando objetos da classe “modelo.Localidade” para cada uma delas. A Figura 24 exibe a interface, após o carregamento;

Figura 24 – Resultado do carregamento das localidades

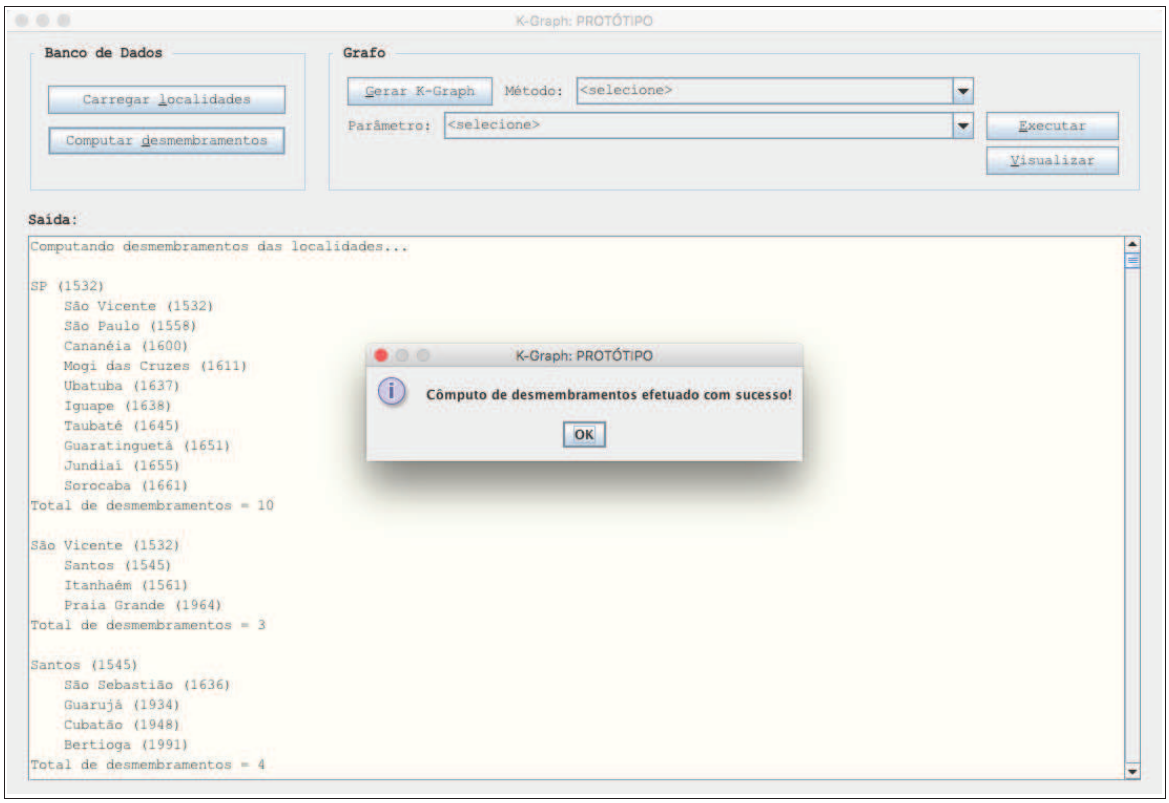


Fonte: Elaborado pelo autor

a.2) Botão **Computar desmembramentos**

Uma vez carregadas as localidades, este botão calcula as localidades-filhas (desmembramentos) para cada localidade, ordenadas pelo ano de criação. Após o cômputo das “descendentes”, estas são atribuídas como uma coleção (atributo) da localidade-mãe (objeto). O resultado é exibido na Figura 25.

Figura 25 – Resultado do cômputo dos desmembramentos



Fonte: Elaborado pelo autor

b) Grupo “Grafo”

Este grupo é composto por três botões e duas caixas de combinação (Figura 26), que permitem manipular o objeto *K-Graph* criado em memória, executando métodos (algoritmos de busca) e visualizando o grafo com seus vértices e linhas.

Figura 26 – Componentes do grupo “Grafo”

Fonte: Elaborado pelo autor

b.1) Botão **Gerar K-Graph**

Instancia um objeto da classe interna “gui.interface.KGraph”, atribuindo vértices (representados pela classe abstrata “grafo.Vertice”) na forma de objetos das classes concretas “grafo.Desmembramento” e “grafo.Localidade”. Um objeto da classe “grafo.Juncao” é instanciado para cada ocorrência de um desmembramento (240 no total), representando uma união geradora de localidades-filhas.

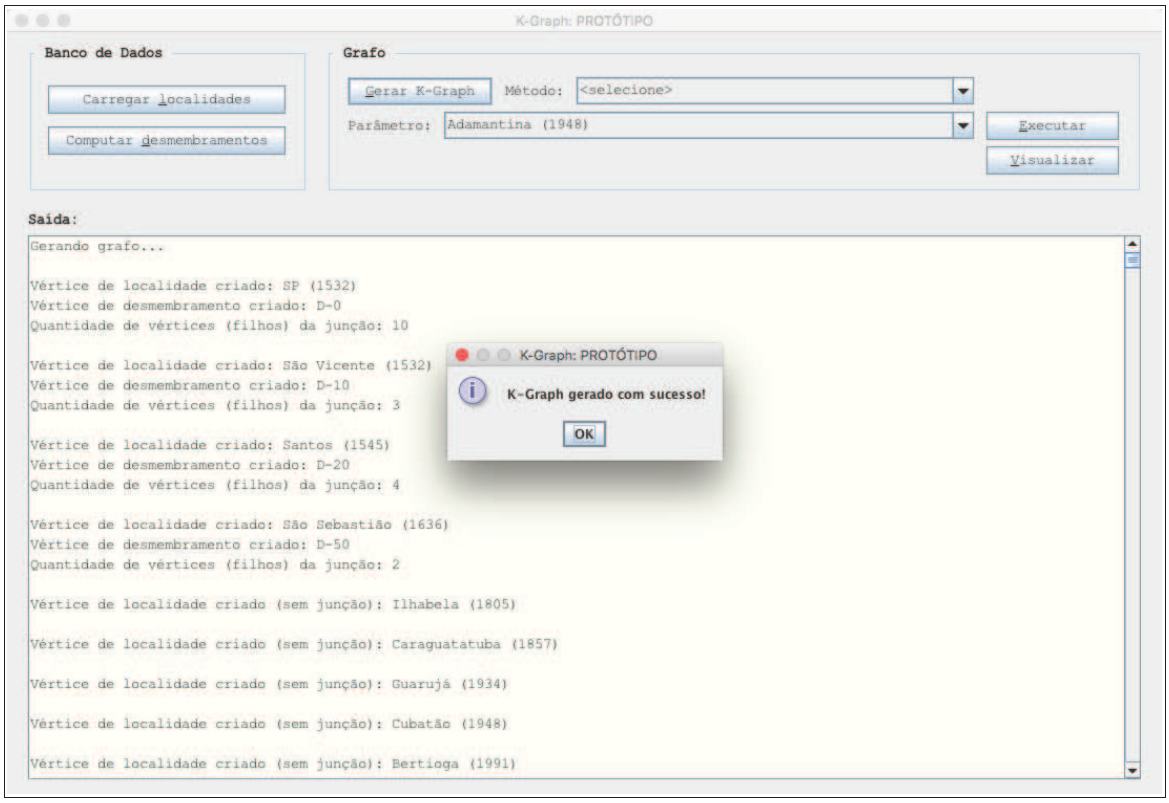
Cabe ressaltar que, conforme consta na planilha original das localidades que serviram de fonte de dados para o protótipo, foi gerado no máximo um desmembramento por localidade, com todas as localidades-filhas de um mesmo desmembramento ordenadas pelo ano de criação (e sendo consideradas “gêmeas” caso possuam o mesmo valor para este atributo). Ainda segundo a planilha, 417 localidades não computam desmembramentos.

Caso o dado temporal fosse uma data específica (dia, mês e ano), uma alternativa – para propósitos de ilustração – seria criar um desmembramento para cada ano distinto das localidades-filhas, desta forma simulando a característica de múltiplos casamentos em um *K-Graph*. Exemplo: para a localidade “Santo André (1889)”, seriam 3 desmembramentos:

1º: São Bernardo do Campo (1944);
2º: São Caetano do Sul (1948);
3º: Mauá (1953) e Ribeirão Pires (1953).

A Figura 27 exibe a interface, após a geração.

Figura 27 – Resultado da geração do *K-Graph*



Fonte: Elaborado pelo autor

A Listagem 2 contém o código do método “instanciaVertices” da classe “gui.Interface”.

Listagem 2 – Código do método privado “instanciaVertices” (em destaque a chamada recursiva)

```
private void instanciaVertices(grafo.Localidade verticeLocalidade,
ArrayList<modelo.Localidade> localidadesDesmembramentos, DefaultListModel
dlm) {

    final Juncao juncao;

    if (!localidadesDesmembramentos.isEmpty()) {

        this.quantidadeJuncoes++;

        dlm.addElement("Vértice de localidade criado: " +
verticeLocalidade);
        grafo.Desmembramento verticeDesmembramento = new
Desmembramento(verticeLocalidade.getCodLocalidade());
        dlm.addElement("Vértice de desmembramento criado: " +
verticeDesmembramento);

        juncao = new Juncao(verticeLocalidade, verticeDesmembramento);
```

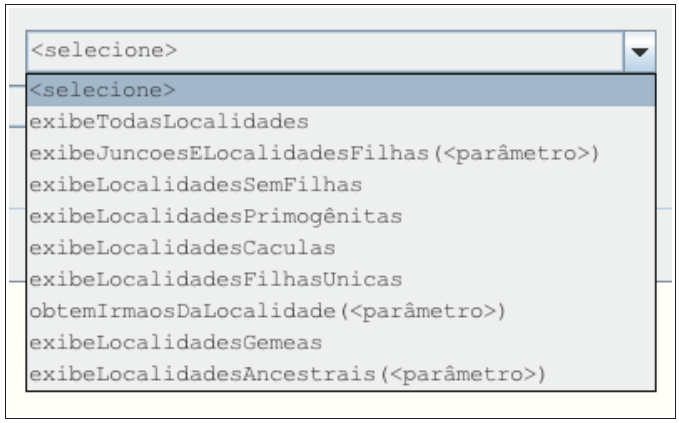
```
        verticeLocalidade.getJuncoes().add(juncao);
        verticeDesmembramento.getJuncoes().add(juncao);
        dlm.addElement("Quantidade de vértices (filhos) da junção: " +
localidadesDesmembramentos.size());
        dlm.addElement("\n");

        ArrayList<grafo.Localidade> localidadesFilhas = new ArrayList<>();

        // estrutura de loop escolhida devido à indexação
        for (int i = 0; i < localidadesDesmembramentos.size(); i++) {

            modelo.Localidade ld;
            ld = localidadesDesmembramentos.get(i);

            grafo.Localidade verticeLocalidadeFilha = new
grafo.Localidade(ld.getCodLocalidade(), ld.getNome(), ld.getAnoCriacao());

            verticeLocalidadeFilha.setOrigem(juncao);

            // relacionamento dos irmãos: menos a "primogênita"
            if (i > 0) {
                // irmã da esquerda
                grafo.Localidade verticeLocalidadeEsquerda =
localidadesFilhas.get(i - 1);

verticeLocalidadeFilha.setLocalidadeEsquerda(verticeLocalidadeEsquerda);
                // irmã da direita
                localidadesFilhas.get(i -
1).setLocalidadeDireita(verticeLocalidadeFilha);
                // localidades gêmeas
                if
(verticeLocalidadeFilha.equals(verticeLocalidadeEsquerda)) {
                    verticeLocalidadeFilha.setGemea(true);
                    verticeLocalidadeEsquerda.setGemea(true);
                    dlm.addElement("Vértices de localidades gêmeas: " +
verticeLocalidadeEsquerda + " & " + verticeLocalidadeFilha);
                    dlm.addElement("\n");
                }
            }
            localidadesFilhas.add(verticeLocalidadeFilha);

            juncao.setLocalidades(localidadesFilhas);


this.kgraph.getVerticesLocalidades().add(verticeLocalidadeFilha);

            this.instanciaVertices(verticeLocalidadeFilha,
ld.getLocalidadesDesmembramentos(), dlm);

        }
    } else {
        dlm.addElement("Vértice de localidade criado (sem junção): " +
verticeLocalidade);
        dlm.addElement("\n");
    }

}
```

Fonte: Implementado pelo autor

b.2) Caixa de Combinação **Método**

Após a geração do grafo, nove métodos são disponibilizados para consulta dos vértices (localidades e desmembramentos). Estes métodos contém algoritmos de busca que “varrem” a estrutura do grafo, localizando e exibindo resultados conforme o propósito e, em três casos, considerando o parâmetro informado pelo usuário. A nomenclatura utilizada foi adaptada para a área aplicada (Geografia), e os códigos foram baseados no Apêndice B. A Figura 28 exibe o conteúdo do componente, e as Listagens 3 a 11 relacionam os métodos que foram implementados no protótipo.

