Das (Muitas) Dúvidas e (Poucas) Certezas do Ensino de Algoritmos

Cristian Koliver[1] **, Ricardo Vargas Dorneles** [1] **, Marcos Eduardo Casa**[1]

[1]Departamento de Informática – Universidade de Caxias do Sul
Caixa Postal 1352 – 95001-970 Caxias do Sul, RS

ckoliver@ucs.br, rvdornel@ucs.br, mecasa@ucs.br

Abstract. *In this paper, we discuss several aspects and problems related to how to teach algorithms. We raise several issues, and share our experience in teaching for trying to provide some answers. In addition, we present some results of our experiments performed in order to improve the results of learning by using a computational tool.*

Resumo. *Neste artigo, nós discutimos uma série de aspectos e problemas relacionados a como ensinar algoritmos. Nós levantamos uma série de questões e compartilhamos nossa experiência para tentar fornecer algumas respostas. Além disso, nós apresentamos os resultados de nossos experimentos relativos ao uso de uma ferramenta computacional como apoio ao aprendizado.*

1. Introdução

Disciplinas voltadas para o desenvolvimento da habilidade de construir algoritmos são, tradicionalmente, problemáticas e costumam apresentar altos índices de desistência e reprovação. Uma das razões para tal é o *despreparo da maior parte dos estudantes* para enfrentar uma disciplina cujo objetivo principal é auxiliá-los a desenvolver a habilidade de projetar soluções para problemas genéricos de uma maneira lógica e em passos.

A disciplina Algoritmos[1] é, normalmente e independentemente do curso para o qual é oferecida, introdutória e situada no início da grade curricular dos cursos, para que habilidades desenvolvidas sejam utilizadas em disciplinas posteriores. Isso faz com que ela tenha um público de calouros, ainda carregando vícios oriundos de práticas mecanicistas passadas, usuais em disciplinas dos ensinos fundamental e médio (como Química, Física e Matemática) que acostumam o estudante a aplicar fórmulas matemáticas de forma direta, sem qualquer tipo de análise mais profunda de problemas que, em geral, representam situações muito distantes do mundo real. Por conseguinte, ele ingressa na universidade apto a resolver apenas determinados tipos de problemas, sem conhecer um método geral de resolução. Esse problema é mundial[2]. P. Henderson [5], por exemplo, menciona que os instrutores costumam ficar pasmos ao constatar que a maioria dos estudantes não consegue entender nem elaborar algoritmos para os mais simples problemas. Esse choque

[1]A partir desse momento, nesse texto será utilizado o nome "Algoritmos"para nos referenciarmos, de maneira genérica, às disciplinas voltadas para o desenvolvimento da habilidade de construir soluções estruturadas para problemas.

[2]Vide, por exemplo, [5], [4], [6], etc.

decorre do fato que o instrutor, muitas vezes, inicia a disciplina com a expectativa equivocada de que seu público já está munido *a priori* de habilidades para análise e resolução de problemas, obtenção de conclusões lógicas e capacidade de sintetização.

Outrossim, a ausência de disciplinas que envolvam Filosofia ou Álgebra Booleana nos currículos da maior parte das escolas implica no fato dos estudantes cursarem a disciplina Algoritmos sem seu pré-requisito fundamental: a Lógica. O aprendizado em Lógica é a alfabetização necessária para Algoritmos. A despeito de nós utilizarmos a Lógica a todo momento, em nosso dia-a-dia, grande parte dos estudantes não consegue aplicá-la de maneira satisfatória na construção de soluções algorítmicas sem os princípios teóricos básicos que a norteiam.

Outra forte razão para as dificuldades encontradas no ensino e na aprendizagem de Algoritmos é a *ausência de uma metodologia de ensino*. A despeito dos inúmeros esforços para definição de uma série de aspectos metodológicos, ainda hoje há mais dúvidas do que certezas. Conforme o relatório final [8] da força tarefa comum (CC2001) da *Computer Society of the Institute for Electrical and Electronic Engineers* (IEEE-CS) e a *Association for Computing Machinery* (ACM), durante toda a história da educação em Ciência da Computação, a estrutura do curso introdutório da informática tem sido assunto de intensos debates, com muitas estratégias tendo sido propostas, a maioria contando tanto com fortes defensores quanto detratores, adquirindo muito freqüentemente um caráter de guerra religiosa, que gera mais calor do que luz. Por ainda não ter sido encontrada nenhuma abordagem ideal e pelo fato de que todas as abordagens propostas têm pontos fracos e fortes, o mesmo relatório não indica qualquer uma específica, sugerindo, porém, que as instituições ou educadores individuais continuem os experimentos na área, já que a inovação pedagógica é necessária para um sucesso contínuo em um campo que muda muito rapidamente.

