Rin'G: Um Ambiente Não-intrusivo para Animação de Algoritmos em Grafos

Eduardo Santos[1] **, Italo Giovani**[1] **, Tays Cristina**[1] **, Fabio Tirelo**[1]

[1]Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais
Av. D. José Gaspar, 500 – Prédio 34 – 30.535-610 – Belo Horizonte, MG

escordeiro@yahoo.com, italogiovani@terra.com.br,

tayscristina@yahoo.com.br, ftirelo@pucminas.br

Abstract. *The study of graph algorithms can be difficult due to differences between classic graphical representation of graphs and computational representation of its structure. The* Reflectin'Graphs (Rin'G) *study environment is presented as a tool to ease the programmer's task of mapping the abstract concepts of graph theory into computational structures for algorithm implementation, by means of algorithm animation. In addition, the tool's architecture, the self-learning methodology it provides, the structure of Rin'G's animation-compatible algorithms and existing related works are described. The paper is concluded with a case study on the implementation of an animation-compatible depth search algorithm and the analysis of future works on Rin'G.*

Resumo. *O estudo de algoritmos em grafos pode ser difícil devido à diferença dos níveis de abstração entre a representação conceitual gráfica clássica de grafos e a representação computacional de suas estruturas. Apresenta-se o ambiente* Reflectin'Graphs (Rin'G)*, que facilita a tarefa de mapear conceitos abstratos da teoria dos grafos em estruturas computacionais para a implementação de algoritmos, por meio da animação dos mesmos. Descrevem-se a arquitetura do ambiente Rin'G, a metodologia de implementação de algoritmos que ele fornece, a estrutura de algoritmos que possam ser animados no ambiente e trabalhos relacionados. Conclui-se o artigo com um estudo de caso da implementação da Busca em Profundidade e a análise de trabalhos futuros sobre o Rin'G.*

1. Introdução

O desenvolvimento de algoritmos em grafos é um tópico de grande importância nos cursos de Ciência da Computação e Sistemas de Informação. Durante as disciplinas que solicitam aos alunos a implementação de tais algoritmos, percebe-se uma grande dificuldade oriunda da lacuna conceitual existente entre a representação clássica, gráfica, e a representação computacional, matrizes e listas de adjacência.

Ao estudarem as técnicas envolvidas na implementação de algoritmos em grafos, alunos normalmente criam, mentalmente ou através de desenhos, visualizações da execução desses algoritmos sobre grafos de exemplo, na tentativa de entender seus passos e

comparar o resultado da execução com o resultado esperado dos algoritmos. Entretanto, este método nem sempre é satisfatório em termos de compreensão de algoritmos.

Nesse artigo, apresenta-se, como proposta de solução para o problema, o ambiente *Reflectin'Graphs (Rin'G)*, que busca fornecer a estudantes de algoritmos em grafos suporte à visualização interativa da execução de seus algoritmos. O *Rin'G* cria um ambiente de estudo onde alunos das disciplinas de Grafos podem criar grafos e incluir e acompanhar passo-a-passo a execução do algoritmo em estudo.

O artigo está dividido da seguinte maneira: na Seção 2, apresenta-se uma descrição do problema a ser resolvido; na Seção 3, são mostrados os requisitos de uma solução para o problema e apresenta-se o Rin'G; na Seção 4, é apresentada a arquitetura da implementação do sistema; na Seção 5, é introduzida a metodologia de desenvolvimento de algoritmos que os alunos devem utilizar para fazerem animações; na Seção 6, a estrutura de um algoritmo em grafos é apresentada; na Seção 7, é apresentado um estudo de caso da implementação da busca em profundidade em grafos; na Seção 8, analisam-se trabalhos relacionados, comparando-os com o ambiente Rin'G; na Seção 9, é apresentada uma avaliação da ferramenta; na Seção 10, são apresentados os interesses do grupo na continuidade deste trabalho; e, finalmente, na Seção 11, são apresentadas as considerações finais do trabalho.

2. Descrição do Problema

O estudo de grafos nos cursos de Ciência da Computação e Sistemas de Informação tipicamente introduz, além do embasamento matemático, o estudo de algoritmos que permitam a manipulação da estrutura [4, 10]. A implementação de algoritmos em grafos engloba, inicialmente, a implementação de uma estrutura de grafo e um ambiente para entrada dos dados dos grafos e exibição dos resultados de algoritmos.