Figura 28 – Itens da Caixa de Combinação “Método"



Fonte: Elaborado pelo autor

Listagem 3 – Código do método protegido “exibeTodasLocalidades”, que recebe como parâmetros a junção “raiz” e uma varíavel para totalização (em destaque a chamada recursiva)

```
protected void exibeTodasLocalidades(Juncao juncaoEstado, int total) {
    if (!juncaoEstado.getLocalidades().isEmpty()) {
        int size = juncaoEstado.getLocalidades().size();
        dlm.addElement("Desmembramentos = " + size);
        totalLocalidades = totalLocalidades + size;
        juncaoEstado.getLocalidades().forEach((localidade) -> {
            dlm.addElement(localidade);
            localidade.getJuncoes().forEach((juncao) -> {
                exibeTodasLocalidades(juncao, totalLocalidades);
            });
        });
    }
}
```

Fonte: Implementado pelo autor

Listagem 4 – Código do método protegido “exibeJuncoesELocalidadesFilhas”, que recebe como parâmetros o vértice “raiz” (localidade SP) e a localidade informada pelo usuário (em destaque a chamada recursiva)

```
protected void exibeJuncoesELocalidadesFilhas(grafo.Localidade
localidadeRaiz, grafo.Localidade localidadePai) {
    if
(localidadeRaiz.getCodLocalidade().equals(localidadePai.getCodLocalidade())
) {
        if (localidadeRaiz.getJuncoes().isEmpty()) {
            dlm.addElement("Localidade não possui filhas");
            return;
        }
        localidadeRaiz.getJuncoes().forEach((juncao) -> {
            dlm.addElement("Junção: " + juncao);
            dlm.addElement("\n");
            juncao.getLocalidades().forEach((filha) -> {
                dlm.addElement(filha);
            });
            dlm.addElement("\n");
            dlm.addElement("Total de localidades-filhas = " +
juncao.getLocalidades().size());
        });
    } else {
        localidadeRaiz.getJuncoes().forEach((juncao) -> {
            juncao.getLocalidades().forEach((filha) -> {
                exibeJuncoesELocalidadesFilhas(filha, localidadePai);
            });
        });
    }
}
```

Fonte: Implementado pelo autor

Listagem 5 – Código do método protegido “exibeLocalidadesSemFilhas”, que recebe como parâmetros a junção “raiz” e uma varíavel para totalização (em destaque a chamada recursiva)

```
protected void exibeLocalidadesSemFilhas(Juncao juncaoRaiz, int total) {
    if (!juncaoRaiz.getLocalidades().isEmpty()) {
        juncaoRaiz.getLocalidades().forEach((localidade) -> {
            if (localidade.getJuncoes().isEmpty()) {
                dlm.addElement(localidade);
                totalLocalidades++;
            } else {
                localidade.getJuncoes().forEach((juncao) -> {
                    exibeLocalidadesSemFilhas(juncao, totalLocalidades);
                });
            }
        });
    }
}
```

Fonte: Implementado pelo autor

Listagem 6 – Código do método protegido “exibeLocalidadesPrimogênitas”, que recebe como parâmetro a junção “raiz” (em destaque a chamada recursiva)

```
protected void exibeLocalidadesPrimogênitas(Juncao juncaoRaiz) {
    if (!juncaoRaiz.getLocalidades().isEmpty()) {
        juncaoRaiz.getLocalidades().forEach((localidade) -> {
            if (localidade.getLocalidadeEsquerda() == null) {
                dlm.addElement(localidade);
            }
            localidade.getJuncoes().forEach((juncao) -> {
                exibeLocalidadesPrimogênitas(juncao);
            });
        });
    }
}
```

Fonte: Implementado pelo autor

Listagem 7 – Código do método protegido “exibeLocalidadesCaculas”, que recebe como parâmetro a junção “raiz” (em destaque a chamada recursiva)

```
protected void exibeLocalidadesCaculas(Juncao juncaoRaiz) {
    if (!juncaoRaiz.getLocalidades().isEmpty()) {
        juncaoRaiz.getLocalidades().forEach((localidade) -> {
            if (localidade.getLocalidadeDireita() == null) {
                dlm.addElement(localidade);
            }
            localidade.getJuncoes().forEach((juncao) -> {
                exibeLocalidadesCaculas(juncao);
            });
        });
    }
}
```

Fonte: Implementado pelo autor

Listagem 8 – Código do método protegido “exibeLocalidadesFilhasUnicas”, que recebe como parâmetros a junção “raiz” e uma varíavel para totalização (em destaque a chamada recursiva)

```
protected void exibeLocalidadesFilhasUnicas(Juncao juncaoRaiz, int total) {
    if (!juncaoRaiz.getLocalidades().isEmpty()) {
        juncaoRaiz.getLocalidades().forEach((localidade) -> {
            if ((localidade.getLocalidadeEsquerda() == null) &&
(localidade.getLocalidadeDireita() == null)) {
                dlm.addElement(localidade);
                totalLocalidades++;
            }
            localidade.getJuncoes().forEach((juncao) -> {
                exibeLocalidadesFilhasUnicas(juncao, totalLocalidades);
            });
        });
    }
}
```

Fonte: Implementado pelo autor

Listagem 9 – Código do método protegido “obtemIrmaosDaLocalidade”, que recebe como parâmetros a junção “raiz”, a localidade informada pelo usuário, e uma coleção para armazenamento (em destaque as chamadas aos métodos privativos)

```
protected LinkedHashSet<grafo.Localidade> obtemIrmaosDaLocalidade(Juncao
juncaoRaiz, grafo.Localidade localidade,
        LinkedHashSet<grafo.Localidade> irmaos) {
    irmaos.addAll(obtemIrmaosMaisVelhos(juncaoRaiz, localidade, new
LinkedHashSet<>()));
    irmaos.addAll(obtemIrmaosMaisNovos(juncaoRaiz, localidade, new
LinkedHashSet<>()));
    return irmaos;
}
```

Fonte: Implementado pelo autor

Listagem 10 – Código do método protegido “exibeLocalidadesGemeas”, que recebe como parâmetro a junção “raiz” (em destaque a chamada recursiva)

```
protected void exibeLocalidadesGemeas(Juncao juncaoRaiz) {
    if (!juncaoRaiz.getLocalidades().isEmpty()) {
        ArrayList<grafo.Localidade> irmaos = juncaoRaiz.getLocalidades();
        if (!irmaos.isEmpty()) {
            List<grafo.Localidade> gemeos = new ArrayList<>();
            irmaos.forEach((irmao) -> {
                if ((irmao.equals(irmao.getLocalidadeDireita())) ||
(irmao.equals(irmao.getLocalidadeEsquerda()))) {
                    gemeos.add(irmao);
                }
            });
            if (!gemeos.isEmpty()) {
                dlm.addElement("\n");
                dlm.addElement("Junção: " + juncaoRaiz);
                Map<Integer, List<grafo.Localidade>> gemelaridades = gemeos
.collect(Collectors.groupingBy(grafo.Localidade::getAnoCriacao));
                for (Integer anoCriacao : gemelaridades.keySet()) {
                    String localidadeGemeas =
gemelaridades.get(anoCriacao).stream().map(grafo.Localidade::getNome).colle
ct(Collectors.joining(", "));
                    dlm.addElement("Localidades-gêmeas: " +
localidadeGemeas + " (ano de criação: " + anoCriacao + ")");
                }
            }
            juncaoRaiz.getLocalidades().forEach((localidade) -> {
                localidade.getJuncoes().forEach((juncao) -> {
                    exibeLocalidadesGemeas(juncao);
                });
            });
        }
    }
}
```

Fonte: Implementado pelo autor

Listagem 11 – Código do método protegido “exibeLocalidadesAncestrais”, que recebe como parâmetro a localidade informada pelo usuário (em destaque a chamada recursiva)

```
protected void exibeLocalidadesAncestrais(grafo.Localidade localidade) {
    if (localidade.getOrigem() != null) {
        grafo.Localidade localidadePai =
localidade.getOrigem().getLocalidade();
        dlm.addElement(localidadePai);
        exibeLocalidadesAncestrais(localidadePai);
    }
}
```

Fonte: Implementado pelo autor

Todos os métodos que recebem como primeiro parâmetro uma junção ou vértice “raiz” realizam buscas em profundidade no grafo, a partir do topo. O método “exibeLocalidadesAncestrais” realiza o caminho inverso, a partir da localidade parametrizada, até o vértice “raiz”.

Conforme a estrutura definida – e a convenção adotada – para a representação de irmandades em um *K-Graph*, os métodos protegidos “exibeLocalidadesPrimogênitas” e “exibeLocalidadesCaculas” podem ser considerados “linearmente opostos”. O primeiro verifica se a localidade selecionada não possui irmãos à esquerda (portanto, primogênita), e a segunda, se não possui irmãos à direita (portanto, caçula). De maneira similar, o método “exibeLocalidadesFilhasUnicas” verifica apenas se a localidade selecionada não possui irmãos (em nenhum dos lados).

b.3) Caixa de Combinação **Parâmetro**

Após a geração do grafo, as localidades são disponibilizadas para serem utilizadas como parâmetro na execução dos métodos "exibeJuncoesELocalidadesFilhas", "obtemIrmaosDaLocalidade" e "exibeLocalidadesAncestrais". Elas são classificadas em ordem alfabética e, além do nome, é exibido o ano de criação da localidade (entre parênteses). Parte do conteúdo do componente pode ser visto na Figura 29.

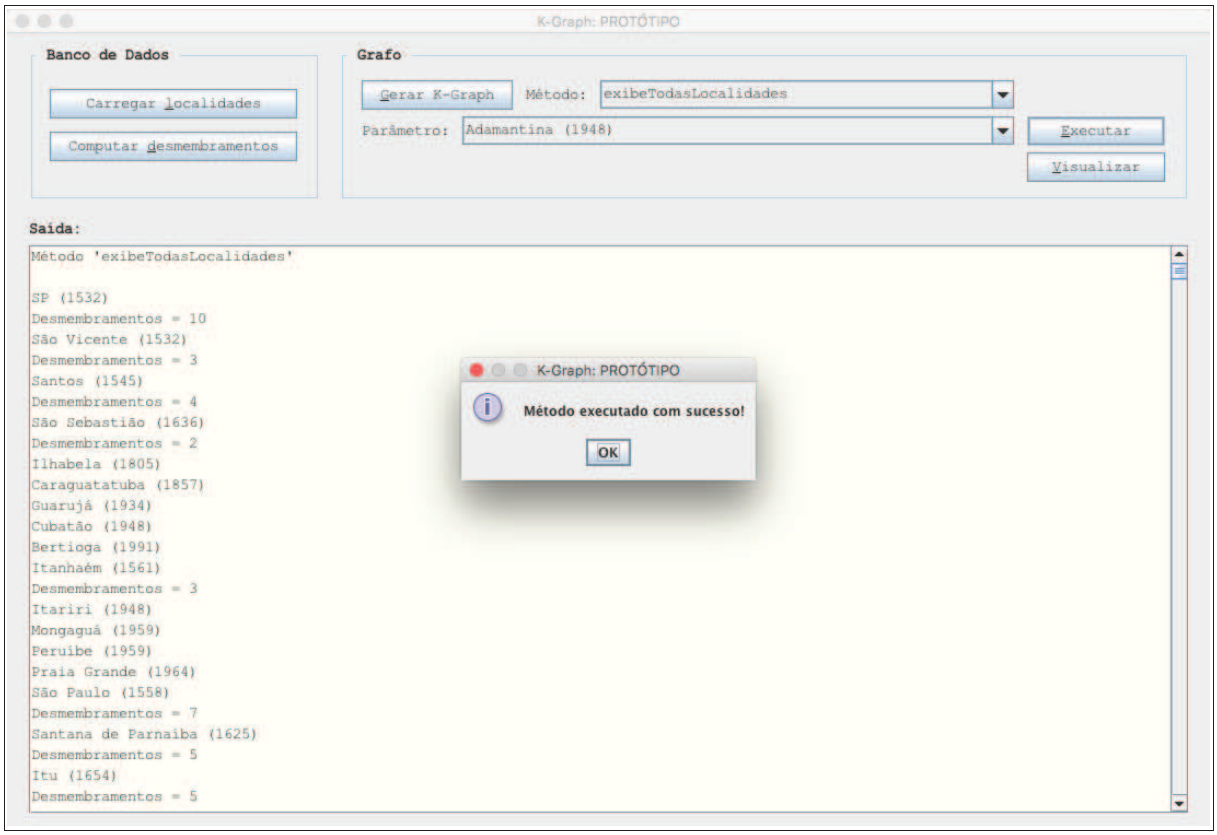
Figura 29 – Parte dos itens da Caixa de Combinação “Parâmetro"

Fonte: Elaborado pelo autor

b.4) Botão **Executar**

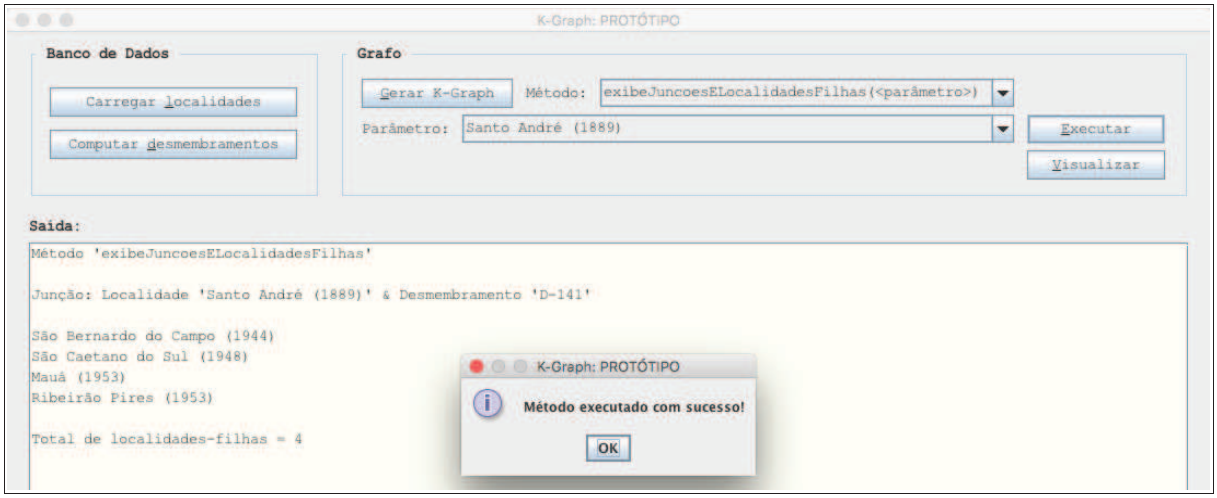
Uma vez validada a seleção de um método, este botão aciona a execução e exibe o respectivo resultado no componente “Saída”. As Figuras 30 a 38 exemplificam a chamada a cada um dos nove métodos disponíveis.

