Raciocínio Baseado em Casos para auxílio a Alunos na Resolução de Problemas por Analogia – Uma abordagem para Representação e Recuperação de Casos

Gilson Pereira dos Santos Júnior[1], **Evandro de Barros Costa**[1,2], **Joseana Macêdo Fechine**[1]

[1]Departamento de Sistemas e Computação
Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 – 58.109-970 – Campina Grande – PB – Brasil
[2] Instituto de Computação. Universidade Federal de Alagoas (UFAL)

{gilson@dsc.ufcg.edu.br, ebcosta@gmail.com, joseana@dsc.ufcg.edu.br}

Abstract. *Experts programmers, constantly, recover to past solutions to solve new problems, avoiding the creation of a solution without a priori information. However, beginner programming students have difficulties in realizing the similarities between the problems, resulting in little reuse of solutions. To help the student to improve the skills for resolving problems through analogies we propose a system of cases based reasoning (CBR) that recovery programming problems similar to the one the student is solving. For the development of the CBR is essencial to represent the case using relevant attributes related to the domain been considered and examine the recovery of appropriate algorithms to this representation. This work presents a report on the representation and retrieval of cases in field-solving programming.*

Resumo. *Programadores experientes recorrem, constantemente, a soluções passadas e as adaptam para solucionar novos problemas, evitando, desta forma, a criação de uma solução sem informações a priori. Entretanto, alunos iniciantes em programação sentem dificuldades em perceber as analogias entre os problemas, resultando em pouca reutilização de soluções. Para auxiliar o aluno no aprimoramento das habilidades de resolução de problemas por meio de analogias é proposta a utilização de um sistema de raciocínio baseado em casos (RBC) que recupera problemas de programação similares ao que o aluno está resolvendo. Para o desenvolvimento desse RBC é primordial representar adequadamente o caso, a partir de atributos significativos ao domínio tratado, e analisar os algoritmos de recuperação adequados a esta representação. Este trabalho apresenta um relato sobre a representação e recuperação de casos no domínio de resolução de problemas de programação.*

1. Introdução

Programadores experientes recorrem, constantemente, a soluções passadas e as adaptam para solucionar novos problemas, evitando, desta forma, a criação de uma solução sem informações *a priori*. Esta heurística, utilizada pelos programadores para solucionar os problemas de programação, é denominada de raciocínio por analogia [de Barros et al. 2005].

Embora o raciocínio por analogia seja natural para os programadores experientes, alunos iniciantes sentem dificuldade em identificar semelhanças entre os problemas apresentados nas disciplinas de introdução à programação [Muller 2005] e, conseqüentemente, em empregar o raciocínio por analogia na resolução destes problemas.

Alguns trabalhos apontam a capacidade limitada dos alunos iniciantes de perceber analogias entre problemas como uma das principais causas das dificuldades na resolução de problemas de programação [Muller 2005, de Barros et al. 2005, Porter and Calder 2003, Proulx 2000]

Além disso, diversos autores destacam que há uma carência de ferramentas que dêem suporte à resolução de problemas por analogia e auxiliem o aluno na identificação de similaridades entre os mesmos.

Diante do exposto, está sendo desenvolvido um ambiente virtual para auxílio ao ensino de programação constituído, dentre outros elementos, por um sistema de raciocínio baseado em casos, para apoiar o aluno, iniciante em programação, na resolução de problemas por meio do raciocínio por analogia.

Assim, ao ser proposto um novo problema de programação a ser resolvido pelo aluno, o RBC apresenta uma coleção de problemas previamente respondidos pelo estudante, com diferentes graus de similares em relação ao atual. O ambiente, por meio de estratégias de interação, irá auxiliar o aluno na reflexão de quais dos problemas apresentados possuem maior grau de similaridade com o problema atual e, portanto, podem ser re-utilizados na solução do problema atual.

O raciocínio baseado em casos é uma técnica da Inteligência Artificial, inspirada no raciocínio por analogia, que tenta resolver novos problemas adaptando soluções passadas.

