# MEDIDAS COGNITIVAS NO ENSINO DE PROGRAMAÇÃO DE COMPUTADORES COM SISTEMAS TUTORES INTELIGENTES

Andrey Ricardo Pimentel e Alexandre Ibrahim Direne

Depto. de Informática - UFPR

andrey@pos.inf.ufpr.br, alexd@inf.ufpr.br

**RESUMO**

Este artigo apresenta a utilização de medidas cognitivas para a descrição e ordenação da base de programas exemplo de Sistemas Tutores Inteligentes destinados ao ensino de programação de computadores. O enfoque abrange o processo de autoria por meio de ferramentas de software que auxiliam a criação de seqüências adequadas de ensino as quais tendem a causar uma redução no tempo de aprendizado do aluno e possibilitar a utilização de diversas estratégias pedagógicas automatizadas.

**ABSTRACT**

This article reports on the application of cognitive measures to describe and order the database of example programs of Intelligent Tutoring Sistems aimed at the teaching of computer programming. The methodology adopted embraces the authoring process by means of software tools that aid the creation of appropriate teaching sequences which tend to cause a reduction in a student's learning time and to allow the use of various pedagogic strategies.

## 1 INTRODUÇÃO

O objetivo deste trabalho é apresentar a definição e a utilização de medidas cognitivas para ordenar um conjunto de programas exemplo de um Sistema Tutor Inteligente (STI) e na implementação de ferramentas de autoria baseadas nestas medidas. A classe de STI compreendida no escopo desta pesquisa é dirigida ao ensino de programação de computadores. O objetivo é possibilitar a automação de uma boa escolha do próximo programa a ser trabalhado com o aprendiz, simulando a experiência do professor, a fim de minimizar o tempo de aprendizado e possibilitar a utilização de diversas estratégias pedagógicas.

Em uma sessão de ensino de programação de computadores, tradicional ou por computador, após terminar a discussão em cima de um exemplo apresentado, o professor simplesmente escolhe outro exemplo e continua a sessão. O grande problema é como selecionar adequadamente o próximo exemplo. Em uma turma real, o professor usa a sua experiência. Nos sistemas tutores, esta escolha ainda não está formalizada,

pois os próprios professores não sabem dizer exatamente quais os critérios usados para tal.

Medidas Cognitivas para a escolha adequada de um próximo exemplo foram apresentadas em (Pimentel e Direne, 1997) para o ensino de conceitos visuais aplicados à radiologia médica. Tanto o ensino de conceitos visuais quanto o ensino de programação de computadores são baseados na aquisição de perícia e na apresentação de casos para os alunos solucionarem. Porém, em radiologia médica a solução é um diagnóstico expresso em um laudo, enquanto que na programação a solução é um algoritmo expresso em um programa de computador.

## 2 STI PARA O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Do ponto de vista de fatores cognitivos, a programação de computadores é uma tarefa de alta carga para iniciantes por duas razões principais: a falta do conhecimento de princípios de programação e a falta de perícia. Comumente, sistemas tutores inteligentes e micromundos destinados a iniciantes de programação tentam lidar com estas duas carências por meio da combinação de (1) técnicas de Inteligência Artificial, (2) ambientação construtivista por exploração livre e (3) recursos de visualização científica para, no final, criar interfaces de razoável versatilidade para o aluno. Alguns exemplos destes sistemas são o Lisp Tutor (Anderson Et. Al., 1985), GIL (Reiser, 1987), LAURA (Adam e Lawrent, 1980), BIP (Barr et. al., 1976), Proust (Johnson, 1986), Spade (Miller et. al., 1982), Talus (Murray, 1986) e Bridge (Bonar, 1988). Todavia, estes sistemas tutores e micromundos tendem a enfatizar apenas os aspectos semânticos da solução de um exemplo de programação, independentemente do grau de dificuldade do enunciado deste exemplo.