Figura 30 – Resultado obtido com a execução do método “exibeTodasLocalidades”



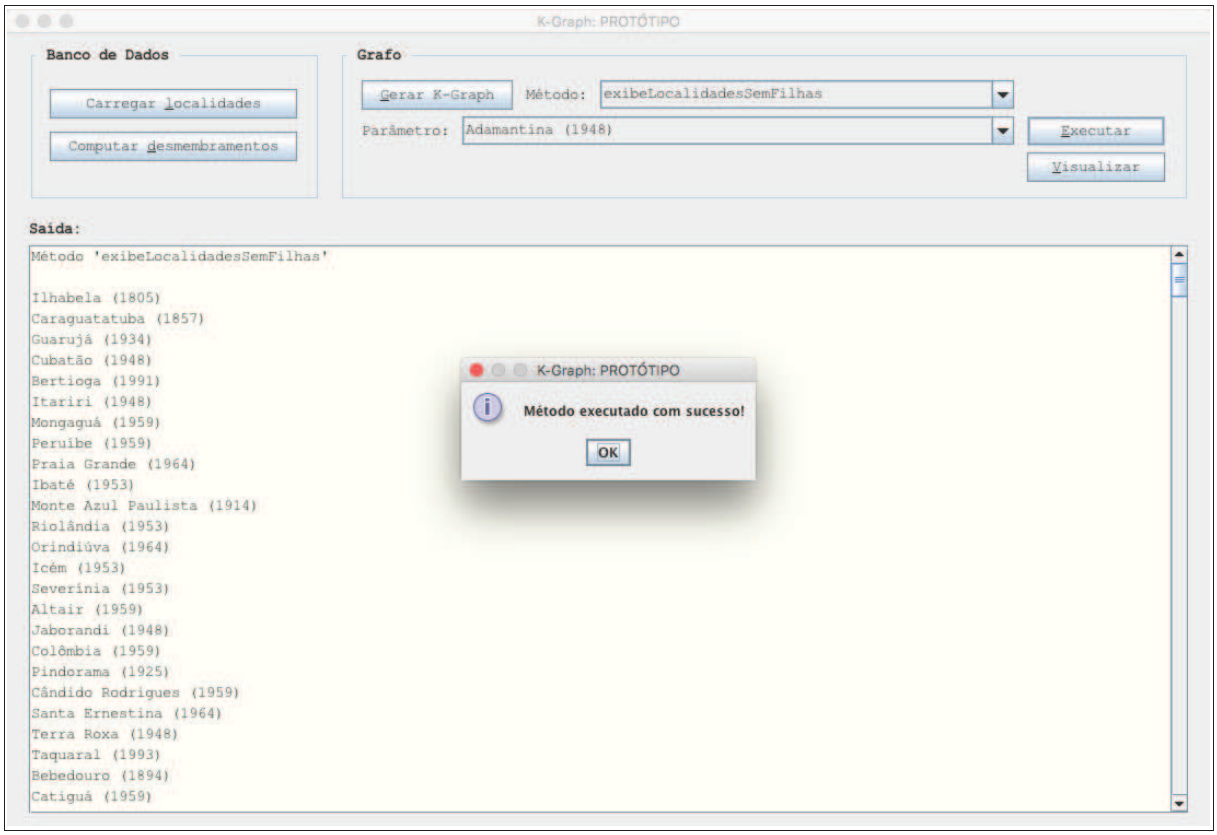
Fonte: Elaborado pelo autor

Figura 31 – Resultado obtido com a execução do método “exibeJuncoesELocalidadesFilhas”, utilizando como parâmetro a localidade “Santo André (1889)”



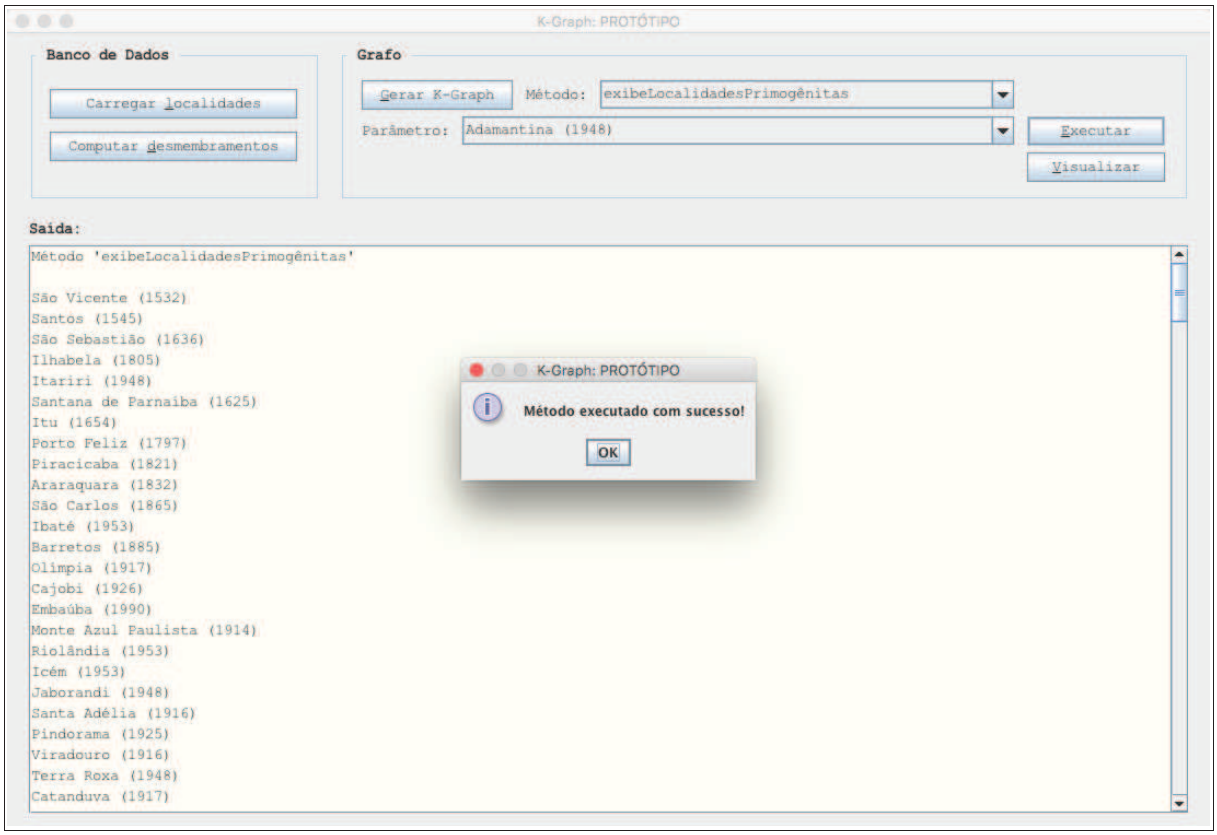
Fonte: Elaborado pelo autor

Figura 32 – Resultado obtido com a execução do método “exibeLocalidadesSemFilhas”



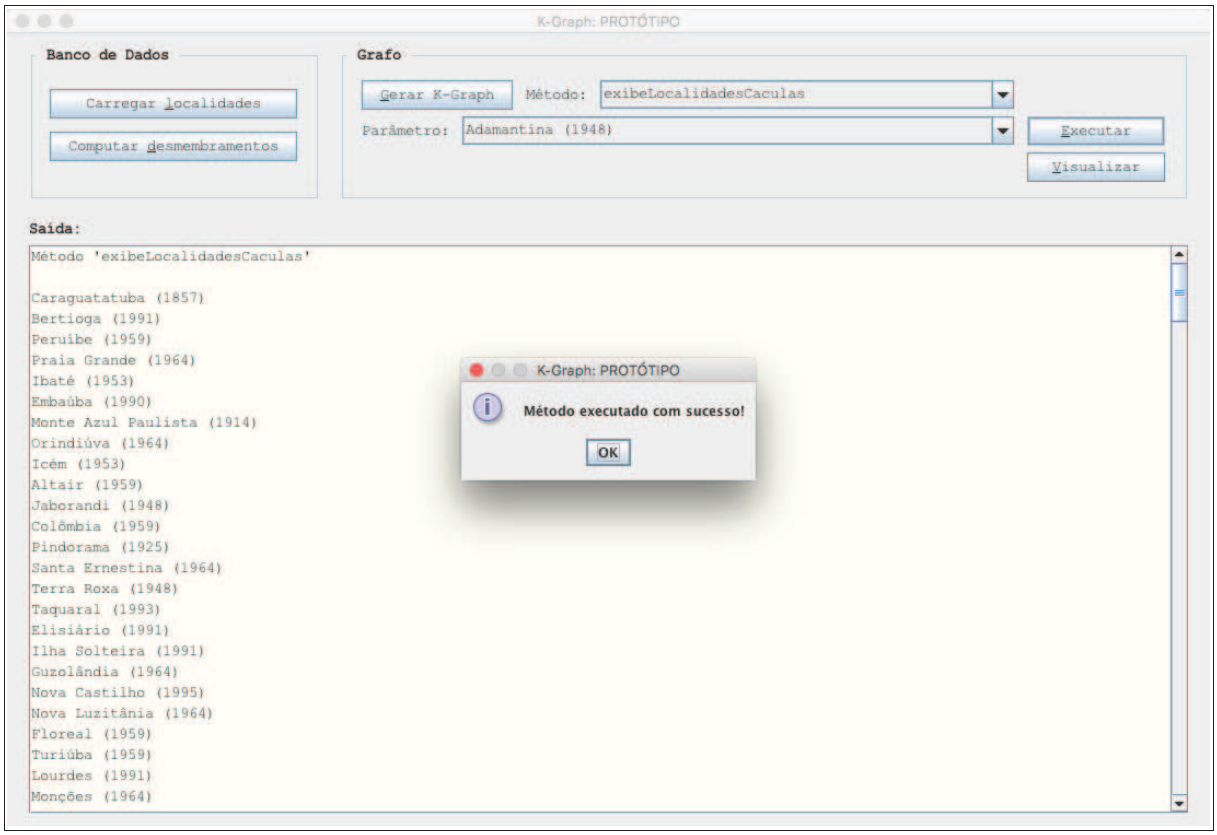
Fonte: Elaborado pelo autor

Figura 33 – Resultado obtido com a execução do método “exibeLocalidadesPrimogênitas”