O ciclo clássico de um sistema de raciocínio baseado em casos foi proposto por [Aamodt and Plaza 1994]. Este ciclo é composto por 4 (quatro) fases, sendo estas:

- Recuperação - recupera um conjunto de casos similares da memória;
- Reuso - adapta os casos recuperados a fim de solucionar o problema atual;
- Revisão - revisa e testa a solução proposta;
- Retenção - armazena a solução revisada;

Um caso, por sua vez, é um registro de uma experiência, uma determinada situação. Normalmente, ele é constituído pela descrição do problema que foi resolvido e a solução empregada.

O objetivo deste artigo é apresentar os aspectos pertinentes à representação do caso, considerando o domínio de programação, e a fase de recuperação do sistema de raciocínio baseado em casos.

A implementação da representação e a recuperação do caso no domínio é fundamental na construção das demais fases do ciclo de um RBC, uma vez que com o cumprimento desta etapa ficam definidos: o caso, seus atributos, sua forma de indexação, como estes estarão organizados na memória, como serão recuperados e como será efetuado o cálculo de similaridade entre os casos. Além disso, a fase de reuso, próxima fase do ciclo de um RBC, necessita dos casos recuperados da memória para realização das devidas adaptações para construção da solução do novo problema.

As demais seções deste artigo estão estruturadas da seguinte forma: na Seção 2 é

apresentado os trabalhos relacionados. Na Seção 3 é descrita visão geral do ambiente. Na Seção 4, é descrita a metodologia utilizada na realização do experimento, que representa um caso de estudo. Na Seção 5, são apresentados e analisados os resultados obtidos. Por fim, na Seção 6 são apresentadas as considerações finais e os trabalhos futuros.

2. Trabalhos Relacionados

2.1. Aprendizagem Baseada em Casos - Um Ambiente de Ensino de Lógica de Programação

[Koslosky 1999] desenvolveu um ambiente de aprendizagem de lógica de programação utilizando a metodologia de raciocínio baseado em casos.

Neste trabalho, a descrição do problema do caso é dada pelo enunciado do problema, enquanto que a solução é composta por: algoritmo da solução, quantidade de instruções (QI), quantidade de variáveis (QV), quantidade de estruturas de repetição (QER) e seleção (QES), a justificativa do aluno para os elementos utilizados e os comentários do professor sobre a solução.

Os casos são recuperados calculando-se a similaridade a partir do enunciado do problema e dos valores para QI, QV, QER e QES. A similaridade local é calculada pela distância entre os atributos numéricos. Koslosky define pesos para cada um dos atributos para o cálculo da similaridade global.

Os atributos para representação e indexação dos casos utilizados por Koslosky torna a recuperação pouco eficiente, uma vez que a similaridade entre os casos leva em consideração apenas o casamento do enunciado do problema e atributos que representam a estrutura do código.

2.2. ProPAT *e-learning tool*

O ProPAT [de Barros et al. 2005, Delgado 2005] é um *plugin* para o Eclipse IDE que permite a estudantes iniciantes do curso de programação aprender a solucionar problema (programar) por meio da utilização de padrões elementares.

Esta ferramenta adiciona ao ambiente do Eclipse duas novas perspectivas: a *Teacher Perspective* e a *Student Perspective*. A *Teacher Perspective* permite ao professor especificar novos exercícios e padrões elementares que serão apresentados na perspectiva do aluno. A *Student Perspective* é o ambiente de trabalho do aluno. Nesta perspectiva, o aluno poderá visualizar os problemas e os padrões elementares cadastrados pelo professor e codificar sua solução.

2.3. *Pattern-Oriented Instruction*

O *Computer Science Group* da Universidade de Tel-Aviv, em Israel, publicou alguns trabalhos [Muller et al. 2004, Muller 2005, Muller et al. 2007] importantes com relação à resolução de problemas de programação utilizando o raciocínio por analogia por meio do uso de padrões algorítmicos.

Este grupo definiu 30 (trinta) padrões algorítmicos para o ensino de programação e avaliou o desenvolvimento cognitivo dos estudantes na resolução de problemas, por meio do raciocínio por analogia.

Embora o grupo tenha experiência na utilização de padrões no contexto presencial, até o momento, não apresentaram a comunidade nenhuma ferramenta que auxilie a utilização destes padrões.