Esta ênfase foi fundamentada até o presente, em grande parte, por resultados empíricos que apontaram a integração de comandos individuais como a principal dificuldade demonstrada por iniciantes que utilizam linguagens de programação convencionais (Bonar e Soloway, 1985) (Chi et. al., 1982) (du Boulay, 1986). Porém, sistemas tutores também deveriam levar em consideração que a complexidade de um enunciado de programação tem muitas componentes cognitivas, tornando um grande desafio de pesquisa a ordenação formal de bases de exemplos por grau de dificuldade. Parte da importância deste procedimento pode ser constatada quando um problema de programação significativamente mais complexo do que o anterior é proposto a um aprendiz. Esta fato pode levar à ocorrência de erros múltiplos, impedindo o sucesso do

aluno por longos períodos de tempo.

Alguns dos sistemas tutores e micromundos de programação mais conhecidos, tais como BALSA (Brown, 1988), TINKER (Liberman, 1984) e TPM (Eisenstadt et. al., 1991), apresentam aos alunos suas bases de enunciados em uma ordem de complexidade extremamente rígida. Nestes casos, a complexidade de tais enunciados foi registrada implicitamente pelos projetistas e construtores de tais sistemas, tornando-os inflexíveis para a adaptação ao grau de competência de cada aluno. Uma contribuição singular (del Soldato e du Boulay, 1995), tenta enfocar a complexidade de enunciados como uma das principais componentes da motivação do aluno no aprendizado. Por exemplo, a repetição sistemática de enunciados completamente diferentes, porém, de graus de complexidade aproximadamente iguais pode elevar consideravelmente a auto-confiança do aprendiz, mantendo-o motivado e produtivo. Todavia, as principais características cognitivas que compõem a complexidade de um enunciado de programação de computadores nunca foram apresentadas explicitamente em nenhum trabalho de pesquisa.

Neste contexto, também é relevante observar que os sistemas tutores e micromundos existentes para programação de computadores são todos ferramentas de ensino específicas para um domínio de conhecimento apenas. Sendo assim, o re-aproveitamento de conceitos de complexidade de enunciados de programação passa necessariamente pela re-implementação completa de um sistema tutor, por exemplo, toda vez que houver a necessidade de se ensinar uma nova linguagem. Se a aquisição e o cômputo de fatores de complexidade de enunciados for efetuado por ferramentas independentes de domínio, isto pode facilitar em muito a construção de software educativo para áreas tão delicadas com as de ensino de linguagens de programação. Avanços recentes na generalização de estratégias de ensino, sistemas de autoria e de shells tutores (Major, 1995) (Woolf, 1996) (Murray e Woolf, 1992) contém informações valiosas sobre arquiteturas internas que podem beneficiar muito a construção de sistemas de ensino de programação, especialmente quando o projeto de tais sistemas inclui aspectos de aprendizagem à distância (Owusu-Sekyere e Branch, 1996).

## 3 MEDIDAS COGNITIVAS E O ENSINO DE PROGRAMAÇÃO DE COMPUTADORES

Descritas em (Pimentel e Direne, 1997) para o ensino de conceito visuais em

radiologia médica, as medidas cognitivas são uma forma de conseguirmos obter algum meta-conhecimento sobre os exemplos de uma base de conhecimento de um STI. Elas também são os principais elos entre o modelo do domínio de um STI e seu modelo de estudante.

Para se calcular a complexidade de um programa de computador a engenharia de software fornece algumas medidas como linhas de código (LOC) ou pontos por função (FP) (Pressman, 1992). Com estas medidas consegue-se estimar o "tamanho" de um programa de computador, o esforço necessário para a sua confecção e em parte até mesmo a complexidade algorítmica de um programa. Mas elas se aplicam a programadores peritos e, portanto, não avaliam o quanto um determinado programa exige de um aprendiz em termos de conhecimentos adquiridos e capacidades desenvolvidas ao longo do aprendizado.