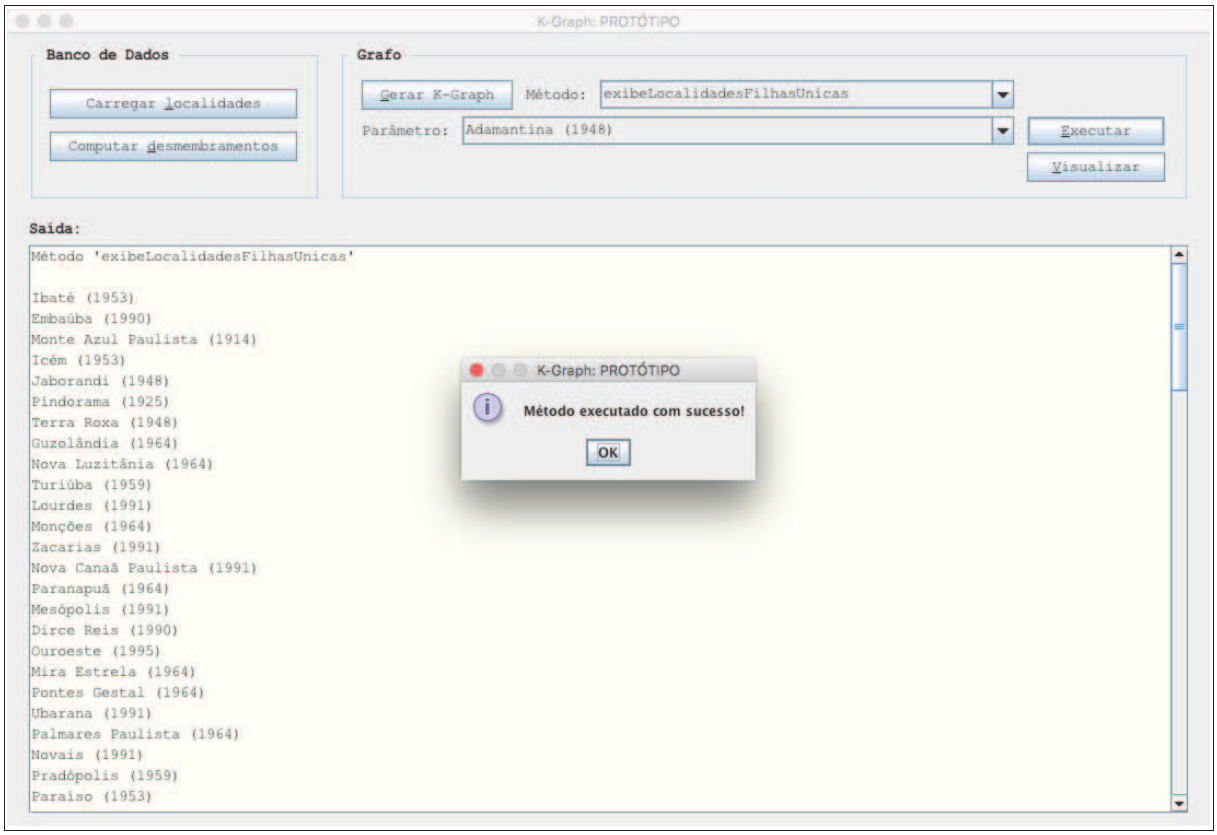
Fonte: Elaborado pelo autor

Figura 34 – Resultado obtido com a execução do método “exibeLocalidadesCaculas”



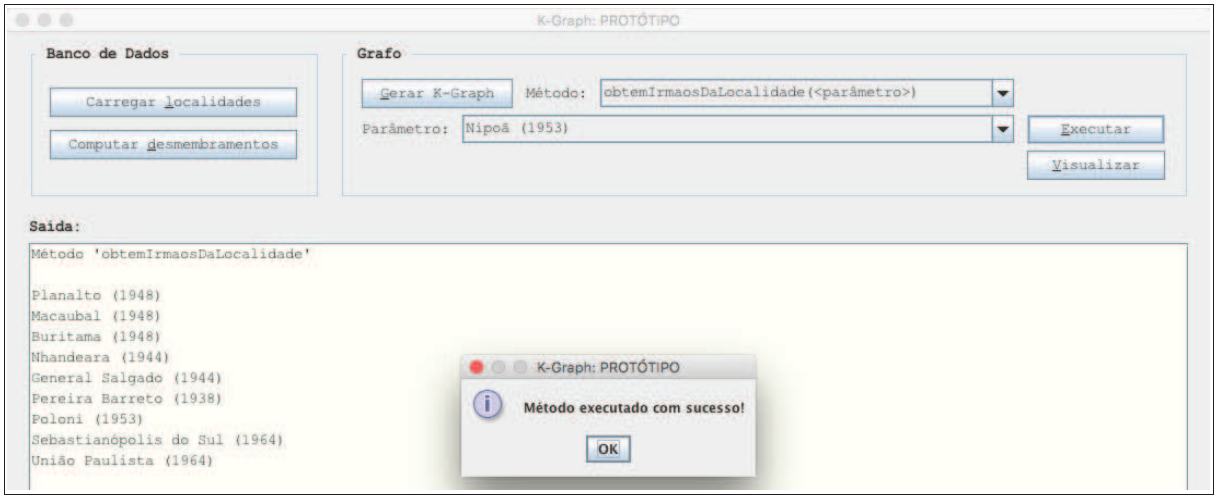
Fonte: Elaborado pelo autor

Figura 35 – Resultado obtido com a execução do método “exibeLocalidadesFilhasUnicas”



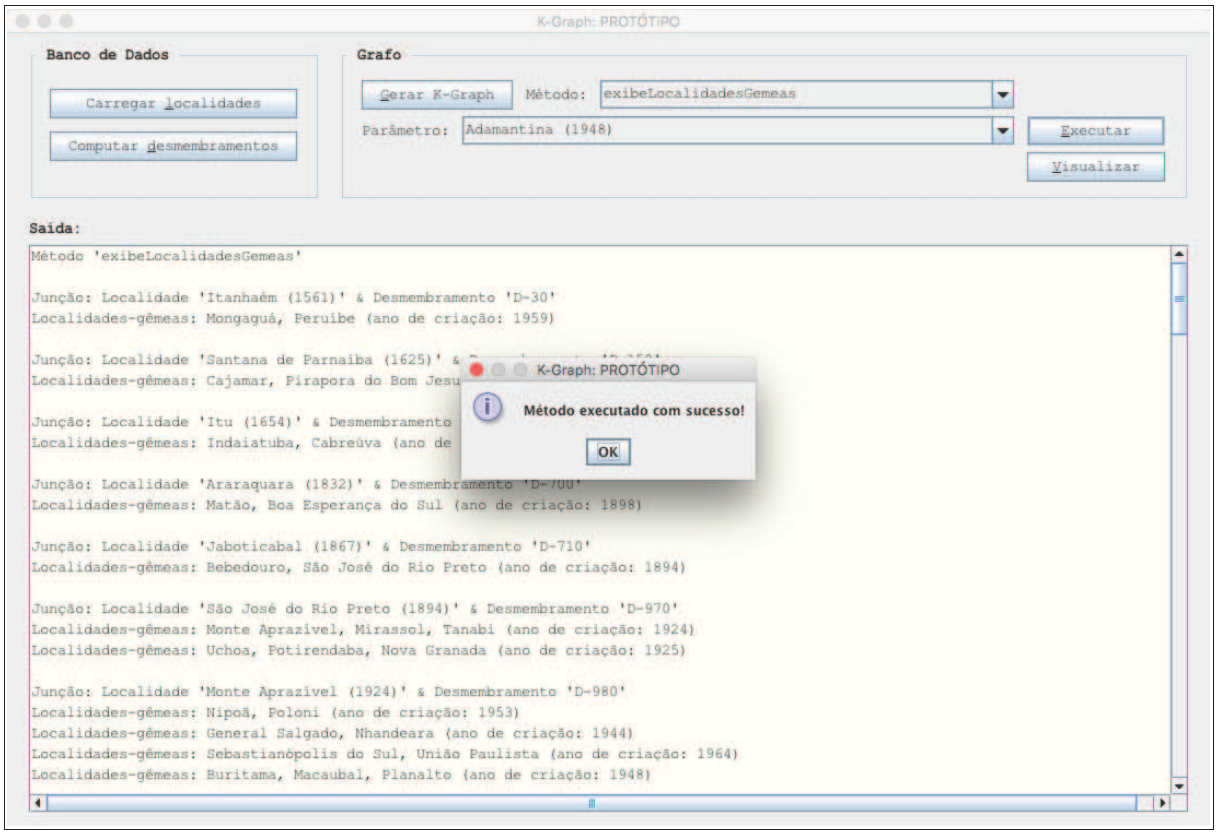
Fonte: Elaborado pelo autor

Figura 36 – Resultado obtido com a execução do método “obtemIrmaosDaLocalidade”, utilizando como parâmetro a localidade “Nipoã (1953)”



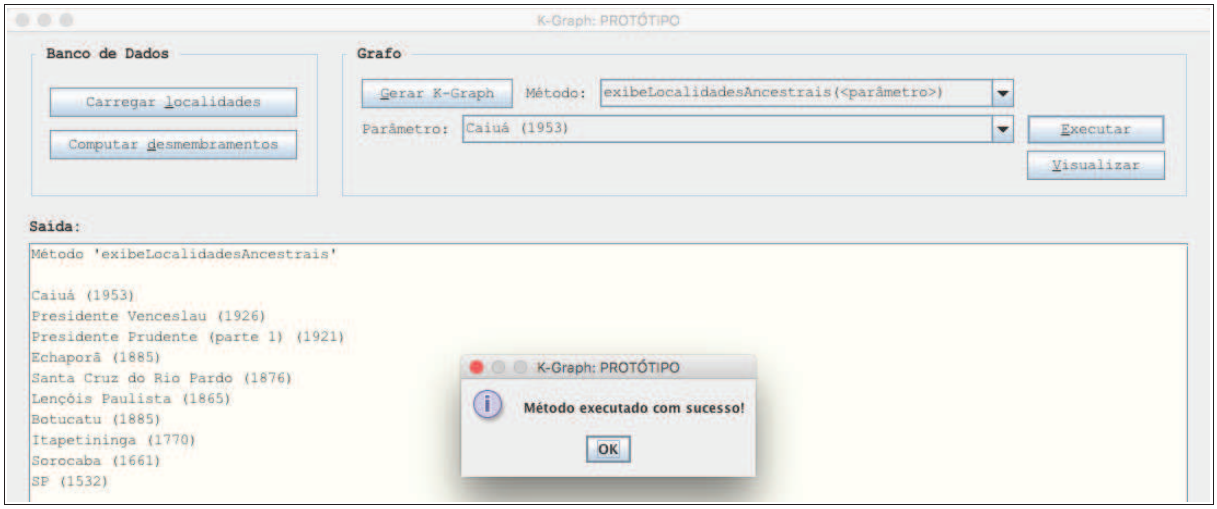
Fonte: Elaborado pelo autor

Figura 37 – Resultado obtido com a execução do método “exibeLocalidadesGemeas”



Fonte: Elaborado pelo autor

Figura 38 – Resultado obtido com a execução do método “exibeLocalidadesAncestrais”, utilizando como parâmetro a localidade “Caiuá (1953)”



Fonte: Elaborado pelo autor

b.5) Botão **Visualizar**

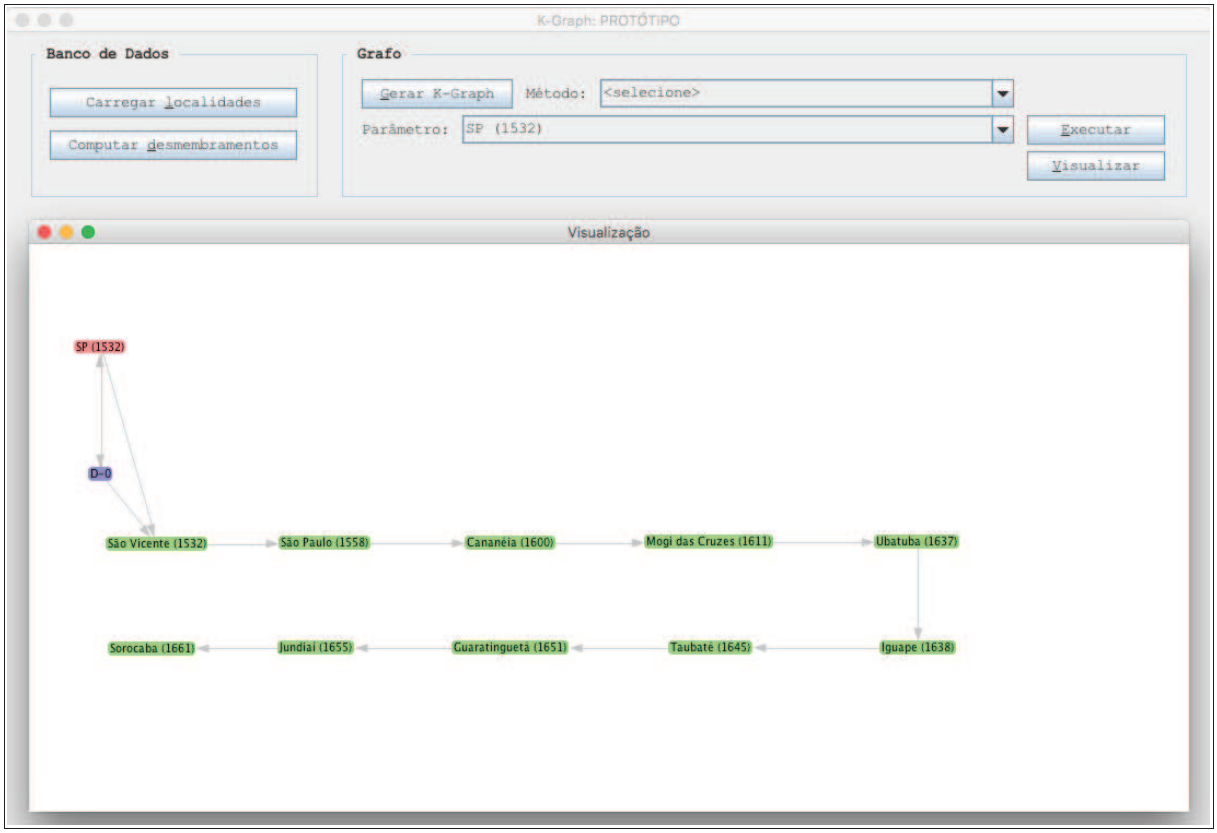
Para a representação visual do grafo foi utilizado o conjunto de ferramentas d e *software* “prefuse”, para criação de visualizações de dados ricas e interativas (PREFUSE, 2018). Na criação de um grafo através do “prefuse” é necessária a configuração prévia do tipo de linhas (“edges”), aplicada em toda a representação visual. Por essa razão, foi utilizada uma topologia com arcos direcionais conectando os vértices.