3. Visão Geral do Ambiente

Para auxiliar o aluno no raciocínio por analogia para resolução de problemas de programação, O ambiente virtual para auxílio ao ensino de programação aqui apresentado é composto por um sistema de raciocínio baseado em casos e um agente, conforme ilustrado na Figura 1. O objetivo do agente neste ambiente é executar estratégias de interação com o aluno para auxiliá-lo na reflexão sobre quais problemas recuperados pelo RBC possuem maior grau de similaridade com o problema atual e quais características levam tais problemas a serem similares.

Os casos recuperados, com diferentes níveis de similaridade, e a interação entre o aluno e o agente possibilitam que o referido aluno perceba quais características fornecem indícios dos problemas mais adequados para reutilização na solução de um dado problema. É importante ressaltar, que a reutilização de soluções é uma prática fortemente indicada na engenharia de software [Pressman and Pressman 2004, Sommerville 2004].

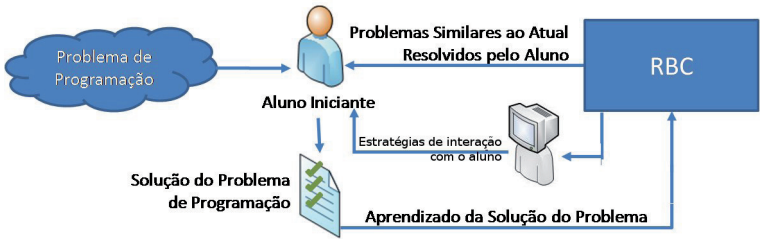

Figura 1. Visão Geral da Solução Proposta

De acordo com a Figura 1, no instante em que um novo problema de programação é apresentado ao aluno, para sua resolução, o sistema de raciocínio baseado em casos recupera automaticamente os problemas previamente resolvidos pelo aluno, com diferentes níveis de similares em relação ao atual. A partir dos problemas recuperados, o aluno desenvolve uma solução para o problema atual, a qual é posteriormente aprendida/apreendida pelo sistema.

Para o desenvolvimento do sistema de raciocínio baseado em casos, no domínio de programação, está sendo utilizado o *framework* JColibri 2.0 [Gaia 2002, Díaz-Agudo et al. 2007, Bello-Tomás et al. 2004]. Esse *framework* é *open-source*, orientado a objetos, e foi desenvolvido em Java pelo grupo GAIA[1]. Além disso, está sendo utilizado o sistema gerenciador de banco de dados (SGBD) PostGreSQL na versão 8.3 [Postgresql 2003] para o armazenamento dos casos do RBC.

3.1. Representação do Caso

O caso, para um sistema de raciocínio baseado em casos, é um registro de uma experiência armazenada na memória. Ele é comumente representado pela descrição de um problema e da forma como este foi solucionado.

No contexto de resolução de problemas de programação, o caso pode ser representado pelo problema de programação e pela solução algorítmica empregada pelo programador para solucioná-lo.

Para este sistema, o problema de programação é descrito da seguinte forma: pelo seu enunciado (*description*), pelos padrões de programação sugeridos pelo professor para

[1]GAIA (*Group for Artificial Intelligence Applications*) Grupo do Departamento de Engenharia de Software e Inteligência Artificial da Universidade de Madrid.

solucioná-lo (*patterns*), pela complexidade (*complexity*) e pela categoria (*category*). A solução (*solution*), por sua vez, é o algoritmo desenvolvido pelo aluno para solucionar tal problema. O diagrama de classes apresentado na Figura 2 ilustra esta representação do caso.

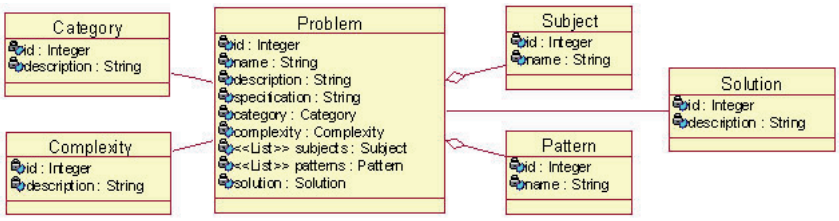

Figura 2. Representação do caso

Os padrões para alunos iniciantes de programação foram adicionados na representação do caso, uma vez que a solução de um problema de programação pode ser construída a partir da combinação de padrões, aninhando ou encadeando. Os padrões são exemplos de soluções elegantes [2] e eficientes [3] .