O esforço inerente à implementação desse amparo ao desenvolvimento de algoritmos em grafos pode desviar estudantes do foco da disciplina. O aluno preocupa-se, nesse caso, não só com as estruturas e técnicas dos algoritmos, mas também com a implementação de um ambiente gráfico ou textual que receba dados do grafo e gere saídas a partir da execução de algoritmos. A teoria dos grafos é inerentemente abstrata e de alto nível e, assim, descrições não-visuais de grafos e suas estruturas são ineficientes para a compreensão, exigindo, portanto, um maior esforço cognitivo para a correspondência entre a representação clássica gráfica e a representação numérica fornecida geralmente na forma de matrizes ou listas de adjacência.

No entanto, em cursos de graduação – em que a teoria dos grafos é normalmente ministrada em um único semestre e em conjunto com várias outras disciplinas – pode ser necessário optar entre direcionar o volume de trabalho para a implementação de algoritmos ou para a implementação de um ambiente de visualização.

Na literatura, são encontrados argumentos contrários e favoráveis a alunos programarem as visualizações [2]. Entretanto, os autores percebem, na prática da disciplina, que sobrecarregar alunos com tarefas de implementação de um ambiente de grafos pode tornar secundária a implementação de algoritmos, que deveria ser o foco da disciplina.

3. Solução Proposta

Para amparar o desenvolvimento de algoritmos em grafos, existem diversos ambientes descritos na literatura que possibilitam ao estudante preocupar-se com o desenvolvimento do algoritmo em si, desviando o mínimo possível sua atenção para a implementação do ambiente de visualização da animação do algoritmo; algumas dessas ferramentas estão descritas na Seção 8.

Além de fornecer as estruturas utilizadas pelos algoritmos a serem implementados, os ambientes de estudo de algoritmos em grafos aplicam, geralmente, metodologias didáticas para auxiliar a sedimentação do conhecimento aplicado ao desenvolvimento dos algoritmos, como animação de algoritmos ou aprendizado colaborativo [3, 8, 9, 12].

O ambiente apresentado, denominado Rin'G, busca ser um ambiente completo para o auto-aprendizado de algoritmos em grafos. Nele está reunida a maior parte das funcionalidades dos ambientes de grafos descritos na literatura - ver Seção 8 - e algumas necessidades identificadas em discussões do grupo.

São descritos a seguir os requisitos definidos inicialmente para o projeto, além dos requisitos básicos de criação, edição e leiaute de grafos. Alguns destes requisitos ainda não estão disponíveis na versão atual do ambiente. Para a primeira implementação, escolheu-se desenvolver as funcionalidades essenciais para que possam ser realizados testes de usabilidade com alunos da disciplina de grafos.

A teoria dos grafos pode ser usada na solução de problemas reais em áreas diversas. Esses problemas são, primeiramente, modelados na teoria dos grafos e, posteriormente, são aplicados algoritmos para a solução. Modelar um grafo a partir do problema significa conseguir representá-lo usando componentes e propriedades permitindo, assim, a utilização de um algoritmo da teoria dos grafos que gere solução do problema.

Executar algoritmos é um requisito essencial para aplicações que trabalhem com grafos. Para ferramentas de auxílio ao ensino da teoria dos grafos, é interessante permitir que os alunos incluam de maneira simples seus algoritmos na ferramenta para testá-los, de modo que o trabalho não fique disperso entre a implementação do algoritmo e de interface gráfica de visualização.

O suporte à animação de algoritmos permite acompanhar o efeito da execução de um algoritmo no grafo. Isso deve ser feito passo-a-passo de modo que facilite a compreensão de algoritmos.

No Rin'G, é possível criar um grafo, alterar as propriedades de seus componentes (vértices e arestas), incluir um algoritmo dinamicamente e executar esse algoritmo sobre o grafo criado. A execução do algoritmo gera uma animação baseada nos passos de sua execução de forma transparente para o usuário: ele não precisa aprender comandos de animação para animar seus algoritmos.