Este botão instancia um objeto da classe interna “gui.interface.Visualizador” que, por sua vez, é uma especialização da classe “prefuse.Display”. A localidade utilizada como raiz é aquela exibida na Caixa de Combinação “Parâmetro”. O algoritmo (não-recursivo) responsável pela construção do grafo foi simplificado, gerando apenas um vértice para representação da localidade-raiz e, caso esta possua um desmembramento, um vértice para representá-lo além dos vértices adicionais para cada uma das localidades-filhas.

Exemplos de *K-Graphs* gerados para quatro localidades distintas podem ser vistos nas Figuras 39 a 42. As três cores utilizadas serviram unicamente ao

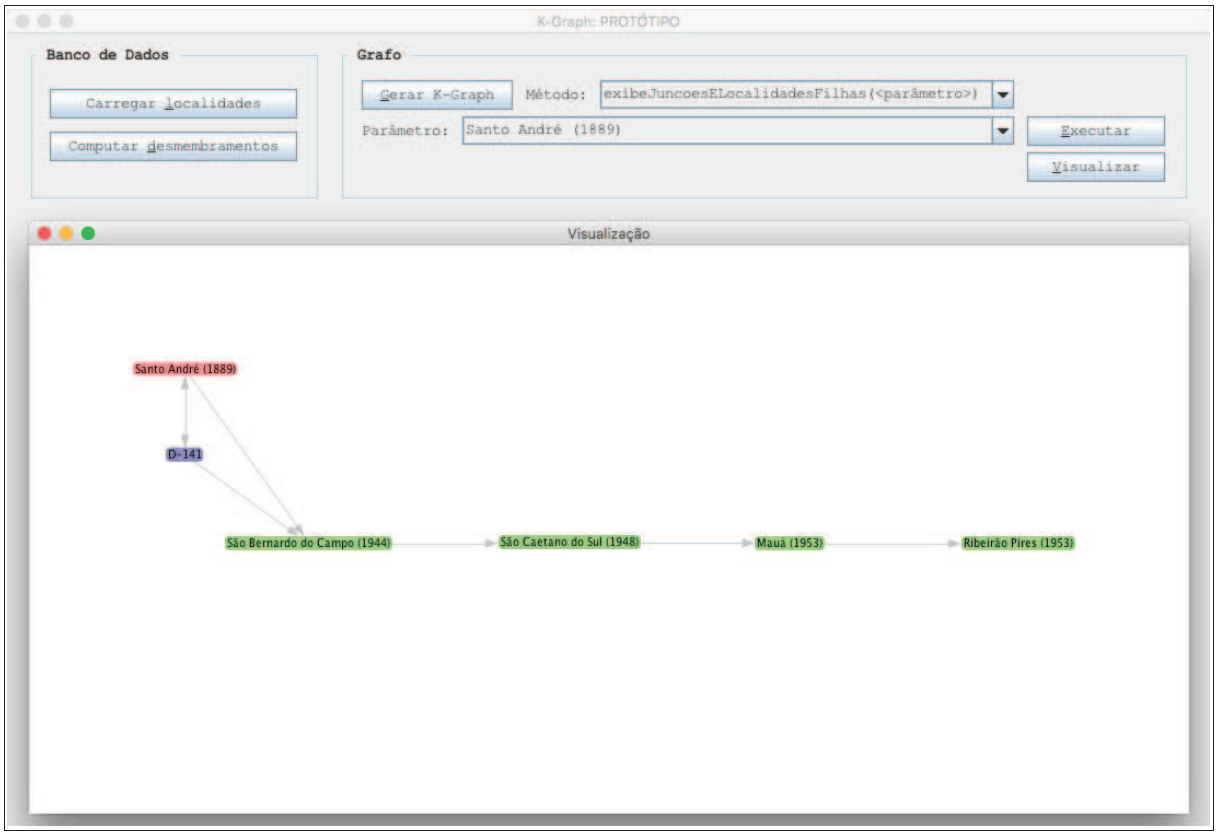
propósito de distinguir vértices (raiz, desdobramento e filha(s)), facilitando a identificação dos componentes. Estas representações gráficas possibilitam uma análise visual dos relacionamentos na rede de localidades geográficas de que trata o protótipo.

Figura 39 – Visualização do *K-Graph* representativo da localidade ”SP (1532)”



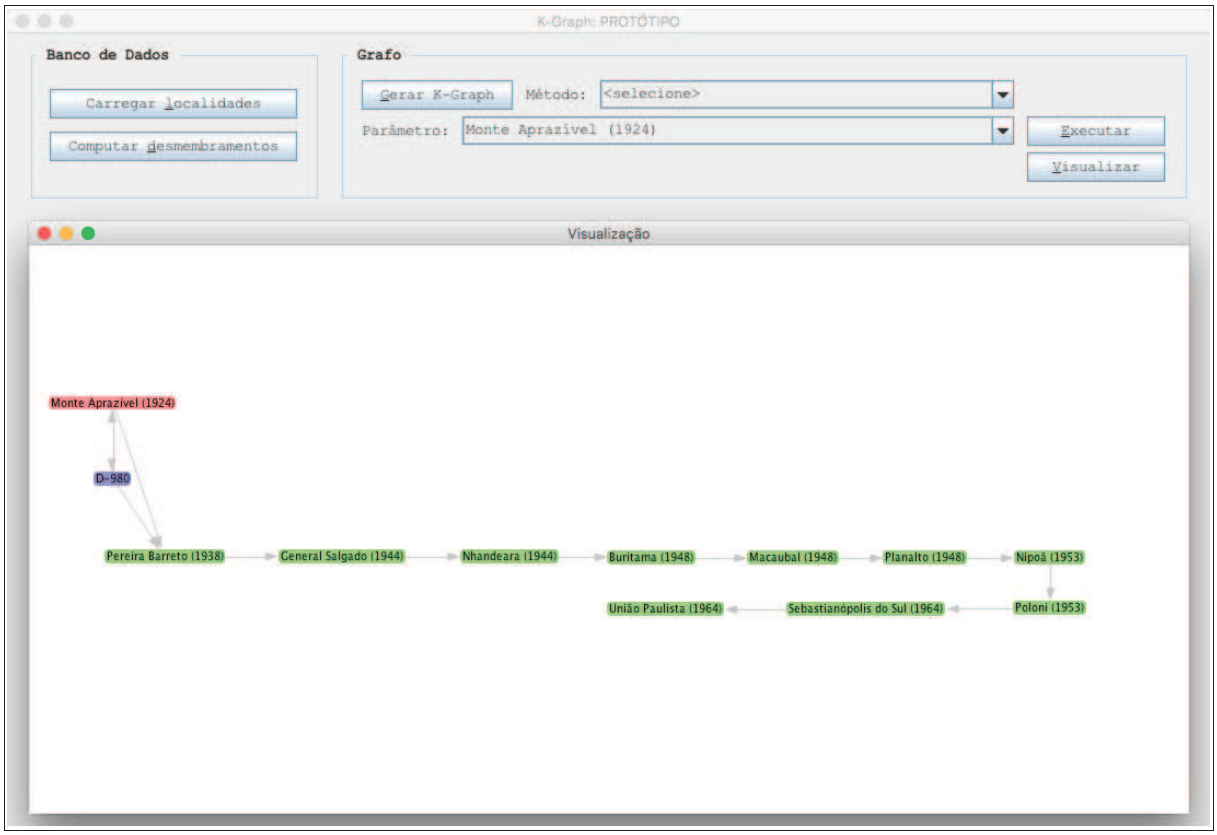
Fonte: Elaborado pelo autor

Figura 40 – Visualização do *K-Graph* representativo da localidade ”Santo André (1889)”



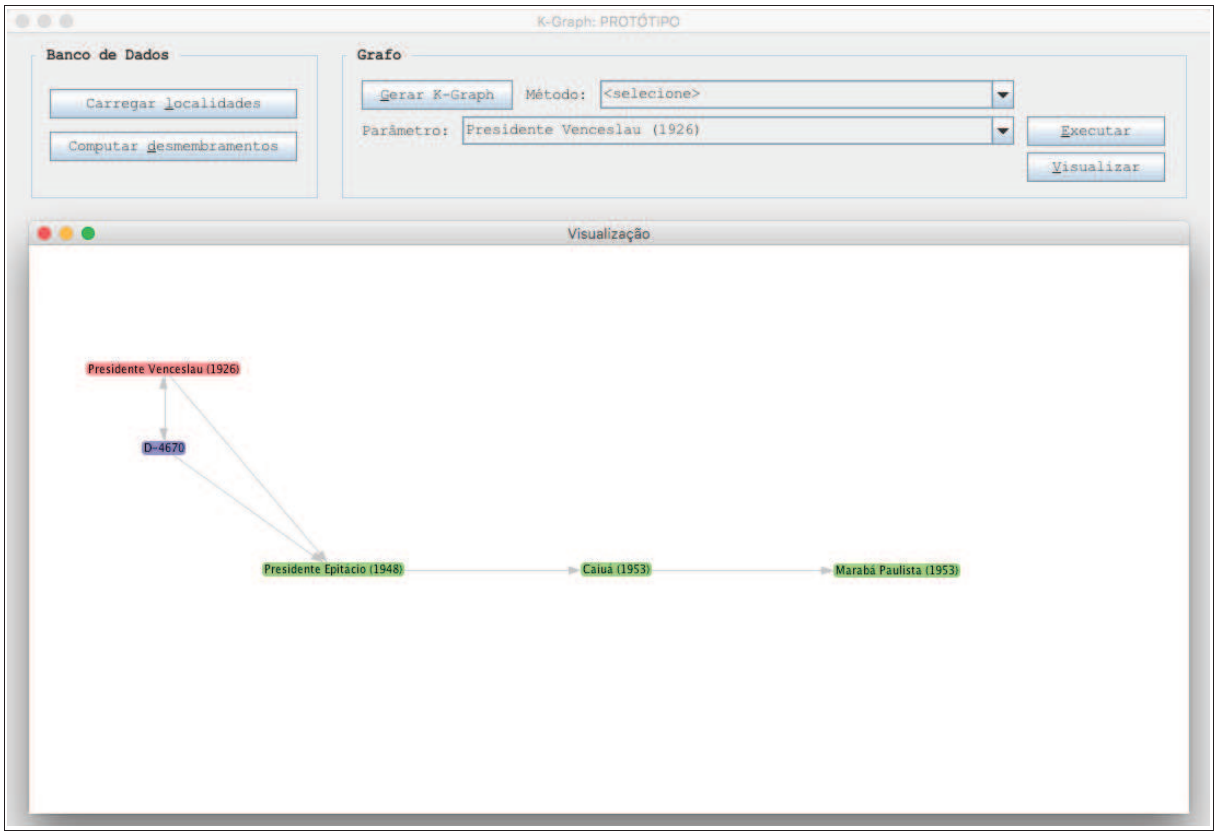
Fonte: Elaborado pelo autor

Figura 41 – Visualização do *K-Graph* representativo da localidade ”Monte Aprazível (1924)”



Fonte: Elaborado pelo autor

Figura 42 – Visualização do *K-Graph* representativo da localidade ”Presidente Venceslau (1926)”



Fonte: Elaborado pelo autor

5 CONCLUSÕES E SUGESTÕES DE PESQUISAS FUTURAS

Este trabalho se constituiu de um estudo, tendo como pilares a Genealogia, a Análise de Redes Sociais e a Teoria dos Grafos. Nesta, as pesquisas realizadas indicaram a existência de três formalizações para a representação de genealogias e, a partir da análise de suas características, uma nova representação foi proposta (*K-Graphs*), implementada em ambiente computacional, e demonstrada a sua eficácia aplicada em outra área de conhecimento.

Tendo em vista atingir o objetivo principal do estudo, *K-Graphs* foram concebidos baseando-se nas deficiências apontadas nos grafos tradicionais. Embora o novo grafo possa ser considerado uma variação dos *Ore Graphs*, ele suprimiu o aspecto dos “arcos cruzados” encontrados nestes, o que simplificou a análise visual e melhorou a legibilidade das representações gráficas.

Uma vez estabelecida a base conceitual do novo grafo, procedeu-se à sua implementação. O primeiro objetivo específico foi atingido com o desenvolvimento de classes estruturais e de algoritmos. Neste sentido, o código constante dos Apêndices A e B poderia ser utilizado e/ou adaptado na elaboração de um *software* de ARS com enfoque em análise de redes de parentesco.