Além disso, diversos trabalhos [Delgado 2005, de Barros et al. 2005, Muller et al. 2004, Muller et al. 2007, Porter and Calder 2003, Proulx 2000] indicam a utilização desses padrões como uma alternativa para empregar o raciocínio por analogia no ensino de programação para alunos iniciantes.

A memória de casos do RBC é alimentada com 200 problemas de programação (casos), conforme a estrutura apresentada na Figura 2. Cada um dos problemas está classificado de acordo com a categoria, a complexidade e os padrões utilizados. Para isso, os níveis de complexidade definidos foram: fácil, médio e difícil. Em relação à categoria, os problemas estão classificados como: matemáticos, de sistema de informação (SI) ou de jogos [Mendonça 2008] (ver Tabela 1).

Tabela 1. Descrição do problema consultado no RBC.

Categoria	Descrição	Exemplos
Matemáticos	Problemas relacionados ao domínio da matemática.	Fatorial, Equação de Segundo Grau, número primo, entre outros.
Jogos	Problemas de diferentes domínios e que se apresentam como jogos.	Torre de Hanoi, Missionários e Canibais, Forca, Damas, entre outros.
SI	Problemas que automatizam as atividades operacionais.	Controle de Pessoal, Controle de Linha de Produção, Controle de Estoque, etc.

No contexto dos alunos iniciantes em programação, os padrões de programação são divididos em duas categorias: padrões elementares e padrões algorítmicos.

- Padrões elementares: são padrões simples, sintéticos e baseados na estrutura. Estes padrões se referem a operações como seleção, repetição e manipulação de dados [Delgado 2005, de Barros et al. 2005, Muller et al. 2004].
- Padrões algorítmicos: são padrões que constituem a base algorítmica da solução do problema. Estes padrões são soluções básicas de problemas como determinação de um padrão de busca, obtenção de valores extremos e checagem da validade dos valores [Muller et al. 2004, Muller 2005].

[2]Soluções elegantes são soluções em que o código é transparente e de fácil entendimento.

[3]Soluções eficientes são soluções em que o código realiza o processamento com um bom desempenho.

No Código 1, está ilustrada a estrutura do padrão elementar repetição com sentinela.

Código Fonte 1. Padrão Elementar de Repetição com Sentinela

```
1 Inicializar variável sentinela;
2 Enquanto (variável sentinela satifaz a condicao) faça inicio
3     Processar o elemento;
4     Atualizar a variável sentinela;
5 fim;
```

No Código 2, está ilustrada a estrutura de um padrão algorítmico utilizado para encontrar o maior valor de um conjunto de dados.

Código Fonte 2. Padrão Algorítmico de Maior Valor

```
1 Inicializar maior_Valor com o valor do primeiro elemento da lista;
2    Enquanto (ainda existir elemento na lista) faça
3       valor recebe o valor do próximo elemento da lista;
4       Se o valor é maior do que o maior_valor
5          O maior_valor recebe valor;
```

Nesse trabalho, foram utilizados 15 (quinze) padrões de programação para iniciantes, 5 (cinco) padrões elementares e 10 (dez) padrões algorítimicos, como mostrado na Tabela 2. Tais padrões foram definidos nos trabalhos de Orna Muller [Muller et al. 2004, Muller 2005] sobre ensino de programação utilizando padrões e no ProPAT [Delgado 2005, de Barros et al. 2005].

Tabela 2. Lista de Padrões para Alunos Iniciantes em Programação.

Nome	Descrição	Padrão
Algum valida	Padrão que verifica se ao menos um elemento da coleção atende a uma determinada condição.	Algorítmico
Buscar item	Padrão de busca de itens em uma coleção.	Algorítmico
Par	Padrão para indicar se um número é par ou impar.	Algorítmico
Primo	Padrão para indicar se um número é primo.	Algorítmico
Inverter a posição do item	Padrão de inversão da posição dos elementos em uma coleção.	Algorítmico
Lista ordenada	Padrão para verificar se uma coleção está ordenada.	Algorítmico
Ordenar lista	Padrão para ordenar uma coleção.	Algorítmico
Repetição com indicador de passagem	Padrão de repetição em que ao fim do *loop* uma variável indica se uma determinada condição foi atendida.	Elementar
Seleção simples	Padrão que representa um comando de seleção simples.	Elementar
Seleção em conjunto	Padrão que representa uma seleção dentre de um conjunto de possibilidades.	Elementar
Repetição com sentinela	Repetição que utiliza um *flag* como condição de parada.	Elementar
Repetição contada	Repetição executada *n* vezes. No qual, *n* é a quantidade de repetições previamente conhecida.	Elementar
Todos validam	Verificar se todos os elementos da coleção atendem a uma determinada condição.	Algorítmico
Trocar a posição de um item	Padrão que troca a posição de um dado item em uma coleção.	Algorítmico