Finalmente, a *heterogeneidade do público* tem sido um fator com o qual os instrutores têm freqüentemente dificuldade em lidar. É inegável que alguns estudantes têm uma engenhosidade superior nata e, por conseguinte, uma facilidade natural para a elaboração de soluções algorítmicas para problemas. Isso acarreta problemas para os instrutores em termos da definição do programa da disciplina e do grau de complexidade dos problemas propostos. Também, para permitir que as turmas sejam compartilhadas por uma maior quantidade de estudantes, em muitas instituições uma mesma disciplina de Algoritmos é oferecida para estudantes de cursos de Ciência da Computação e Engenharias das mais diversas, com diferentes expectativas e anseios. Isso traz uma dificuldade adicional no uso de abordagens oriundas de investigações que têm como foco principal, na maior parte das vezes, o ensino dessa disciplina no contexto de cursos de graduação em Ciência da Computação ou correlatos, onde Algoritmos compõe uma linha mestra da grade curricular. Além disso, para a maioria das Engenharias, o desenvolvimento de programas e sistemas computacionais é uma atividade apenas periférica, o que muitas vezes é fator de desmotivação para os estudantes[3].

Os problemas mencionados acima já são por demais conhecidos e nesse trabalho não temos a pretensão de apresentar soluções definitivas para quaisquer deles. Nossa

[3]Não excluímos aqui a hipótese da falta de motivação ser decorrência de nossa inabilidade em demonstrar para o estudante de Engenharia que a capacidade de projetar algoritmos transcende a simples tarefa de programação, abrangendo a elaboração de soluções para *qualquer* tipo de problema.

intenção é, a partir da experiência acumulada em anos de docência na disciplina Algoritmos, compartilhar e justificar nossa visão em relação a alguns aspectos metodológicos do ensino dessa disciplina sobre os quais ainda pairam mais dúvidas do que certezas. Por acreditarmos que o computador, de uma maneira genérica, constitui-se em uma ferramenta poderosa na educação, neste trabalho também apresentamos alguns resultados obtidos em turmas de Algoritmos nas quais essa ferramenta foi usada como suporte para o ensino.

Este trabalho é assim dividido: na Seção 2, discutimos alguns pontos relativos ao ensino de algoritmos sobre os quais ainda pairam dúvidas a respeito da metodologia mais adequada; na Seção 3, descrevemos os resultados preliminares obtidos através do ensino de algoritmos em laboratório, em termos de aproveitamento e motivação dos estudantes; na Seção 4, discutimos nossos resultados e apontamos propostas para trabalhos futuros.

2. Algumas Questões

Conforme dito na seção anterior, existe uma grande gama de abordagens para tratar uma série de aspectos metodológicos (e também pedagógicos) relacionados à disciplina Algoritmos, sendo muitas dessas abordagens conflitantes. A seguir, serão descritos alguns pontos relativos ao ensino de algoritmos sobre os quais existem ainda incertezas a respeito da abordagem mais adequada.

2.1. Fundamentação Matemática: Necessária ou Adequada?

É ponto pacífico que a Lógica Matemática constitui o alicerce da lógica de programação. Estabelecer essa associação, contudo, é significativamente difícil em um nível introdutório no qual os estudantes carecem tanto de maturidade matemática quanto de um entendimento básico da própria disciplina.

P. B. Henderson [4] acredita que a disciplina Algoritmos deva ser integrada com Matemática Discreta, pois se ensinarmos Algoritmos de uma maneira informal, mostrando apenas os rudimentos da Lógica Booleana, só podemos esperar que os estudantes desenvolvam soluções algorítmicas para problemas extremamente simples.

Deve ser ressalvado que a opinião acima não é unânime: para R. A. Baeza-Yates [1], por exemplo, no início a disciplina Algoritmos necessita apenas de técnicas básicas de programação. Mais tarde, quando a complexidade e análise tornam-se importantes, o curso passa a exigir Matemática Discreta e fundamentos teóricos da Ciência da Computação, como Teoria dos Autômatos e Linguagens Formais.

Nossa experiência têm mostrado que a apresentação dos *princípios básicos* da Lógica é *suficiente* para a resolução da maior parte dos problemas propostos para um disciplina de nível introdutório. Contudo, uma questão em aberto é *quanto* deve-se trabalhar nesses princípios, através da resolução e aplicação de expressões lógicas, antes que o estudante esteja, de fato, habilitado a utilizar a Lógica na elaboração de algoritmos. Com relação à Matemática Discreta, consideramos um pré-requisito *desejável*, mas não absolutamente necessário no nível introdutório no qual a disciplina está inserida.