Como prova de conceito, demonstrando a eficácia da aplicação do *K-Graph* na solução de um problema similiar ao de representações genealógicas, o segundo objetivo específico foi atingido com o desenvolvimento de um protótipo. Nele, localidades geográficas foram representadas e visualizadas como na estrutura do novo grafo, além de oferecer opções de busca parametrizadas, auxiliando na análise da rede, extraindo informações de seus vértices e de seus desmembramentos. Desta forma, foi possível prever uma contribuição em outras áreas onde os preceitos da Genealogia sejam aplicáves como, por exemplo, linhas ou famílias de produtos de evolução incremental.

Não obstante os resultados atingidos com este estudo, pesquisas mais aprofundadas seriam necessárias para se encontrar outras formalizações que, possivelmente, existam na Teoria do Grafos para se representar redes de parentesco e, a partir delas, comparar suas características com as dos *K-Graphs*. Este estudo limitou-se às representações mais usuais disponíveis na literatura correspondente, o que, até certo ponto, pode ser considerado um viés do trabalho.

Ainda como sugestão de continuidade da pesquisa, faz-se necessário ressaltar que os algoritmos foram aqui concebidos e implementados tendo-se como preocupação primordial a sua funcionalidade. Portanto, um levantamento e análise quanto à sua performance e otimização – ou mesmo um comparativo com outros algoritmos já utilizados por *softwares* de análise de redes no mercado – seria de fundamental importância.

Uma outra restrição adotada neste estudo diz respeito à forma como a Genealogia, como disciplina, foi tratada: tradicionalmente, caracterizada por laços convencionais de parentesco entre os membros de uma família (relacionamentos heterossexuais, filhos biológicos, irmãos como filhos de um mesmo pai e de uma mesma mãe, etc.). Tal premissa foi assumida unicamente por uma questão pragmática.

Uma abordagem suplementar deste estudo poderia considerar endogamias e relações incestuosas, por exemplo, as quais são recorrentes em populações endêmicas, além de casos de adoções (descendências não biológicas) que desfrutam do mesmo *status* de filiações.

Sistemas sociais não familiares e relações homoafetivas são cada vez mais uma realidade irrefutável nas sociedades e, em muitas delas, com prerrogativas legais. Estes cenários não ortodoxos podem ser retratados como Afinografos (tradução livre da palavra inglesa "Affinographs”) e, igualmente, serem objeto de pesquisas futuras onde a Teoria dos Grafos possa fornecer uma representação formal destes laços.

Um afinografo é uma imagem gráfica dinâmica das relações familiares acumuladas através da descendência, uniões civis, uniões informais, casamento, procriação e adoção. Um afinografo dá um significado mais preciso às famílias referidas como quebradas, nucleares, reconstituídas, mescladas, monoparentais e outras (RESEARCHGATE, 2018).

A palavra “família" tradicionalmente se refere a dois pais e seus filhos, um conceito que tem sido constantemente revisitado ao longo do último meio século com a aceitação gradual da sociedade de famílias monoparentais, mescladas, adotadas e do mesmo sexo. E, à medida que as relações aumentam em complexidade, também aumentam seus problemas, exigindo ferramentas profissionais inovadoras para entender e avaliar as famílias (SPRINGER, 2018).

REFERÊNCIAS

BALL, Robert. **Visualizing Genealogy Through a Family-Centric Perspective**. Information Visualization. SAGE Publications. Sage UK: London, England, 2017, v.16, pp. 74-89.

BARNES, John A.; HARARY, Frank. **Graph Theory in Network Analysis**. Social Networks. Elsevier, 1983, v.5, pp. 235-244.

BATAGELJ, Vladimir; MRVAR, Andrej. **Analysis of Kinship Relations with Pajek**. Social Science Computer Review. Sage Publications. Sage CA: Los Angeles, CA, 2008, v.26, pp. 224-246.

CHARTRAND, Gary; LESNIAK, Linda; ZHANG, Ping. **Graphs & Digraphs**. 6ª edição. CRC Press, 2010.

DBEAVER. **Free Universal SQL Client**. Disponível em: <https://dbeaver.jkiss.org>. Acesso em: 01/05/2018.

DEO, Narsingh. **Graph Theory with Applications to Engineering and Computer Science** (Prentice Hall Series in Automatic Computation). Prentice-Hall, 1974.

FISCHER, Michael D. **Kinship Terminology**. Representing Anthropological Knowledge: Calculating Kinship. Disponível em: <http://era.anthropology.ac.uk/Kinship/prologTerm5.html>. Acesso em: 03/11/2017.

FUNDAÇÃO SEADE. **Institucional**. Disponível em: <http://www.seade.gov.br/institucional/quem-somos/>. Acesso em: 21/04/2018.

GENI. **Family Tree & Family History at Geni.com**. Disponível em: <https://www.geni.com>. Acesso em: 06/11/2017.

HAMBERGER, Klaus; HOUSEMAN, Michael; WHITE, Douglas R. **Kinship Network Analysis**. The Sage Handbook of Social Network Analysis, Sage Publications, 2011, pp. 533-549.

HEVNER, Alan R.; MARCH, Salvatore T.; PARK, Jinsoo. **Design Science in Information Systems Research**. MIS Quaterly, 2004, v.28, n.1, pp. 75-105.

IBGE. **Mapas político-administrativos estaduais**. Disponível em: <ftp://geoftp.ibge.gov.br/cartas_e_mapas/mapas_estaduais_e_distrito_federal/politico/2015/sp_politico850k_2015.pdf>. Acesso em: 23/04/2018.

JABREF. **Graphical Application For Managing Bibliographical Databases**. Disponível em: <http://www.jabref.org/>. Acesso em: 10/05/2018.

LACERDA, Daniel Pacheco; DRESCH, Aline; PROENÇA, Adriano; ANTUNES JÚNIOR, José Antonio Valle. **Design Science Research**: Método de Pesquisa para a Engenharia de Produção. Universidade do Vale do Rio dos Sinos – UNISINOS. São Leopoldo, RS, 2013.

MRVAR, Andrej; BATAGELJ, Vladimir. **Relinking Marriages in Genealogies**. Metodoloˇki zvezki, 2004, v.1, n.2, pp. 407-418.

MRVAR, Andrej; BATAGELJ, Vladimir. **Analysis and visualization of large networks with program package Pajek**. Complex Adaptive Systems Modeling, 2016, v.4, n.1.

MUSEU DA IMIGRAÇÃO DO ESTADO DE SÃO PAULO. **Sobre o Museu da Imigração**. Disponível em: <http://museudaimigracao.org.br/o-museu/sobre/>. Acesso em: 03/03/2018.

NETWORK ENGINEER. **What are the Differences Between Linux Distributions [Debian | Slackware | Redhat]**. Disponível em: <https://networkengineer.me/2015/06/13/what-are-the-differences-between-linux-distributions-debian-slackware-redhat/>. Acesso em: 26/04/2018.

NOOY, Wouter de.; MRVAR, Andrej; BATAGELJ, Vladimir. **Exploratory Social Network Analysis with Pajek**. Cambridge University Press, 2011.

PAJEK. **Program Package Pajek / PajekXXL**. Disponível em: <http://mrvar.fdv.uni-lj.si/pajek/>. Acesso em: 26/11/2017.

POZ, João Dal; SILVA, Marcio Ferreira da. **Informatizando o método genealógico**: um guia de referência para a Máquina do Parentesco. Teoria e Cultura, 2008, v.3.

PREFUSE. **Interactive Information Visualization Toolkit**. Disponível em: <http://prefuse.org>. Acesso em: 21/05/2018.

RESEARCHGATE. **What Is an Affinograph?**. Disponível em: <https://www.researchgate.net/publication/302505307_What_Is_an_Affinograph>. Acesso em: 02/06/2018.

ROOTSWEB. **The GEDCOM Standard Release 5.5**: Introduction. Disponível em: <http://homepages.rootsweb.ancestry.com/~pmcbride/gedcom/55gcint.htm#S1>. Acesso em: 04/11/2017.

ROSO, Adriane. **Psicologia e história**: acerca da construção de árvores genealógicas ou como retomar lembranças de família em sociedades de rede. Revista Psico, jul./set. 2010, v.41, n.3, pp. 385-392.

SEADE [1]. **Caracterização do Território**. Disponível em: <http://produtos.seade.gov.br/produtos/atlasecon/intro/cap2_intro.pdf>. Acesso em: 22/04/2018.

SEADE [2]. **Conheça quando e como se desmembraram os 645 municípios paulistas**. Disponível em: <http://www.seade.gov.br/conheca-quando-e-como-se-desmembraram-os-645-municipios-paulistas/>. Acesso em: 24/04/2018.

SPRINGER. **Affinographs - A Dynamic Method for Assessment of Individuals, Couples, Families, and Households**. Disponível em: <https://www.springer.com/br/book/9781441993946>. Acesso em: 02/06/2018.

TOM'S HARDWARE. **Test-Marathon: Alle CPUs von 1993 bis 2006**. Disponível em: <http://www.tomshardware.de/prozessor-vergleich-benchmark-marathon,testberichte-235198-54.html>. Acesso em: 19/04/2018.

WASSERMAN, Stanley; FAUST, Katherine. **Social Network Analysis**: Methods And Applications. Cambridge University Press, 1994.

WHITE, Douglas R.; JORION, Paul. **Kinship Networks and Discrete Structure Theory**: Applications and Implications. Social Networks. Elsevier, 1996, v.18, pp. 267-314.

WHITE, Douglas R.; BATAGELJ, Vladimir; MRVAR, Andrej. **Anthropology: Analyzing Large Kinship and Marriage Networks With Pgraph and Pajek**. Social Science Computer Review. Sage Publications, 1999, v.17, n.3, pp. 245-274.

WHITE, Douglas R. **Ring Cohesion Theory in Marriage and Social Networks**. Mathematics and Social Sciences. 42e année, 2004, n.168, pp. 59-82.

YED. **Graph Editor**. Disponível em: <https://www.yworks.com/products/yed>. Acesso em: 15/04/2018.

APÊNDICE A – CLASSES ESTRUTURAIS

O propósito dos apêndices deste estudo é demonstrar a viabilidade de se aplicar a representação de um *K-Graph* em um ambiente computacional, por meio de uma linguagem de programação orientada a objetos.

Buscou-se demonstrar a robustez do grafo quanto à sua estrutura (classes, atributos, métodos e dependências) e aos seus algoritmos de busca. O exemplo implementado é o que consta na Figura 14 (Seção 4.1.3). A nomenclatura utilizada nos códigos das listagens dos Apêndices A e B foi adaptada para um domínio específico da ARS (genealogias).

A Linguagem de Programação utilizada foi Java (SE Runtime Environment (build 1.8.0_131-b11)), na IDE (*Integrated Development Environment*) eclipse (Version: Oxygen.2 Release (4.7.2)). Testes foram executados em uma máquina MacBook Pro (Apple macOS High Sierra Version 10.13.3 (17D102)).

Listagem 1 – Classe "Gender" do pacote "enumeration" para representação do gênero da pessoa

```
package enumeration;

public enum Gender {

    M("Male"),
    F("Female");

    private final String description;

    Gender(String description) {
        this.description = description;
    }

    public String getDescription() {
        return this.description;
    }

}
```

Listagem 2 – Classe "Person" do pacote "node" para representação de uma pessoa (vértice de um grafo)

```
package node;
```

```
import enumeration.Gender;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

public class Person {

    private Couple parents;
    private String name;
    private Date dateOfBirth;
    private Date dateOfDeath;
    private Gender gender;
    private Person leftSibling;
    private Person rightSibling;
    private Boolean twin = Boolean.FALSE;
    private Boolean ego;
    private ArrayList<Couple> unions = new ArrayList<>();

    public Person(Couple parents, String name, Date dateOfBirth, Date
dateOfDeath, Gender gender, Person leftSibling) {
        this.parents = parents;
        this.name = name;
        this.dateOfBirth = dateOfBirth;
        this.dateOfDeath = dateOfDeath;
        this.gender = gender;
        this.leftSibling = leftSibling;
    }

    public Couple getParents() {
        return this.parents;
    }

    public void setParents(Couple parents) {
        this.parents = parents;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Date getDateOfBirth() {
        return this.dateOfBirth;
    }

    public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    public Date getDateOfDeath() {
        return this.dateOfDeath;
    }

    public void setDateOfDeath(Date dateOfDeath) {
        this.dateOfDeath = dateOfDeath;
```

```
    }

    public Gender getGender() {
        return this.gender;
    }

    public void setGender(Gender gender) {
        this.gender = gender;
    }

    public Person getLeftSibling() {
        return this.leftSibling;
    }

    public void setLeftSibling(Person leftSibling) {
        this.leftSibling = leftSibling;
    }

    public Person getRightSibling() {
        return this.rightSibling;
    }

    public void setRightSibling(Person rightSibling) {
        this.rightSibling = rightSibling;
    }

    public Boolean getTwin() {
        return this.twin;
    }

    public void setTwin(Boolean twin) {
        this.twin = twin;
    }

    public Boolean getEgo() {
        return this.ego;
    }

    public void setEgo(Boolean ego) {
        this.ego = ego;
    }

    public ArrayList<Couple> getUnions() {
        return this.unions;
    }

    public void setUnions(ArrayList<Couple> unions) {
        this.unions = unions;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (this.dateOfBirth != null ? this.dateOfBirth.hashCode() :
0);
        return hash;
    }

    @Override
```

```
    // two persons are considered different, if they do not have both the
same name and the same DOB
    public boolean equals(Object object) {
        if (!(object instanceof Person)) {
            return false;
        }
        Person other = (Person) object;
        if (((this.name == null && other.name != null) || (this.dateOfBirth
== null && other.dateOfBirth != null))
                || ((this.name != null && !this.name.equals(other.name)) ||
(this.dateOfBirth != null && !this.dateOfBirth.equals(other.dateOfBirth))))
{
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        String dataFormatada = sdf.format(this.dateOfBirth);
        return this.name + " (" + dataFormatada + ")";
    }

}
```