3.2. Recuperação dos Casos

A Recuperação é a primeira fase executada no ciclo de um sistema de raciocínio baseado em casos e consiste em encontrar, dentro da memória de casos, aqueles que são similares ao caso corrente.

Os critérios de seleção dos casos estão inteiramente ligados à forma como eles foram representados e indexados na memória. Assim, o julgamento de que a recuperação é apropriada depende diretamente do domínio tratado e da sua organização.

Os métodos de recuperação mais comumente utilizados em RBC, nos diferentes domínios, são o *k-Nearest Neighbors* (k-NN) e as árvores de decisão [Wangenheim 2003].

Neste trabalho foi utilizado o algoritmo k-NN. Na recuperação, utilizando o algoritmo k-NN, um caso é recuperado quando a similaridade global deste em relação ao caso apresentado ao RBC é superior aos demais existentes na base.

A similaridade global é dada pela média ponderada das similaridades locais, ajustadas pelos pesos de cada atributo, como apresentado na equação a seguir.

$$Similaridade(C, A) = \frac{\sum_{i=1}^{n} f(Ci, Ai) * W_i}{\sum_{i=1}^{n} W_i}$$

em que,
C = caso consultado; A = caso analisado; n = número de atributos no caso;
w = peso atribuído ao i-ésimo atributo; i = i-ésimo atributo;
f = função que calcula a similaridade local entre os casos C e A para o i-ésimo atributo.

As funções de similaridade local dependem da característica do atributo. Foram utilizadas 2 (duas) funções de similaridade local: a *MaxString()* e a *Equal*. A função *MaxString()* retorna um valor entre 0 e 1 indicando a semelhança da maior *substring* que pertence a ambas as strings passadas como parâmetro.

$$MaxString(x, y) = \frac{f(x, y)}{g(x, y)}$$

em que,
x, y = *strings* passada como parâmetro;
f(x,y) = função que calcula o tamanho da maior *substring* comum às *strings* x e y;
g(x,y) = função que retorna o tamanho da maior *string*.

A função *Equal()* é mais simples e retorna o valor 1 se os atributos possuem o mesmo valor. Caso contrário, retorna 0. Na Tabela 3 é apresentada a função de similaridade local e o peso atribuído a cada atributo representado no caso. A fim de efetuar

Tabela 3. Similaridade local para os atributos do caso.

Atributo	Similaridade Local	Peso
Categoria do problema	*Equal()*	1.0
Complexidade	*Equal()*	1.0
Enunciado do problema	*MaxString()*	1.0
Assuntos tratados	*Equal()*	5.0
Padrões de programação sugeridos pelo professor	*Equal()*	1.0

o ajuste ideal dos pesos para cada atributo representado no caso, foi realizado um experimento, no qual foram testadas combinações de pesos conforme descrito na seção a seguir.

4. Metodologia do Experimento

Para a realização do experimento, a base de dados com 200 problemas de programação foi dividida aleatoriamente em 2 (dois) conjuntos: o conjunto de consulta, contendo 25% dos problemas e; a memória de casos do RBC, constituída pelos demais 75% dos problemas.

Nesta divisão, o conjunto de consulta representa os casos que serão utilizados como entradas para consulta ao RBC. A memória de casos representa os casos que serão consultados pelo RBC.

Os pesos dos atributos do enunciado do problema, assuntos tratados e padrões de programação sugeridos pelo professor variam de 1 a 5, sendo o valor 5 considerado o maior peso possível a ser aplicado a um atributo.