Focando usabilidade e personalização do ambiente, foram implementadas funcionalidades de seleção de preferências e gerência do *menu* de algoritmos. Algoritmos são adicionados pelo usuário e, portanto, o *menu* onde são alocados pode ser organizado pelo próprio usuário, em tempo de execução no Rin'G, de forma intuitiva.

Como o Rin'G foi desenvolvido para ser utilizado em ambientes acadêmicos, é

comum que estudantes utilizem instâncias do software em computadores diferentes - em casa e nos laboratórios da universidade, por exemplo. Para que o perfil do usuário esteja disponível onde ele estiver, os dados de suas preferências e os seus algoritmos do usuário são guardados em arquivos que possam ser transportados se necessário.

A animação de algoritmos pode ser feita de duas formas: *offline* e *on-the-fly*. Na animação *offline*, os algoritmos são executados integralmente e, após o término da sua execução, o usuário pode acompanhar passo-a-passo a animação; essa metodologia funciona perfeitamente para algoritmos que estejam implementados corretamente. Para algoritmos que ainda estejam em desenvolvimento, no entanto, a animação *offline* pode ser insuficiente para depuração: um exemplo de problema que pode surgir com esse tipo de animação é impossibilidade de visualizar a animação se o algoritmo estiver em *loop* infinito.

Por isso, implementou-se também a animação *on-the-fly*, ou *online*, da execução de algoritmos. Essa metodologia permite ao usuário do Rin'G depurar visualmente o seu algoritmo enquanto ele é executado, e identificar erros em um nível mais abstrato do que aquele permitido pela metodologia de depuração tradicional, baseada nos valores numéricos do grafo.

O ambiente está sendo desenvolvido totalmente em Java, para que seja independente de plataforma e para que possamos usar recursos úteis da linguagem, como reflexividade e suporte nativo a *multithreading*.

4. Arquitetura da Ferramenta

Grande parte dos requisitos identificados para o Rin'G se repetem nos vários ambientes de edição de grafos existentes na literatura. Por essa razão, desenvolveu-se um *framework*, denominado Diamante, que forma uma base de sustentação para o desenvolvimento de aplicações em grafos.

Um *framework* [7] é o esqueleto de uma implementação, a estrutura necessária para o funcionamento de um conjunto de aplicações; em um *framework* orientado por objetos, define-se um conjunto de classes que será estendido por aplicações para a implementação de seus requisitos específicos. O Diamante é um arcabouço desenvolvido para minimizar o esforço de projeto de aplicações de grafos, fornecendo um conjunto de funcionalidades comuns a essas aplicações, como a criação e edição de grafos, execução de comandos, animação de algoritmos, propriedades de componentes de grafos e outras.

Desenvolvido com base no Diamante, o Rin'G consiste em um conjunto de classes de fachada [1], ou classes para controle de funcionalidades específicas da própria interface. Para as funcionalidades específicas de grafos, o Rin'G delega a responsabilidade de execução para o Diamante. É possível também definir algoritmos em grafos que utilizem as classes do *framework* Diamante.

Esta separação em camadas, mostrada na Figura 1, facilita a reutilização do código do *framework*, enquanto operações específicas da ferramenta ficam implementadas separadamente, em outro nível da aplicação. Além disso, é importante destacar que os algoritmos são implementados de forma completamente independente do ambiente de execução. De fato, eles podem ser considerados entradas do Rin'G.

Figura 1: Arquitetura do Rin'G

5. Metodologia de Desenvolvimento de Algoritmos

Em algumas das ferramentas estudadas, o desenvolvimento de algoritmos compatíveis com o funcionamento da ferramenta implica no aprendizado de uma nova linguagem ou na inclusão de código específico da ferramenta no código do algoritmo. Pretende-se, no Rin'G, tornar mínima a necessidade de conhecer a estrutura interna do ambiente durante o desenvolvimento do algoritmo, de forma que as operações do ambiente não sejam intrusivas.

O *framework* Diamante fornece suporte a algoritmos e comandos, de forma que a animação de algoritmos em grafos seja baseada nos comandos executados pelo algoritmo do aluno. Estes comandos são criados pelo Rin'G, de forma transparente ao usuário, quando são realizadas operações sobre o grafo, tais como alterações de propriedades de vértices e arestas, ou testes sobre propriedades do grafo, tais como testes de existência de arestas.