Para se conseguir uma medida que quantifique cognitivamente um exemplo ou, no caso, um enunciado de programação a ser mostrado para um aprendiz, precisamos analisar como este exemplo irá contribuir para a evolução do aprendizado. A carga cognitiva de um exemplo, pode ser definida como a capacidade que um exemplo tem em exercitar o aprendiz na construção de um programa de computador. A carga cognitiva do exemplo pode ser dividida em algumas componentes. Cada uma destas componentes irá medir um tipo de contribuição ao aprendizado.

Uma destas componentes é a complexidade de software que pode ser avaliada através das medidas convencionais como LOC ou FP. No caso do ensino de programação para iniciantes os programas são, na grande maioria dos casos, pequenos, e a sua complexidade pode ser melhor determinada usando LOC em conjunto com sua complexidade estrutural (CE) e seus detalhes de implementação (DI). A complexidade estrutural avalia o número de estruturas de decisão e repetição, e o grau de dependência entre as estruturas (aninhamentos). Os detalhes de implementação avaliam a quantidade de condições especiais que dificultam a implementação do programa.

Cada programa apresentado ao aluno contribui também para a aquisição das características do estereótipo de um programador perito. Cada programa tem peculiaridades que contribuem mais para a aquisição de determinadas características do estereótipo que outras. Além de CE e DI, as outras componentes são baseadas nas referidas peculiaridades e suas medidas são definidas a partir do estereótipo de um programador perito. Elas são usadas justamente para quantificar a exigência deste programa em cada uma das características deste estereótipo.

De maneira análoga à descrição das características do estereótipo de um radiologista perito feita por Cury (1997), um conjunto de características do estereótipo de um perito em programação de computadores foi identificado em projetos anteriores (Direne et. al., 1996) (**ver** tabela 1). As mais significativas são: (a) precisão sintática/semântica, (b) análise do problema, (c) reutilização de soluções e (d) simulação mental dos estados do computador durante a execução.

| |
|---|
| precisão sintática |
| precisão semântica |
| identificação de estruturas principais no programa fonte (busca por palavra-chave) |
| simulação mental dos estados do computador durante a execução |
| catálogo de erros |
| mapear mentalmente as estruturas do programa |
| checagem de pré-condições |
| análise do problema |
| integração dos subproblemas |
| generalização da solução |
| reutilização de soluções já conhecidas |
| catálogo de soluções |

Tabela 1 - Características Do Estereótipo De Um Programador Perito

A partir das características do *estereótipo* de um programador perito citadas no parágrafo anterior podemos medir a exigência do enunciado do programa para cada uma, conseguindo assim, além da complexidade, as outras componentes da carga cognitiva para o enunciado. A precisão sintática/semântica, a análise do problema, a reutilização de soluções e a simulação mental foram escolhidas porque elas representam diferentes dimensões ou "tarefas" (Rogalski e Samurçay, 1992) da construção de programas. A precisão sintática/semântica representa a componente da codificação propriamente dita ao passo que a análise do problema e a reutilização de soluções representam a componente da compreensão e projeto. A simulação mental representa a componente da manutenção.

A medida precisão sintática/semântica é definida como o quanto o exercício exige do aprendiz a capacidade de escrever de forma correta os comandos individuais de um programa e utilizá-los em contexto adequado; análise é definida como o quanto o exercício exige que o aprendiz tenha a capacidade de decompor o problema em

subpartes e projetar soluções integradas para cada uma. A medida reutilização de soluções é definida como o quanto este exercício exige do aprendiz a capacidade de adaptar soluções já conhecidas para resolver novos problemas, simulação mental mede o quanto o exercício exige do aprendiz a capacidade de simular mentalmente os valores das variáveis e arquivos envolvidos no programa durante todos os passos da execução deste.