Para os atributos categoria e complexidade, convencionou-se o peso 1. Esta convenção ocorreu por motivos pedagógicos, visto que para o aluno é difícil identificar se dois problemas são similares levando em consideração apenas a categoria e a complexidade do problema. Ou seja, se dois problemas não são similares a partir dos demais atributos, a categoria e a complexidade não devem influenciar fortemente no resultado.

No experimento, foram realizadas 50 consultas aos RBC, para obter os 5 problemas mais similares ao consultado, para cada configuração de pesos. Para cada conjunto de consultas realizadas, com uma determinada configuração de pesos, foram calculados a média, o desvio padrão e o coeficiente de variação (C.V.)[4]. O algoritmo de recuperação utilizado pelo RBC foi o k-NN e as funções de similaridade local utilizadas para cada atributo estão definidas na Tabela 3.

A partir da análise dos resultados iniciais obtidos, foi possível ajustar os pesos conforme apresentado na Tabela 3. Na seção a seguir, serão descritos e analisados os resultados obtidos.

5. Apresentação e Análise dos Resultados

Na Tabela 4 são apresentados os 5 (cinco) melhores resultados obtidos com o ajuste de pesos para os atributos analisados.

Tabela 4. Melhores resultados obtidos com o ajuste de pesos.

Enunciado	Assuntos	Padrões	Média	Desvio Padrão	C.V. (%)
1	5	1	0.66	0.18	26.68
1	5	2	0.62	0.18	28.48
2	4	1	0.6	0.17	28.38
2	5	1	0.63	0.18	27.87
2	5	2	0.59	0.17	29.30

Como pode ser observado na Tabela 4, os melhores resultados apresentam um peso superior, em relação aos demais, para o atributo assunto tratado. Isto comprova a importância deste atributo na identificação da similaridade entre os problemas. Além disso, tal fato pode ser justificado pedagogicamente, uma vez que problemas que tratam do mesmo assunto geralmente possuem um grande número de características em comum, o que afeta diretamente a similaridade entre os problemas.

Outro aspecto relevante, é que os pesos obtidos para os atributos enunciado do problema e padrões de programação variam entre 1 e 2, o que caracteriza a relação restrita entre estes atributos. Essa observação é importante, no sentido em que o conjunto de padrões sugeridos pelo professor representam os principais elementos necessários para construção da solução do problema.

Para exemplificar os resultados obtidos na recuperação dos problemas de programação por meio do raciocínio baseado em casos, foi realizada uma consulta à base de casos. A Tabela 5 apresenta a descrição do problema consultado. A partir dessa consulta, foram selecionados os 2 (dois) problemas mais similares ao problema consultado. A Tabela 6 apresenta os resultados obtidos nessa consulta. Vale ressaltar que o peso dos atributos para esta consulta está descrito na Tabela 3.

[4]Medida de dispersão que compara diferentes distribuições. Essa função é calculada por meio da divisão do desvio padrão pela média [Pimentel-Gomes 1987].

Tabela 5. Descrição do problema consultado no RBC.

Enunciado	Comp.	Categoria	Assuntos	Padrões
Escreva um programa em PASCAL para ler um número inteiro N e imprimir os N primeiros números primos.	Fácil	Matemático	Operadores, variáveis, comandos de entrada e saída, comandos de seleção e comandos de repetição.	Primo e Repetição contada

Tabela 6. Resultados obtidos na consulta ao RBC.

Enunciado	Comp.	Categoria	Assuntos	Padrões	Sim.
Escreva um programa em PASCAL para ler um número inteiro qualquer e determinar todos os seus divisores exatos.	Fácil	Matemático	Operadores, variáveis, comandos de entrada e saída, comandos de seleção e comandos de repetição.	Repetição contada	0.724
Escreva um programa em PASCAL para determinar um número inteiro N tal que N + 3N + 5 seja divisível por 121.	Fácil	Matemático	Operadores, variáveis, comandos de entrada e saída, comandos de seleção e comandos de repetição.	Repetição por sentinela	0.702

Estes resultados comprovam que os dois problemas recuperados apresentam características semelhantes para resolução do problema, em relação ao consultado, uma vez que os recuperados tratam de problemas que necessitam realizar múltiplas divisões para sua solução, assim como ocorre na identificação de um número primo.