A adaptação feita pelo usuário, ao desenvolver um algoritmo compatível com o Rin'G, se dá apenas no conhecimento das estruturas do *framework* Diamante, conforme descrito na Seção 6. Embora a estrutura da classe que contenha o algoritmo e o acesso a propriedades de componentes do grafo sejam dependentes do Diamante, o algoritmo em si é totalmente desenvolvido em Java, sem a necessidade do aprendizado de uma nova linguagem ou da inclusão de código especial para a sua animação.

6. Estrutura de Algoritmos

Algoritmos compatíveis com o Rin'G seguem o formato definido pelo *framework* Diamante. Como esses algoritmos são desenvolvidos em Java, não é necessário aprender uma linguagem específica para implementar algoritmos utilizando o Rin'G.

Para ser adicionado ao Rin'G, um algoritmo deve obedecer as normas definidas no *framework* Diamante: deve estar implementado em uma subclasse de *Algorithm* e implementar, além de um construtor, um método de nome *execute*. Os parâmetros que o algoritmo utiliza são parâmetros do construtor da classe.

Quando um algoritmo é incluído no Rin'G, a ferramenta analisa o construtor da classe via reflexividade [6] e, antes do algoritmo ser executado, solicita ao usuário os parâmetros que ele requer. Durante a execução do algoritmo, propriedades de componentes do grafo podem ser alteradas, e essas operações serão monitoradas pelo Rin'G para gerar passos da animação da execução do algoritmo, de forma transparente para o usuário.

7. Estudo de Caso

Essa seção demonstra a implementação no Rin'G de um algoritmo clássico da teoria dos grafos: a Busca em Profundidade.

A Busca em Profundidade tem por objetivo visitar todos os vértices alcançáveis a partir de um vértice de origem. A estratégia utilizada por esse algoritmo, como o nome sugere, é visitar primeiro as arestas adjacentes ao vértice mais recentemente descoberto e que levem a vértices ainda não visitados, percorrendo assim caminhos em profundidade no grafo [11].

7.1. Escrita do Algoritmo

A escrita de algoritmos não é realizada no ambiente. Algoritmos podem ser desenvolvidos no ambiente de desenvolvimento preferido do programador. Os algoritmos são incluídos na forma de código-objeto (arquivo *.class*, em Java) no Rin'G, após terem sido compilados.

7.2. Pacotes Utilizados e Construtor da Classe

Para utilizar a estrutura do *framework* Diamante, é necessário importar os pacotes *diamante.core.command* e *diamante.core.graph*. O algoritmo do usuário deve estar na pasta *user* que acompanha o Rin'G e, portanto, o pacote declarado para o algoritmo deve ser *user*.

O construtor da classe do algoritmo deve receber pelo menos um parâmetro: o grafo sobre o qual o algoritmo será executado. Além desse parâmetro, na versão atual do ambiente, podem ser utilizados parâmetros inteiros, que são índices de vértices usados pelo algoritmo. O Rin'G analisa a assinatura do construtor via reflexividade e, antes de criar uma instância da classe do algoritmo, solicita ao usuário que selecione os vértices requisitados pelo algoritmo. Esse código inicial do algoritmo, juntamente com o esqueleto do restante da classe, está mostrado na Figura 2; os códigos dos métodos *execute* e *visit* são mostrados nas Figuras 3 e 4.

A primeira linha do construtor do algoritmo deve chamar o construtor da superclasse *Algorithm*, para que sejam guardados a referência para o grafo e o nome do algoritmo.

7.3. O Método *execute*

Como a implementação do algoritmo compatível com o Rin'G exige que a classe estenda *Algorithm*, é necessário implementar o método *execute*. Esse método não recebe parâmetros, não possui valor de retorno e é chamado para iniciar a execução do algoritmo. A implementação do método *execute* desse algoritmo de exemplo está mostrada na Figura 3.