Listagem 3 – Classe "Couple" do pacote "node" para representação de relacionamentos conjugais

```
package node;

import java.util.ArrayList;
import java.util.Date;

public class Couple {

    private Person male;
    private Person female;
    private Date unionDate;
    private ArrayList<Person> descendents = new ArrayList<>();

    public Couple(Person male, Person female) {
        this.male = male;
        this.female = female;
    }

    public Person getMale() {
        return this.male;
    }

    public void setMale(Person male) {
        this.male = male;
    }

    public Person getFemale() {
        return this.female;
    }
```

```
    public void setFemale(Person female) {
        this.female = female;
    }

    public Date getUnionDate() {
        return this.unionDate;
    }

    public void setUnionDate(Date unionDate) {
        this.unionDate = unionDate;
    }

    public ArrayList<Person> getDescendents() {
        return this.descendents;
    }

    public void setDescendents(ArrayList<Person> descendents) {
        this.descendents = descendents;
    }

    @Override
    public String toString() {
        return "• " + this.male + " & " + this.female;
    }

}
```

Listagem 4 – Classe "KGraph" do pacote "graph" para representação de um K-Graph

```
package graph;

import node.Couple;
import java.util.ArrayList;

public class KGraph {

    private String title;
    private ArrayList<Couple> ancestors = new ArrayList<>();

    public String getTitle() {
        return this.title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public ArrayList<Couple> getAncestors() {
        return this.ancestors;
    }

    public void setAncestors(ArrayList<Couple> ancestors) {
        this.ancestors = ancestors;
    }
```

```
}
```

Listagem 5 – Classe "Main" do pacote "graph", contendo o método principal para execução do projeto (com os dados genealógicos do exemplo)

```
package graph;

import java.text.ParseException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.LinkedHashSet;

import enumeration.Gender;
import node.Couple;
import node.Person;

public class Main {

      private static KGraph maternalAscendancy = new KGraph();
      private static int totalDescendents, onlyChildrenCount = 0;

      public static void main(String[] args) throws ParseException {

            Date dateOfBirth, dateOfDeath;

            dateOfBirth = new GregorianCalendar(1870, Calendar.JUNE,
12).getTime();
            Person grandparent1 = new Person(null, "Ludwig Plöger",
dateOfBirth, null, Gender.M, null);
            dateOfBirth = new GregorianCalendar(1879, Calendar.MARCH,
27).getTime();
            Person grandparent2 = new Person(null, "Henriette Plöger",
dateOfBirth, null, Gender.F, null);

            Couple unionGreatGrandParents = new Couple(grandparent1,
grandparent2);
            grandparent1.getUnions().add(unionGreatGrandParents);
            grandparent2.getUnions().add(unionGreatGrandParents);

            maternalAscendancy.setTitle("Maternal Ascendancy of Victor
Alexandre Ploeger Mansueli");
            maternalAscendancy.getAncestors().add(unionGreatGrandParents);

            // descendants of the great-grandparents
            dateOfBirth = new GregorianCalendar(1912, Calendar.JUNE,
18).getTime();
            Person elsePloger = new Person(unionGreatGrandParents, "Else
Plöger", dateOfBirth, null, Gender.F, null);
            unionGreatGrandParents.getDescendents().add(elsePloger);

            dateOfBirth = new GregorianCalendar(1913, Calendar.NOVEMBER,
30).getTime();
            Person ludwigPloger = new Person(unionGreatGrandParents,
"Ludwig Plöger", dateOfBirth, null, Gender.M,
```

```
                        elsePloger);
            unionGreatGrandParents.getDescendents().add(ludwigPloger);
            elsePloger.setRightSibling(ludwigPloger);

            dateOfBirth = new GregorianCalendar(1921, Calendar.MAY,
2).getTime();
            dateOfDeath = new GregorianCalendar(1970, Calendar.FEBRUARY,
17).getTime();
            Person erichPloger = new Person(unionGreatGrandParents, "Erich
Plöger", dateOfBirth, dateOfDeath, Gender.M,
                        ludwigPloger);
            unionGreatGrandParents.getDescendents().add(erichPloger);
            ludwigPloger.setRightSibling(erichPloger);

            dateOfBirth = new GregorianCalendar(1922, Calendar.JANUARY,
23).getTime();
            dateOfDeath = new GregorianCalendar(2011, Calendar.OCTOBER,
8).getTime();
            Person luisePloger = new Person(null, "Luise Erna Knippelberg
Plöger", dateOfBirth, dateOfDeath, Gender.F,
                        null);

            Couple unionGrandParents = new Couple(erichPloger,
luisePloger);
            erichPloger.getUnions().add(unionGrandParents);
            luisePloger.getUnions().add(unionGrandParents);

            // descendants of the grandparents
            dateOfBirth = new GregorianCalendar(1944, Calendar.MARCH,
23).getTime();
            dateOfDeath = new GregorianCalendar(1945, Calendar.NOVEMBER,
4).getTime();
            Person agnesPloeger = new Person(unionGrandParents, "Agnes
Auguste Ploeger", dateOfBirth, dateOfDeath, Gender.F,
                        null);
            unionGrandParents.getDescendents().add(agnesPloeger);

            dateOfBirth = new GregorianCalendar(1946, Calendar.OCTOBER,
4).getTime();
            Person helenaBacron = new Person(unionGrandParents, "Helena
Agnésia Bacron", dateOfBirth, null, Gender.F,
                        agnesPloeger);
            unionGrandParents.getDescendents().add(helenaBacron);
            agnesPloeger.setRightSibling(helenaBacron);

            dateOfBirth = new GregorianCalendar(1947, Calendar.NOVEMBER,
26).getTime();
            Person gustavoPloeger = new Person(unionGrandParents, "Gustavo
Alvino Ploeger", dateOfBirth, null, Gender.M,
                        helenaBacron);
            unionGrandParents.getDescendents().add(gustavoPloeger);
            helenaBacron.setRightSibling(gustavoPloeger);

            dateOfBirth = new GregorianCalendar(1953, Calendar.NOVEMBER,
22).getTime();
            Person helminaMansueli = new Person(unionGrandParents, "Helmina
Ploeger Mansueli", dateOfBirth, null, Gender.F,
                        gustavoPloeger);
            unionGrandParents.getDescendents().add(helminaMansueli);
```

```
            gustavoPloeger.setRightSibling(helminaMansueli);

            // descendants of the aunt
            dateOfBirth = new GregorianCalendar(1941, Calendar.SEPTEMBER,
13).getTime();
            Person pedroBacron = new Person(null, "Pedro Bacron",
dateOfBirth, null, Gender.M, null);

            Couple unionAunt = new Couple(pedroBacron, helenaBacron);
            pedroBacron.getUnions().add(unionAunt);
            helenaBacron.getUnions().add(unionAunt);

            dateOfBirth = new GregorianCalendar(1969, Calendar.JUNE,
4).getTime();
            Person cesarBacron = new Person(unionAunt, "Pedro César
Bacron", dateOfBirth, null, Gender.M, null);
            unionAunt.getDescendents().add(cesarBacron);

            dateOfBirth = new GregorianCalendar(1970, Calendar.JUNE,
25).getTime();
            Person claudioBacron = new Person(unionAunt, "Cláudio Gustavo
Bacron", dateOfBirth, null, Gender.M,
                        cesarBacron);
            unionAunt.getDescendents().add(claudioBacron);
            cesarBacron.setRightSibling(claudioBacron);

            dateOfBirth = new GregorianCalendar(1975, Calendar.AUGUST,
29).getTime();
            Person simoneBacron = new Person(unionAunt, "Simone Bacron",
dateOfBirth, null, Gender.F, claudioBacron);
            unionAunt.getDescendents().add(simoneBacron);
            claudioBacron.setRightSibling(simoneBacron);

            dateOfBirth = new GregorianCalendar(1977, Calendar.AUGUST,
2).getTime();
            Person rosanaBacron = new Person(unionAunt, "Rosana Bacron",
dateOfBirth, null, Gender.F, simoneBacron);
            unionAunt.getDescendents().add(rosanaBacron);
            simoneBacron.setRightSibling(rosanaBacron);

            // descendants of the uncle
            dateOfBirth = new GregorianCalendar(1951, Calendar.AUGUST,
26).getTime();
            Person veraPloeger = new Person(null, "Vera Marta Gomes
Ploeger", dateOfBirth, null, Gender.F, null);

            Couple unionUncle = new Couple(gustavoPloeger, veraPloeger);
            gustavoPloeger.getUnions().add(unionUncle);
            veraPloeger.getUnions().add(unionUncle);

            dateOfBirth = new GregorianCalendar(1974, Calendar.JUNE,
8).getTime();
            Person denisePloeger = new Person(unionUncle, "Denise Gomes
Ploeger", dateOfBirth, null, Gender.F, null);
            unionUncle.getDescendents().add(denisePloeger);

            dateOfBirth = new GregorianCalendar(1977, Calendar.OCTOBER,
6).getTime();
            Person elizePloeger = new Person(unionUncle, "Elize Gomes
```

```
Ploeger", dateOfBirth, null, Gender.F, denisePloeger);
            unionUncle.getDescendents().add(elizePloeger);
            denisePloeger.setRightSibling(elizePloeger);

            dateOfBirth = new GregorianCalendar(1980, Calendar.AUGUST,
15).getTime();
            Person gustavoJunior = new Person(unionUncle, "Gustavo Alvino
Ploeger Júnior", dateOfBirth, null, Gender.M,
                        elizePloeger);
            unionUncle.getDescendents().add(gustavoJunior);
            elizePloeger.setRightSibling(gustavoJunior);

            // descendants of the mother
            dateOfBirth = new GregorianCalendar(1949, Calendar.FEBRUARY,
23).getTime();
            Person helioMansueli = new Person(null, "Hélio Mansueli",
dateOfBirth, null, Gender.M, null);

            Couple unionMother = new Couple(helioMansueli,
helminaMansueli);
            helioMansueli.getUnions().add(unionMother);
            helminaMansueli.getUnions().add(unionMother);

            dateOfBirth = new GregorianCalendar(1975, Calendar.AUGUST,
10).getTime();
            Person victorMansueli = new Person(unionMother, "Victor
Alexandre Ploeger Mansueli", dateOfBirth, null,
                        Gender.M, null);
            victorMansueli.setEgo(Boolean.TRUE);
            unionMother.getDescendents().add(victorMansueli);

            dateOfBirth = new GregorianCalendar(1978, Calendar.OCTOBER,
17).getTime();
            Person reneMansueli = new Person(unionMother, "Renê Ploeger
Mansueli", dateOfBirth, null, Gender.M,
                        victorMansueli);
            unionMother.getDescendents().add(reneMansueli);
            victorMansueli.setRightSibling(reneMansueli);

            dateOfBirth = new GregorianCalendar(1983, Calendar.JULY,
11).getTime();
            Person ginaMansueli = new Person(unionMother, "Gina Ploeger
Mansueli", dateOfBirth, null, Gender.F,
                        reneMansueli);
            ginaMansueli.setTwin(Boolean.TRUE);
            unionMother.getDescendents().add(ginaMansueli);
            reneMansueli.setRightSibling(ginaMansueli);

            dateOfBirth = new GregorianCalendar(1983, Calendar.JULY,
11).getTime();
            Person carlaMansueli = new Person(unionMother, "Carla Ploeger
Mansueli", dateOfBirth, null, Gender.F,
                        ginaMansueli);
            carlaMansueli.setTwin(Boolean.TRUE);
            unionMother.getDescendents().add(carlaMansueli);
            ginaMansueli.setRightSibling(carlaMansueli);

}
```

APÊNDICE B – ALGORITMOS DE BUSCA

Todos os 14 algoritmos aqui listados são parametrizados, e fazem uso da técnica de busca por profundidade e por largura. 12 deles utilizam recursão.

Na sequência, a relação dos métodos contendo os algoritmos de busca, com uma breve descrição do propósito de cada um deles.