6. Considerações Finais

A principal contribuição do trabalho apresentado consite em uma alternativa para representação e recuperação de casos no domínio de programação. Para isso, foram utilizados elementos que fornecem indícios de similaridades entre os problemas como, por exemplo, seu enunciado e os assuntos tratados pelo problema.

Além disso, a utilização de padrões de programação para alunos iniciantes como um dos atributos na representação e recuperação do caso possibilita ao aluno identificar a importância destes no emprego do raciocínio por analogia para resolução dos problemas de programação.

Os estudos preliminares realizados, a fim de ajustar os pesos para cada um dos atributos, apontam para fortes indícios que indicam que a alternativa de representação e recuperação dos casos está adequada ao domínio. Entretanto, sugere-se uma avaliação qualitativa dos problemas recuperados Esta avaliação será realizada com uma amostra de alunos de um instituição de ensino superior que estejam cursando as disciplinas introdutórias de programação.

Com a conclusão do trabalho, dar-se-á início ao desenvolvimento das demais etapas do sistema de raciocínio baseado em casos (reuso, revisão e retenção) e, assim que estas forem concluídas, irão compor um ambiente virtual para auxílio ao ensino de programação, cujo objetivo é aprimorar a habilidade do aluno na resolução de problemas de programação por meio do raciocínio por analogia.

Vale ressaltar, que este sistema de raciocínio baseado em casos tem como principal público-alvo os alunos iniciantes das disciplinas de programação. Tal fato, porém, não descarta sua utilização por programadores experientes visto que poderá auxiliá-los a relembrar problemas similares que já resolveram anteriormente.

Referências

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59.

Bello-Tomás, J. J., González-Calero, P. A., and Díaz-Agudo, B. (2004). JColibri: An Object-Oriented Framework for Building CBR Systems. In *ECCBR*, pages 32–46.

Díaz-Agudo, B., González-Calero, P. A., Recio-García, J. A., and Sánchez, A. (2007). Building cbr systems with jcolibri. *Special Issue on Experimental Software and Toolkits of the Journal Science of Computer Programming*, 69(1-3):68–75. Impact Factor 0.734 Journal Citation Reports® 2005, published by Thomson Scientific.

de Barros, L. N., Ana, Delgado, K. V., and Matsumoto, P. M. (2005). A tool for programming learning with pedagogical patterns. In *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 125–129, New York, NY, USA. ACM.

Delgado, K. V. (2005). Diagnóstico baseado em modelos num sistema tutor inteligente para programação com padrões pedagógicos. Master's thesis, Universidade de São Paulo. Instituto de Matemática e Estatística.

Gaia (2002). jCOLIBRI CBR Framework. http://gaia.fdi.ucm.es/projects/jcolibri/.

Koslosky, M. A. N. (1999). Aprendizagem baseada em casos - um ambiente para ensino de lógica de programação. Master's thesis, Universidade Federal de Santa Catarina.

Mendonça, A. P. (2008). Entendimento de problemas: Uma investigação exploratória com alunos iniciantes de programação. Technical report, Departamento de Sistemas e Computação. Coordenação de Pós-Graduação em Ciência da Computação.

Muller, O. (2005). Pattern oriented instruction and the enhancement of analogical reasoning. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 57–67, New York, NY, USA. ACM.

Muller, O., Ginat, D., and Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. *SIGCSE Bull.*, 39(3):151–155.

Muller, O., Haberman, B., and Averbuch, H. (2004). (an almost) pedagogical pattern for pattern-based problem-solving instruction. *SIGCSE Bull.*, 36(3):102–106.

Pimentel-Gomes, F. (1987). *Curso de estatística experimental*. 12 edition.

Porter, R. and Calder, P. (2003). A pattern-based problem-solving process for novice programmers. In *ACE '03: Proceedings of the fifth Australasian conference on Computing education*, pages 231–238, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Postgresql (2003). PostgreSQL. http://www.postgresql.org.

Pressman, R. S. and Pressman, R. (2004). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science/Engineering/Math.

Proulx, V. K. (2000). Programming patterns and design patterns in the introductory computer science course. In *SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 80–84, New York, NY, USA. ACM.

Sommerville, I. (2004). *Software Engineering*. Addison Wesley, seventh edition.

Wangenheim, A. V. O. N. (2003). *Raciocínio Baseado em Casos*. 1 edition.