7.4. O Método *visit*

A implementação do método *execute* chama um outro método da classe *DepthSearch*: *visit*. Esse método é recursivo e visita em profundidade os vértices adjacentes ao vértice passado como parâmetro. A implementação do método *visit* altera propriedades dos

```
package user;
import diamante.core.command.*;
import diamante.core.graph.*;

public class DepthSearch extends Algorithm {
  boolean visited [];
  int startNode;

  public DepthSearch (Graph g, int startNode) {
    super(g,"DepthSearch");
    this.startNode = startNode;
  }
  public void execute() {
    (...)
  }
  void visit (int v) {
    (...)
  }
}
```

Figura 2: Código de inicialização do algoritmo e esqueleto da classe

```
public void execute() {
  visited = new boolean[g.getN()];
  visit(startNode);
}
```

Figura 3: Código do método *execute*

vértices e arestas, colorindo-os para indicar o caminho percorrido na busca. São estas alterações de propriedades que são convertidas pelo Rin'G em passos da animação.

A implementação do método *visit* pode ser vista na Figura 4. É importante observar que, embora o programador do algoritmo tenha que se preocupar com a exibição de um resultado do algoritmo por meio das alterações de propriedades, essa preocupação é inerente a qualquer implementação do algoritmo. Isso se dá porque, embora a busca em profundidade funcione perfeitamente sem uma saída de resultados, não é interessante didaticamente executar um algoritmo sem poder visualizar o seu resultado.

7.5. Inclusão e Execução do Algoritmo no Ambiente

Uma vez que o algoritmo tiver sido compilado, o aluno pode adicionar o arquivo compilado, de extensão *.class*, de maneira semelhante a uma abertura de arquivos via caixa de diálogo. Internamente, o ambiente o adiciona ao *menu* de algoritmos disponíveis, de modo que a sua execução possa ser iniciada diretamente a partir do *menu* gerado.

Ao escolher o *menu* para executar o algoritmo, uma nova instância da classe do algoritmo é criada por meio de reflexividade computacional de Java [6]. Na criação da instância, a lista de parâmetros do construtor é analisada para permitir que o usuário selecione, na interface gráfica, os vértices que o algoritmo exigir como parâmetros - ver exemplo na Seção 7.2.

```
void visit (int v) {
  visited[v] = true;
  this.setNodeProperty(v,"ComponentColor","0,255,0");

  for (int w=0 ; w<g.getN() ; w++)
    if(g.e(v,w) && !visited[w]) {
      this.setEdgeProperty(v,w,"ComponentColor","0,0,255");

      visit(w);
    }
}
```

Figura 4: Código do método *visit*

Uma vez instanciada a classe, o método *execute* é chamado para que o algoritmo inicie. É importante observar que, uma vez escrito o algoritmo, as inclusões e execuções do algoritmo são feitas sem que o aluno precise editar o código da interface do ambiente.

7.6. Exibição da Animação

Ao fim da execução, uma janela é aberta para exibir a animação do algoritmo - um passo da animação do algoritmo de exemplo está mostrado na Figura 5. Nessa janela, há os controles usuais em visualização de mídia, como *reproduzir*, *pausar*, *avançar* e *retroceder*. É possível navegar progressiva ou regressivamente pelos passos da animação do algoritmo por meio destes controles.

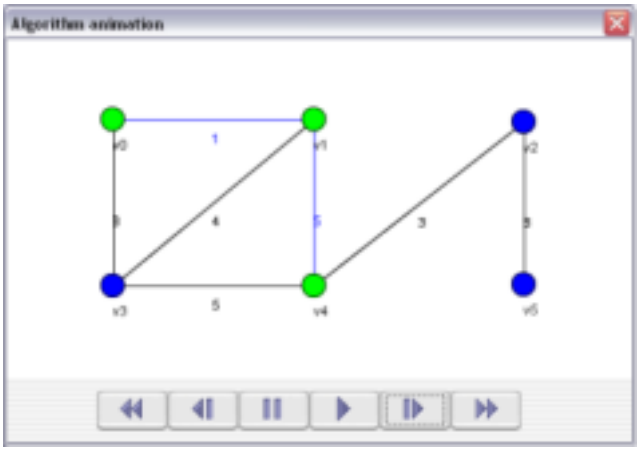

Figura 5: Janela de animação de algoritmo

Na versão atual do Rin'G, é possível animar qualquer operação que altere o grafo ou suas propriedades.