2.2. Como Projetar um Algoritmos?

Existe praticamente um consenso quanto à necessidade de análise do problema e projeto da solução quando da elaboração de um algoritmo. T. McGill e S. Volet [6] sustentam essa afirmação através de estudos [9] que mostram que enquanto programadores novatos tendem a iniciar a codificação do algoritmo imediatamente, programadores com experiência despendem muito mais tempo projetando e planejando a solução do problema antes de iniciar a codificação. Outro argumento utilizado é que a construção de algoritmos de qualidade – que tem influência direta sobre a qualidade dos programas [6] – pressupõe uma capacidade de abstração e conhecimento dos princípios básicos do processo de projeto. Por fim, quando a análise e o projeto são aspectos enfatizados na disciplina Algoritmos, fica claro para os estudantes que ela é uma disciplina voltada para a *resolução de problemas* e não simplesmente para codificação de soluções [2].

Na prática, contudo, o que temos observado são propostas de metodologias de análise e projeto absolutamente *ad hoc* ou, na melhor das hipóteses, utilizando princípios de Engenharia de *Software* (ciclo de vida do *software*, abstração e ocultação de dados, documentação, etc.) de uma maneira rudimentar. Isso é bastante compreensível, devido à limitação temporal de um semestre e à provável inadequação de um estudo mais profundo das técnicas completas de análise e projeto em uma disciplina introdutória. A despeito da maioria dos autores enfatizar a importância da análise minuciosa do problema, geralmente o foco central da metodologia (ou ao menos o aspecto mais detalhado) é o projeto da solução. Para tal, existem duas escolas de pensamento principais [11]. Na primeira escola, costuma-se apresentar problemas clássicos com soluções algorítmicas associadas tendo graus variados de eficiência (complexidade). Na segunda escola, o curso concentra-se nas técnicas usadas de projeto de algoritmos, como divisão-e-conquista, programação dinâmica, algoritmo guloso, ramificação e poda, etc. A ênfase é na técnica de projeto *per si*, exemplificada através de problemas de uma variedade de áreas de aplicação. Observamos que em ambas escolas a metodologia de projeto guarda pouquíssima semelhança com a maioria das metodologias utilizadas para análise e projeto de sistemas.

Até recentemente, uma das formas mais utilizadas para representação do projeto de um algoritmo era o fluxograma. Contudo, essa forma é trabalhosa, exigindo muitos diagramas para representação de algoritmos simples e a a necessidade de refazer todos os diagramas quando da alteração de pequenos detalhes do projeto. Atualmente, a forma mais utilizada para representação do projeto é através do uso de uma linguagem textual estruturada (pseudo-código) [2] [11] [4]. Tais linguagens costumam ser muito semelhante a uma linguagem de programação real simplificada, com sintaxe e semântica rígidas, tipos básicos e estruturados, definição de funções, etc. diferindo apenas pelo fato de que não existe um compilador ou interpretador para executar os algoritmos representados através delas. É interessante notar que o uso de pseudo-código torna muito tênue a linha que separa os processos de projeto e codificação. De fato, na maioria das vezes, eles acabam confundindo-se.

2.3. Usar ou Não Laboratório?

Não existe um consenso a respeito da adequação do uso de ferramentas automatizadas para a visualização da dinâmica da execução do [projeto de um] algoritmo. Para autores como S. Pollard e J. Forbes [10], o suporte para o uso de recursos visuais oferecido pelo

computador permite a proposição de problemas mais complexos e torna os encontros mais divertidos. Essa também é nossa opinião. Nossa experiência pessoal no ensino de algoritmos tem mostrado que aulas dedicadas a projetos "estáticos" entediam os estudantes, uma vez que eles nunca enxergam a saída de seus projetos. Assim, acreditamos que o uso de ferramentas que permitam não só visualizar o resultado do algoritmo, mas também o fluxo de execução e a dinâmica de mudanças nos valores das variáveis (teste de mesa), não só torna os encontros mais interessantes como também torna mais claros certos conceitos básicos (variável, entrada e saída de dados, etc.).

Uma desvantagem apontada por alguns quanto ao uso de ferramentas para execução dos projetos é que muitos estudantes tendem a corrigir o projeto, quando o resultado da execução não é aquele esperado, sem uma análise mais profunda, partindo para a abordagem da tentativa-e-erro. Na nossa opinião, essa possibilidade é bastante concreta quando do uso de ambientes que não permitem a execução passo-a-passo com monitoramento de variáveis.

2.4. Codificação em Linguagem "Real" ou Pseudo-Código?

Tanto linguagens de programação quanto pseudo-códigos são, em essência, apenas uma das inúmeras formas existentes para representar um algoritmo. No passado, era comum o uso de linguagens de programação "didáticas" (particularmente, Pascal) para representação de algoritmos já quando da apresentação das primeiras noções (vide, por exemplo, [3]) em parte pela fato de elas possuírem ambientes de programação, obtendo-se nos encontros, assim, os benefícios enumerados na seção anterior.