Listagem 1 – Método "displayAllDescendents", contendo algoritmo para exibição de todos os descendentes de um casal (com subtotais por descendente, e total geral)

```
// invoking
int totalDescendents = 0;
displayAllDescendents(unionGreatGrandParents, totalDescendents);
System.out.println("\nTotal descendents = " + totalDescendents);

void displayAllDescendents(Couple root, int total) {
      if (!root.getDescendents().isEmpty()) {
            int size = root.getDescendents().size();
            System.out.println("Descendents = " + size);
            totalDescendents = totalDescendents + size;
            root.getDescendents().forEach((person) -> {
                  System.out.println(person);
                  person.getUnions().forEach((union) -> {
                        displayAllDescendents(union, totalDescendents);
                  });
            });
}
```

Listagem 2 – Método "displayChildren", contendo algoritmo para exibição dos filhos de cada um dos descendentes de um casal

```
// invoking
displayChildren(unionGreatGrandParents);

void displayChildren(Couple root) {
      if (!root.getDescendents().isEmpty()) {
            ArrayList<Person> siblings = root.getDescendents();
            if (!siblings.isEmpty()) {
                  for (Person sibling : siblings) {
                        System.out.println(sibling);
                  }
                  System.out.println();
            }
            root.getDescendents().forEach((person) -> {
                  person.getUnions().forEach((union) -> {
                        displayChildren(union);
                  });
            });
```

```
	}
}
```

Listagem 3 – Método "displayUnionsAndDescendentsFromPerson", contendo algoritmo para exibição de cada união e respectivos descendentes de uma pessoa

```
// invoking
displayUnionsAndDescendentsFromPerson(unionGreatGrandParents, erichPloger);

void displayUnionsAndDescendentsFromPerson(Couple root, Person
personDescendents) {
	if (!root.getDescendents().isEmpty()) {
		root.getDescendents().forEach((person) -> {
			if (person.equals(personDescendents)) {
				System.out.println("Person: " + person.getName());
				if (person.getUnions().isEmpty()) {
					System.out.println("No descendents...");
					return;
				}
				person.getUnions().forEach((union) -> {
					if (person.getGender().equals(Gender.F)) {
						System.out.println("Union: " +
union.getMale().getName());
					} else {
						System.out.println("Union: " +
union.getFemale().getName());
					}
					System.out.println("Descendents = " +
union.getDescendents().size());
					union.getDescendents().forEach((descendent) ->
{
						System.out.println(descendent);
					});
				});
			} else {
				person.getUnions().forEach((union) -> {
					displayUnionsAndDescendentsFromPerson(union,
personDescendents);
				});
			}
		});
	} else {
		System.out.println("No descendents...");
	}
}
```

Listagem 4 – Método "displayChildlessnessUnions", contendo algoritmo para exibição dos cônjuges de cada união que não gerou filhos

```
// invoking
displayChildlessnessUnions(unionGreatGrandParents);

void displayChildlessnessUnions(Couple root) {
```

```
      if (!root.getDescendents().isEmpty()) {
            root.getDescendents().forEach((person) -> {
                  if (!person.getUnions().isEmpty()) {
                        person.getUnions().forEach((union) -> {
                              if (union.getDescendents().isEmpty()) {
                                    System.out.println(union);
                              } else {
                                    displayChildlessnessUnions(union);
                              }
                        });
                  }
            });
      }
}
```

Listagem 5 – Método "displayFirstBorns", contendo algoritmo para exibição dos primogênitos

```
// invoking
displayFirstBorns(unionGreatGrandParents);

void displayFirstBorns(Couple root) {
      if (!root.getDescendents().isEmpty()) {
            root.getDescendents().forEach((person) -> {
                  if (person.getLeftSibling() == null) {
                        System.out.println(person);
                  }
                  person.getUnions().forEach((union) -> {
                        displayFirstBorns(union);
                  });
            });
      }
}
```

Listagem 6 – Método "displayLastBorns", contendo algoritmo para exibição dos caçulas

```
// invoking
displayLastBorns(unionGreatGrandParents);

void displayLastBorns(Couple root) {
      if (!root.getDescendents().isEmpty()) {
            root.getDescendents().forEach((person) -> {
                  if (person.getRightSibling() == null) {
                        System.out.println(person);
                  }
                  person.getUnions().forEach((union) -> {
                        displayLastBorns(union);
                  });
            });
      }
}
```

Listagem 7 – Método "displayOnlyChildren", contendo algoritmo para exibição dos filhos únicos e total geral

```
// invoking
int onlyChildrenCount = 0;
displayOnlyChildren(unionGreatGrandParents, onlyChildrenCount);
System.out.println("\nOnly children count = " + onlyChildrenCount);

void displayOnlyChildren(Couple root, int count) {
      if (!root.getDescendents().isEmpty()) {
            root.getDescendents().forEach((person) -> {
                  if ((person.getLeftSibling() == null) &&
(person.getRightSibling() == null)) {
                        System.out.println(person);
                        onlyChildrenCount++;
                  }
                  person.getUnions().forEach((union) -> {
                        displayOnlyChildren(union, onlyChildrenCount);
                  });
            });
      }
}
```

Listagem 8 – Método "getYoungerSiblingsFromPerson", contendo algoritmo para obter os irmãos mais novos de uma pessoa

```
// invoking
LinkedHashSet<Person> youngerSiblings =
getYoungerSiblingsFromPerson(unionGreatGrandParents, victorMansueli, new
LinkedHashSet<>());
for (Person youngerSibling : youngerSiblings) {
      System.out.println(youngerSibling);
}

LinkedHashSet<Person> getYoungerSiblingsFromPerson(Couple root, Person
personSiblings,
            LinkedHashSet<Person> youngerSiblings) {
      if (!root.getDescendents().isEmpty()) {
            root.getDescendents().forEach((person) -> {
                  if (person.equals(personSiblings)) {
                        Person rightSibling = person.getRightSibling();
                        if (rightSibling != null) {
                              youngerSiblings.add(rightSibling);

      getYoungerSiblingsFromPerson(person.getParents(), rightSibling,
youngerSiblings);
                        }
                  } else {
                        person.getUnions().forEach((union) -> {
                              getYoungerSiblingsFromPerson(union,
personSiblings, youngerSiblings);
                        });
                  }
            });
      } else {
```

```
            System.out.println("No descendents...");
      }
      return youngerSiblings;
}
```

Listagem 9 – Método "getOlderSiblingsFromPerson", contendo algoritmo para obter os irmãos mais velhos de uma pessoa

```
// invoking
LinkedHashSet<Person> olderSiblings =
getOlderSiblingsFromPerson(unionGreatGrandParents, carlaMansueli, new
LinkedHashSet<>());
for (Person olderSibling : olderSiblings) {
      System.out.println(olderSibling);
}

LinkedHashSet<Person> getOlderSiblingsFromPerson(Couple root, Person
personSiblings,
            LinkedHashSet<Person> olderSiblings) {
      if (!root.getDescendents().isEmpty()) {
            root.getDescendents().forEach((person) -> {
                  if (person.equals(personSiblings)) {
                        Person leftSibling = person.getLeftSibling();
                        if (leftSibling != null) {
                              olderSiblings.add(leftSibling);

      getOlderSiblingsFromPerson(person.getParents(), leftSibling,
olderSiblings);
                        }
                  } else {
                        person.getUnions().forEach((union) -> {
                              getOlderSiblingsFromPerson(union,
personSiblings, olderSiblings);
                        });
                  }
            });
      } else {
            System.out.println("No descendents...");
      }
      return olderSiblings;
}
```

Listagem 10 – Método "getSiblingsFromPerson", não recursivo, contendo a lógica para obter todos os irmãos de uma pessoa, a partir da chamada de outros métodos

```
// invoking
LinkedHashSet<Person> siblings =
getSiblingsFromPerson(unionGreatGrandParents, ginaMansueli, new
LinkedHashSet<>());
for (Person sibling : siblings) {
      System.out.println(sibling);
}
```

```
LinkedHashSet<Person> getSiblingsFromPerson(Couple root, Person
personSiblings,
            LinkedHashSet<Person> siblings) {
      siblings.addAll(getOlderSiblingsFromPerson(root, personSiblings, new
LinkedHashSet<>()));
      siblings.addAll(getYoungerSiblingsFromPerson(root, personSiblings,
new LinkedHashSet<>()));
      return siblings;
}
```

Listagem 11 – Método "displayAllTwins", contendo algoritmo para exibir todos os gêmeos

```
// invoking
displayAllTwins(unionGreatGrandParents);

// in a K-Graph data structure, two or more persons are considered twins if
they are descendents of the same couple, and were born on the same day
void displayAllTwins(Couple root) {
      if (!root.getDescendents().isEmpty()) {
            ArrayList<Person> siblings = root.getDescendents();
            if (!siblings.isEmpty()) {
                  ArrayList<Person> twins = new ArrayList<>();
                  for (int i = 0; i < siblings.size() - 1; i++) {
                        Person current = siblings.get(i);
                        Person next = siblings.get(i + 1);
                        if
(next.getDateOfBirth().equals(current.getDateOfBirth())) {
                              twins.add(current);
                              twins.add(next);
                        }
                  }
                  for (Person twin : twins) {
                        System.out.println(twin);
                  }
                  root.getDescendents().forEach((person) -> {
                        person.getUnions().forEach((union) -> {
                              displayAllTwins(union);
                        });
                  });
            }
      }
}
```

Listagem 12 – Método "displayUnclesAndAuntsFromPerson", não recursivo, contendo a lógica para exibir todos os tios e tias de uma pessoa, a partir da chamada de outro método

```
// invoking
displayUnclesAndAuntsFromPerson(unionGreatGrandParents, victorMansueli);

// uncles and aunts are the brothers and sisters from each of the person's
parents
void displayUnclesAndAuntsFromPerson(Couple root, Person person) {
      if (person.getParents() != null) {
```

```
            Person father = person.getParents().getMale();
            Person mother = person.getParents().getFemale();
            LinkedHashSet<Person> unclesAndAuntsFromFather =
getSiblingsFromPerson(root, father, new LinkedHashSet<>());
            if (!unclesAndAuntsFromFather.isEmpty()) {
                  System.out.println("Uncles and aunts from father:");
            }
            for (Person uncleAunt : unclesAndAuntsFromFather) {
                  System.out.println(uncleAunt);
            }
            LinkedHashSet<Person> unclesAndAuntsFromMother =
getSiblingsFromPerson(root, mother, new LinkedHashSet<>());
            if (!unclesAndAuntsFromMother.isEmpty()) {
                  System.out.println("Uncles and aunts from mother:");
            }
            for (Person uncleAunt : unclesAndAuntsFromMother) {
                  System.out.println(uncleAunt);
            }
      }
}
```

Listagem 13 – Método "displayAllAncestorsFromPerson", contendo algoritmo para exibir todos os ancestrais (pais) de uma pessoa

```
// invoking
displayAllAncestorsFromPerson(unionGreatGrandParents, victorMansueli);

// parents of each person (recursively) to the root (first couple) of the
graph
void displayAllAncestorsFromPerson(Couple root, Person person) {
      if (person.getParents() != null) {
            Person father = person.getParents().getMale();
            System.out.println(father);
            displayAllAncestorsFromPerson(root, father);
            Person mother = person.getParents().getFemale();
            System.out.println(mother);
            displayAllAncestorsFromPerson(root, mother);
      }
}
```

Listagem 14 – Método "displayBloodRelativesFromPerson", contendo algoritmo para exibir todos os parentes consanguíneos de uma pessoa (excetuando irmãos e primos)

```
// invoking
displayBloodRelativesFromPerson(unionGreatGrandParents, victorMansueli);

// display all persons which have common ancestors, excluding siblings and
cousins from the person
void displayBloodRelativesFromPerson(Couple root, Person person) {
      if (person.getParents() != null) {
            Person father = person.getParents().getMale();
            System.out.println(father);
            LinkedHashSet<Person> siblings;
```

```
            siblings = getSiblingsFromPerson(root, father, new
LinkedHashSet<>());
            for (Person sibling : siblings) {
                  System.out.println(sibling);
            }
            displayBloodRelativesFromPerson(root, father);
            Person mother = person.getParents().getFemale();
            System.out.println(mother);
            siblings = getSiblingsFromPerson(root, mother, new
LinkedHashSet<>());
            for (Person sibling : siblings) {
                  System.out.println(sibling);
            }
            displayBloodRelativesFromPerson(root, mother);
      }
}
```

REPÚBLICA FEDERATIVA DO BRASIL
Ministério Da Indústria, Comércio Exterior e Serviços
Instituto Nacional da Propriedade Industrial
Diretoria de Patentes, Programas de Computador e Topografias de Circuitos Integrados

Certificado de Registro de Programas de Computador

Processo nº: BR 51 2018 001141-1

O Instituto Nacional da Propriedade Industrial expede o presente certificado de Registro de Programas de Computador, válido por 50 anos a partir de 1º de janeiro subsequente à data de Publicação: 26 de abril de 2018 , em conformidade com o parágrafo 2º , artigo 2º da Lei Nº 9.609, de 19 de Fevereiro de 1998.

Título: **K-Graph: Representations of Genealogies**

Data de Criação: 01 de fevereiro de 2018

Data de publicação: 26 de abril de 2018

Titular(es): MARCELO TSUGUIO OKANO
VICTOR ALEXANDRE PLOEGER MANSUELI

Autor(es): MARCELO TSUGUIO OKANO
/ VICTOR ALEXANDRE PLOEGER MANSUELI

Linguagem: JAVA

Campo de Aplicação: CO-06

Tipo Programa: AP-01

Algoritmo Hash: SHA-512

Resumo Digital: 1fb5627720bf5c12629472e93aa19c57d419cccf60208352e90727f27ab37254a19294a00e90d2a79fc23d5092d51e90160e7a78212ca90286bf086d1a8af28a

Expedido em: 17 de julho de 2018

Aprovado por Liane Elizabeth Caldeira Lage