7.7. Resultado da execução do algoritmo

As alterações de propriedades geram um efeito colateral transparente para o programador: o Rin'G transforma tais alterações em passos da animação do algoritmo. Essa animação é feita *on-the-fly*, permitindo o acompanhamento da execução do algoritmo durante a sua execução, objetivando uma maior qualidade da ferramenta como instrumento de depuração visual de algoritmos.

Uma observação crucial sobre a implementação de algoritmos compatíveis com a animação do Rin'G: a ordem das alterações de propriedades afeta a ordem dos passos da animação. Essa correspondência entre a ordem de alterações de propriedades e a ordem dos passos da animação é interessante por manter a visualização mental da execução do algoritmo criada pelo programador durante o desenvolvimento, e leva a depuração do algoritmo a um nível visual mais abstrato e próximo das metáforas conceituais criadas pelo programador.

8. Trabalhos Relacionados

Na literatura são descritos vários ambientes para visualização e animação de algoritmos em grafos. Esses ambientes são, muitas vezes, implementados atendendo a necessidade dos professores das disciplinas de grafos de diferentes universidades, e, assim, fornecem soluções para os problemas específicos dos alunos no contexto do desenvolvedor da ferramenta. O estudo da literatura levou a verificar requisitos e funcionalidades existentes em ferramentas criadas por outros grupos, para que seja possível levantar o maior número de requisitos possível para o Rin'G, e, assim, atenter às necessidades dos estudantes da disciplina de lugares e contextos diversos.

Inúmeras ferramentas foram propostas para permitir a visualização da execução de algoritmos e, em especial, algoritmos em grafos e visualização de grafos. Dentre as ferramentas analisadas, destacam-se o EVEGA [8], o DisViz [9], o DIDAGRAPH [12], o VisualGraph [3] e o JEliot [5].

O EVEGA é uma ferramenta desenvolvida para o aprendizado de algoritmos em grafos, e fornece uma interface poderosa e de boa usabilidade para criação e edição interativa de grafos e animação de algoritmos. As execuções de vários algoritmos podem ser comparadas por meio de gráficos de desempenho gerados pela ferramenta. Um algoritmo de fluxo máximo acompanha a ferramenta para demonstração e é possível incluir no ambiente novos algoritmos. A operação do EVEGA, no entanto, para geração da animações, é feita de forma intrusiva no código do algoritmo, e a interface de animação de algoritmos é pouco intuitiva, e não permite voltar a execução a passos anteriores, como ocorre no Rin'G.

O DisViz foi desenvolvido para permitir o aprendizado de grafos em grupo, por meio de redes locais, em um ambiente P2P no qual é possível criar grafos de teste e animações de algoritmos que possam ser visualizadas por todos os membros da rede. Esta ferramenta, no entanto, não fornece uma maneira fácil de acoplamento de algoritmos desenvolvidos por usuários, de forma que alunos só podem utilizar os algoritmos já presentes. O estudo de algoritmos em grafos utilizando o DisViz, no entanto, é restrito aos algoritmos já implementados na ferramenta e ao acompanhamento da animação desses algoritmos. Além disso, o estudo depende da existência de uma rede local e da colaboração dos membros dessa rede, o que pode dificuldar o estudo fora dos horários de aulas práticas. Com o Rin'G, o aluno pode desenvolver seus trabalhos a qualquer momento sem que haja a necessidade de montar uma rede de computadores, além de ser possível acoplar seus próprios algoritmos na ferramenta sem alterar o seu código fonte.

O DIDAGRAPH permite o acompanhamento de animações da execução de algoritmos, em alto nível, por meio de uma visualização do grafo com o algoritmo em

andamento e de uma descrição em linguagem de alto nível do algoritmo. Esta ferramenta, atualmente, conta apenas com poucos algoritmos, porém futuramente permitirá que usuários criem seus algoritmos em uma linguagem de comandos para grafos, através da qual animações serão geradas. Para estudar algoritmos utilizando esta ferramenta, é necessário aprender uma nova linguagem, fato que pode desviar a atenção do objetivo das disciplinas de grafos. No Rin'G, o aluno desenvolve o algoritmo em Java e não precisa misturar códigos de visualização ou de animação na sua implementação.