A carga cognitiva de um enunciado de programa é conseguida através da composição das medidas cognitivas deste programa. Cada medida contribui com uma parcela da carga cognitiva, mas esta contribuição não é uniforme. Sendo assim, cada medida tem um peso diferente na composição da carga cognitiva, onde os pesos são obtidos através de informações dos especialistas no ensino de programação.

## 4 APLICANDO AS MEDIDAS COGNITIVAS NO ENSINO DE PROGRAMAÇÃO

As medidas cognitivas apresentadas na seção 3 constituem uma abordagem formal, do ponto de vista computacional, para a representação de metaconhecimento sobre programas exemplo de um STI. Esta seção apresenta uma ferramenta para permitir a utilização de medidas cognitivas por STI e algumas considerações sobre um exemplo prático.

### 4.1 Sequence: Uma Ferramenta de Autoria para Medidas Cognitivas

Sequence é uma ferramenta de autoria que utiliza medidas cognitivas. Esta ferramenta tem por objetivo permitir que o autor de um STI utilize a carga cognitiva de programas exemplo para a estruturação de uma seqüência de sessões de ensino que estarão disponíveis para os aprendizes. Ela foi inicialmente construída para complementar o sistema RUI (Direne, 1993), sendo na sua forma inicial adaptada para o ensino de radiologia médica (Pimentel e Direne, 1997). A ferramenta Sequence permite que um STI tenha um controle maior sobre a sessão de ensino. Sequence orienta o autor do STI para que este determine quantos e quais são os exemplos que serão mostrados ao usuário, tanto de maneira direta, indicando os próprios exemplos, como de maneira indireta, indicando parâmetros para que o próprio STI escolha estes exemplos.

A autoria usando a Sequence é realizada em dois níveis. No nível de cursos o autor irá determinar: quantos e quais serão os programas exemplo que irão compor a seqüência de exemplos, a sua ordem de apresentação e outros parâmetros que controlem

a apresentação dos programas exemplo durante o curso. No nível descrição cognitiva dos programas é feito o armazenamento do conhecimento sobre a capacidade cognitiva de cada exemplar, atribuindo valores que irão quantificá-lo cognitivamente.

A conexão entre os níveis de curso e de descrição cognitiva se dá através dos valores cognitivos. Estes valores são atribuídos a cada programa exemplo no nível de descrição cognitiva e servirão para que o autor do curso saiba a carga cognitiva de cada programa e assim possa escolher e/ou parametrizar a escolha automática deles.

Além de sugerir uma ordem inicial dos exemplos com base na carga cognitiva, a Sequence é capaz de adaptar a seqüência dos exemplos ao estado atual do aprendiz. Ela o faz escolhendo, com base em informações sobre o desempenho do aprendiz, recebidas do modelo do aprendiz do STI, o exemplo mais adequado a ser apresentado.

### 4.2 Um exemplo da Aplicação das Medidas Cognitivas no Ensino de Programação

No exemplo, tem-se dois exercícios de programação de computadores apresentados aos alunos em uma prova, cujos enunciados foram extraídos de (Farrer et. al. 1995) e estão apresentados na Figura 1.

Comparando os dois exercícios do ponto de vista da complexidade do programa resultado tem-se que o resultado do exercício 2 tem mais linhas de código e uma complexidade estrutural um pouco maior que o exercício 1, e ambos possuem o mesmo grau de detalhes de implementação. Por este ponto de vista, poder-se-ia dizer que o exercício 2 exige mais dos alunos que o exercício 1. Apesar disto, os alunos de duas turmas foram unânimes em afirmar, e também a correção das provas mostrou que o exercício 1 é mais difícil que o exercício 2.

1)Faça um programa que calcule e escreva o valor do co-seno do ângulo A, dado em radianos, com precisão de 0,000001, usando a série:

$$\cos(A) = 1 - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} + \frac{A^8}{8!} - \frac{A^{10}}{10!} + \ldots$$

Para obter a precisão desejada, adicionar apenas os termos cujo valor absoluto seja maior ou igual a 0,000001.