O VisualGraph fornece uma biblioteca para a animação de algoritmos em grafos. Essa biblioteca, no entanto, não fornece uma interface gráfica para o usuário, servindo apenas como base para a criação de grafos que possam ser utilizados em outros projetos e ambientes. Para visualizar a animação gerada por algoritmos criados com o VisualGraph, é necessário o uso de outra ferramenta. A interface do VisualGraph para criação de grafos é baseada em texto e pouco intuitiva. Além disso é necesário fazer chamadas a funções de biblioteca dentro do código do aluno. No Rin'G, o aluno utiliza somente um ambiente que integra visualização e animação do algoritmo, além de não ser necessário incluir código de animação na sua implementação.

O JEliot é uma ferramenta de visualização de programas, e exibe uma animação da execução de um programa. O código do programa é totalmente independente da ferramenta, e o usuário deve apenas indicar, na própria ferramenta, quais estruturas de dados serão animadas, e ela se encarrega da animação das operações realizadas sobre aquela estrutura durante a execução do programa. Este ambiente propõe a visualização genérica de programas, e não é específica para algoritmos em grafos. Por esse motivo, as visualizações que ela gera não necessariamente correspondem às visualizações conceituais de grafos e, assim, ela ainda exige um esforço mental do programador na conversão das representações do ambiente e a representação visual de grafos. O Rin'G, por outro lado, é voltado para visualização de algoritmos em grafo, o que torna a visualização mais próxima do esperado pelos alunos de grafos.

9. Avaliação

As ferramentas de auxílio visual à programação dividem-se em dois grupos principais: intrusivas e não-intrusivas [2].

Ferramentas intrusivas exigem que o programador insira código específico de visualização em seu programa ou algoritmo. Para ferramentas educacionais, a intrusão do código pode desviar a atenção do programador do seu objetivo de aprender a implementar algoritmos básicos, e, por isso, julgamos que elas se tornam desinteressantes para o uso didático. Além disso, essas ferramentas ferem o Princípio da Incerteza de Heisenberg, já que, para que uma animação seja possível, elas exigem que código de animação seja incluído no código do algoritmo; porém, a inclusão de código de animação pode influenciar no processo de desenvolvimento e aprendizado do próprio algoritmo, causando assim um efeito colateral no uso do ambiente.

Ferramentas não-intrusivas conseguem analisar o código do usuário de forma transparente, e gerar visualizações dinâmicas a partir desse código. Do ponto de vista didático, essas ferramentas são interessantes porque permitem que o usuário preocupe-se apenas com o problema que tem em mãos, e utilize a ferramenta como auxílio natural ao

desenvolvimento.

O Rin'G é não-intrusivo no código dos algoritmos do usuário. Ele permite que algoritmos, desenvolvidos sob a estrutura do *framework* Diamante, sejam implementados sem a preocupação com interfaces gráficas ou o aprendizado de linguagens específicas do ambiente e acoplados ao ambiente para animação.

10. Trabalhos Futuros

O Rin'G está em processo contínuo de desenvolvimento e, por isso, alguns dos requisitos inicialmente levantados ainda não estão implementados na versão atual; o processo de desenvolvimento da ferramenta está sendo realizado de forma incremental. Algumas das funcionalidades a serem incluídas estão descritas a seguir.

A teoria dos grafos pode ser aplicada a um vasto conjunto de áreas do conhecimento humano. Entretanto, a representação na teoria dos grafos desses problemas pode ser de difícil reconhecimento. Para simplificar a modelagem de problemas via teoria dos grafos, o grupo pretende implementar suporte à modelagem de problemas na ferramenta, na tentativa de simplificar esse processo.

O grupo discute ainda uma metodologia de desenvolvimento de algoritmos em grafos baseada em estruturas matemáticas de alto nível, tais como conjuntos, o que dará ao programador de algoritmos para o Rin'G um nível maior de abstração durante a implementação de seus algoritmos, mais próximo da teoria dos grafos. Espera-se que, com o alto nível de abstração que pode ser atingido com essa metodologia, o processo de aprendizado de algoritmos em grafos torne-se mais simples.