2)Numa certa loja de eletrodomésticos, cada vendedor de televisores recebe mensalmente, um salário mais uma comissão referente aos aparelhos vendidos por ele, por mês, obedecendo o quadro abaixo:

| TIPO | NO. DE TELEVISORES VENDIDOS | COMISSÕES |
|---|---|---|
| a cores | maior ou igual a 10<br>menor do que 10 | 14% do preço por TV<br>13% do preço por TV |
| preto e branco | maior ou igual a 20<br>menor do que 20 | 13% do preço por TV<br>12% do preço por TV |

Fazer um programa que leia o número de matrícula do vendedor, o número de televisores p&b e o número de coloridos vendidos e calcule e escreva o valor devido, relativo às comissões para os 30 vendedores da loja. Sabe-se que o preço de um tv colorido é $372,99 e o de um p&b é $184,99.

Figura 1 - Trecho De Uma Prova De Programação

A explicação deste fato se deve às medidas cognitivas. Apesar do exercício 1 exigir uma solução menor em tamanho, ele tem valores significativamente maiores para as medidas simulação mental e reutilização de soluções que o exercício 2. O exercício 1 tem um valor um pouco maior para a medida análise que o exercício 2 e ambos têm valores equivalentes para a medida precisão sintática/semântica.

## 5 CONCLUSÕES

As medidas cognitivas podem ser utilizadas no ensino de outras classes de domínio. Isto foi ilustrado com a comparação da carga cognitiva de dois exercícios de programação, onde o exemplo que foi considerado mais fácil por um professor, o qual não considerou a sua carga cognitiva, foi considerado mais difícil pelos aprendizes, o que ilustra a utilização de medidas cognitivas em outro domínio do ensino de perícia.

Neste trabalho encontramos e descrevemos medidas de relevância cognitiva para ordenar uma dada base de exemplos, de modo a permitir que o modelo tutorial de um STI modifique facilmente a ordem de apresentação dos programas exemplo. As medidas encontradas quantificam o potencial que programa exemplo tem em exercitar o aprendiz em uma determinada capacidade que ele deve desenvolver para se tornar perito. Elas, também, medem e representam computacionalmente a carga cognitiva destes programas exemplo. Além disto estas medidas visam a individuação do ensino,

isto é, permitem a aplicação de estratégias pedagógicas que mais se adaptam ao aprendiz e permite mudanças para corrigir eventuais falhas do aluno e mantê-lo em um estado motivacional alto (del Soldato e du Boulay, 1995), assim tentando reduzir o tempo de aprendizado.

Como trabalho futuro, propomos a definição e implementação de uma interface de comunicação entre os modelos tutorial e do estudante de um STI. A interseção entre os modelos do estudante e o tutorial é muito grande num STI. Um modelo tutorial precisa das informações que o modelo do estudante obtém sobre o desempenho e, até mesmo, sobre o estado motivacional do aprendiz. Portanto, a definição desta comunicação vai ajudar na construção de STI, facilitada provavelmente pela utilização de arquiteturas baseadas em multi-agentes (Oliveira e Viccari, 1996).

## REFERÊNCIAS


Adam, A. e Lawrent, J.. LAURA: A System to Debug Student Programs, *Journal of Artificial Intelligence* 15 (1980) 75-122.

Anderson, J. et al.. *Intelligent Tutoring Systems*, Science, 228 (1985) 456-462.

Barr, A. et al.. The Computer as a Tutorial Laboratory, *International Journal of Man-Machine Studies* 8 (1976) 567-596.

Bonar, J.. Intelligent Tutoring with Intermediate Representations. *Proceedings of the Conference on Intelligent Tutoring Systems (ITS'88),* 1988.

Bonar, J. e Soloway, E.. Pre-programming Knowledge: A Major Source of Misconceptions in Novice Programmers, *Human-Computer Studies in Mathematics* 20 (1985) 293-316.

Brown, M.. Exploring Algorithms Using Balsa-II, *IEEE Computer* 21 (1988) 14-36.

Chi, M. et al.. Expertise in Problem Solving. In: R. Strenberg (ed.), *Advances in the Psychology of Human Intelligence*. Lawrence Erlbaum, 1982.

Cury, D.. *FLAMA: Ferramentas e Linguagem de autoria para a Modelagem da Aprendizagem.* Tese de Doutorado, Divisão de Pós-graduação do Instituto Tecnológico de Aeronáutica. São José dos Campos. 1996.

Direne, A.; Nascimento, E.; Preto, T.; Sunye, M.; Muller Jr., B.; Valerio, M.; Sistemas Tutoriais para Assistir o Treinamento da Operação de Centrais de Comutação. *Anais do VII Simpósio Brasileiro de Informática na Educação (SBIE' 96)*, Belo Horizonte, 1996.

Direne, A.. Methodology and Tools for Designing Concept Tutoring Systems. *Proceedings of World Conference on Artificial Intelligence in Education (AIED'93),* 1993.

Del Soldato, T. e Du Boulay, B.. Implementation of Motivational Tactics in Tutoring Systems. *Journal of Artificial Inteligence in Education*. 6 (4). pp 337-378. 1995.

Du Boulay, B.. Some Difficulties in Learning to Program*, Journal of Educational Computing Research* 2 (1986) 57-63.

Eisenstadt, M. et al.. *Novice Programming Environments*. Ablex Publishing, Brighton-UK, 1991.

Farrer, H.; Becker, C.; Faria; E.; Campos, F.; Matos, H.; Santos, M. e Maia, M.. *Pascal Estruturado*. Ed. Guanabara-Koogan, 2ª. ed., 1995.

Johnson, W.. Intention-based Diagnosis of Novice Programming Errors - *Research Notes in Artificial Intelligence*. Morgan Kaufmann, 1986.

Liberman, H.. Seeing What Your Programs are Doing*, International Journal of Man-Machine Studies* 21 (1984) 311-331.

Major, N.. Modelling Teaching Strategies, *Journal of Artificial Intelligence in Education* 6 (1995) 117-152.

Miller, L. et al.. *Intelligent Tutoring for Programming Tasks*. Technical report, Texas Instruments, 1982.

Murray, W.. *Automatic Program Debugging for Intelligent Tutoring Systems*. PhD thesis, Texas University-Austin, 1986.

Murray, T. e Woolf, B.. Results of Encoding Knowledge with Tutor Construction Tools. *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92),* 1992.

Oliveira, F. M. e Vicari, R. M.. Are Learning Systems Distributed or Social Systems. *European Conference on A. I. in Education*, Lisbon. 1996.

Owusu-Sekyere, C. e Branch, R.. Computer Mediated Communication as a Means to Enhance Interaction and Feedback for Distance Education*, International Journal of Educational Telecommunications* 2 (1996) 199-227.

Pimentel, A. e Direne, I.. Medidas Cognitivas para o Ensino de Conceitos Visuais com Sistemas Tutores Inteligentes*. Anais do VIII Simpósio Brasileiro de Informática na Educação (SBIE'97)*, São José dos Campos, 1997.

Pressman, R.. *Software Engineering: A Pratictioner's Approach*. Mc Graw-Hill Inc., 1992.

Reiser, B.. Causal Models in Programming. *Proceedings of World Conference on Artificial Intelligence in Education* (AIED'87), 1987.

Rogalski, J. e Samurçay, R.. Task Analysis and Cognitive as a Framework to Analyse Environments for Learning Programming. Em: du Boulay, Dettori e Lemut (eds.),*Cognitive Models and Intelligent Environments for Learning Programming.* Springer-Verlag, 1992.

Woolf, B.. Intelligent Multimedia Tutoring Systems, *Communications of the ACM* 39 (1996) 30-31.