A visualização de grafos e animações de algoritmos são recursos interessantes utilizados pelo Rin'G. A representação interna desses recursos, no entanto, não é compatível com formatos comumente utilizados em outros ambientes, como páginas na *web* e documentos de texto. A exportação desses recursos para formatos externos permitirá que grafos, animações de algoritmos e outros recursos gerados no Rin'G sejam utilizados em outros ambientes.

11. Conclusões

Foi apresentado nesse artigo o ambiente de desenvolvimento de algoritmos em grafos denominado *Reflectin'Graphs (Rin'G)*. No Rin'G, é possível criar, alterar propriedades de componentes de grafos e incluir algoritmos criados pelo usuário. A execução de algoritmos no Rin'G gera uma animação, que pode ser acompanhada passo-a-passo, com o objetivo de fornecer ao programador uma abstração visual próxima à utilizada na teoria dos grafos e, assim, auxiliá-lo no desenvolvimento e compreensão do algoritmo.

O Rin'G é uma ferramenta para o estudo de algoritmos em grafos, e auxilia o aprendizado desses por prover um ambiente no qual estudantes podem simplesmente desenvolver seus algoritmos, sem maiores preocupações com entrada de dados de grafos e exibição de resultados de algoritmos, ou a implementação da representação computacional de grafos. A qualidade da ferramenta no auxílio ao aprendizado está em prover um

ambiente no qual as estruturas de grafos e uma interface gráfica já estão desenvolvidos, e o aluno concentra-se apenas no desenvolvimento de seus algoritmos.

A ferramenta é um complemento a disciplinas de teoria dos grafos, e não uma metodologia completa de auto-aprendizado - estudantes podem, de fato, aprender com a ferramenta e sedimentar seus conhecimentos sobre os algoritmos implementados, porém é necessário o aprendizado prévio de teoria de grafos, suas estruturas, propriedades e algoritmos para que o uso do Rin'G seja proveitoso.

Referências

[1] Erich Gamma ET AL. *Design Patterns*. Addison-Wesley, 1995.

[2] C.D. Hundhausen, S. A. Douglas, and J. T. Stasko. A Meta-study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing*, 13:259–314, 2002.

[3] Jeffrey M. Lucas, Thomas L. Naps. VisualGraph - A Graph Class Designed For Both Undergraduate Students and Educators. *SIGCSE 2003, Special Interest Group on Computer Science Education, Reno - Nevada, USA*, 2003.

[4] John T. Gorgone ET AL. Model Curriculum and Guidelines for Undergraduate Degree Programs in Information System. Technical report, Association for Computing Machinery (ACM), 2002.

[5] Jorma Tarhio Matti Lattu, Veijo Meisalo. On using a Visualization Tool as a Demonstration Aid - http://www.cs.helsinki.fi/research/aaps/Jeliot/. Technical report, 2003.

[6] Sun Microsystems. Especificação da API *java.lang.reflect*. Disponível em java.sun.com/j2se/1.4.2/docs/api/java/lang/reflect/package-summary.html. Último acesso em Abril de 2004.

[7] Mohamed E. Fayad. Building Application Frameworks: Object-Oriented Foundations of Framework Design. *John Wiley & Sons New York*, 1999.

[8] Sami Khuri, Klaus Holzapfel. EVEGA: An Educational Visualization Environment for Graph Algorithms. *Proceedings of ITiCSE 2001, The 6th Annual Conference on Innovation and Technology in Computer Science Education - Canterbury, UK 2001*, 2001.

[9] Alexander A. Sherstov. Distributed Visualization of Graph Algorithms. *SIGCSE 2003, Special Interest Group on Computer Science Education, Reno - Nevada, United States*, 2003.

[10] Sociedade Brasileira de Computação. Currículo de Referência para Cursos de Bacharelado em Sistemas de Informação. Technical report, SBC, Diretoria de Educação, 2002.

[11] Thomas H. Cormen ET AL. *Algoritmos: teoria e prática*, chapter 22, pages 429–434. Editora Campus, 2002. Tradução da 2ª edição americana.

[12] V. Dagdilelis,M. Stratzemi. DIDAGRAPH: Software for Teaching Graph Theory Algorithms. *ITiCSE 1998. Integrating Technology into Computer Sciente Education.Dublin - Ireland 1998*, 1998.