Em contrapartida, acreditamos que existem diversas razões que tornam as linguagens de programação reais inadequadas para a codificação de soluções algorítmicas em cursos introdutórios. Dentre essas razões, destacamos: (1) linguagens de programação tradicionais são usualmente difíceis de programar, ler e entender em decorrência do grande número de detalhes sintáticos-semânticos necessários para implementar suas funcionalidades. Isso faz com que estudantes noviços tenham a tendência de se concentrar nesses detalhes (codificação) ao invés da concentrar-se no desenvolvimento da solução; (2) as linguagens de programação tradicionais, em virtude da riqueza de comandos, possibilitam a codificação de um número infinito (ou quase) de programas representando soluções corretas para um mesmo problema mas que, do ponto de vista algorítmico, representam a mesma solução; (3) linguagens de programação tradicionais muitas vezes não permitem uma total abstração de aspectos da máquina, quando da codificação da solução, em virtude da grande variedade de tipos de dados oferecidos, com diferentes representações internas que limitam o conjunto de valores representáveis; e (4) o uso de uma linguagem de programação real em um curso introdutório pode "amarrar" o estudante ao paradigma de programação da linguagem (procedural, lógico, funcional ou orientado a objeto).

2.5. Qual o melhor Paradigma de Representação?

Conforme mostramos, um ponto bastante comum nas propostas metodológicas da disciplina Algoritmos refere-se à representação da solução de problemas através de um pseudo-código ou mesmo de uma linguagem de programação real. Qualquer forma de representação pode seguir algum paradigma de programação (procedural, funcional, lógico ou orientado a objeto). Para R. A. Baeza-Yates [1] o paradigma para a disciplina

Algoritmos deve ser imperativo, preferencialmente orientado a objeto. Uma das justificativas para essa preferência deve-se ao fato de que linguagens imperativas (na ótica do autor) estão mais próximas da máquina real do que linguagens funcionais ou lógicas, sendo mais fácil modelar e entender o tempo e/ou espaço usado por um algoritmo.

Entretanto, essa posição entra em choque com o relatório da força tarefa CC2001 [8] no qual consta que o estudo de algoritmos deve fornecer o *insight* da natureza intrínseca do problema e tanto quanto possível, *técnicas de solução independentes de linguagem de programação, paradigma de programação, hardware de máquina ou qualquer outro aspecto de implementação*. Autores como P. B. Henderson [5] compartilham nossa opinião de que na disciplina Algoritmos a representação utilizada deve ser livre de paradigma. Nesse nível, a maioria dos problemas propostos utilizam estruturas muito simples para justificar o uso de, por exemplo, orientação a objeto. Além disso, a maioria dos estudantes carece de experiência e maturidade suficientes para o entendimento de recursividade, conceito bastante explorado nos paradigmas funcional e lógico. Para o final do curso, acreditamos que a introdução de funções, procedimentos ou métodos é válida para que, na elaboração da solução para problemas mais complexos, o estudante possa utilizar uma abordagem *top down*.

2.6. Discussão

Dados os inúmeros problemas relacionados ao ensino de Algoritmos descritos neste trabalho, nós acreditamos que o uso de ferramentas computacionais das mais diversas naturezas – tutores inteligentes, ferramentas de visualização gráfica, aplicativos para apresentações de *slides*, etc. – pode ser um suporte pedagógico de grande valia. Na próxima seção, mostraremos os resultados preliminares obtidos com o uso de uma ferramenta para execução e depuração de algoritmos.

3. Ensino de Algoritmos em laboratório: Alguns Resultados

Neste primeiro momento, a fim de avaliar a eficiência do uso do computador como suporte para o ensino de Algoritmos, nosso foco restringiu-se ao uso de uma ferramenta (ambiente) para execução de algoritmos chamada Visualg[4]. A Fig.1 mostra uma tela da ferramenta contendo o código de um algoritmo para imprimir um histograma com a freqüência de distribuição de conceitos de uma turma com um número desconhecido de alunos.

A escolha desta ferramenta em detrimento de várias outras similares disponíveis é que (1) a sintaxe e a semântica da linguagem usada pela ferramenta são extremamente simples (vide descrição a seguir), suportando construções e tipos básicos suficientes para representação de uma ampla gama de soluções algorítmicas; (2) o vocabulário da linguagem é em português, o que nossa experiência tem mostrado que faz alguma diferença para a maior parte dos alunos; e (3) a ferramenta permite que o estudante edite sua solução e visualize sua execução passo a passo, através de interpretação, com verificação do conteúdo das variáveis e apontamento de erros de sintaxe. Ferramentas gráficas (por exemplo, Lens [7]) foram descartadas por considerarmos sua utilidade apenas em disciplinas mais avançadas, onde os algoritmos exigem estruturas de dados mais complexas, como listas, pilhas, filas, etc.

[4]Criado por Cláudio Morgado de Souza e disponível em http://www.apoioinformatica.inf.br.

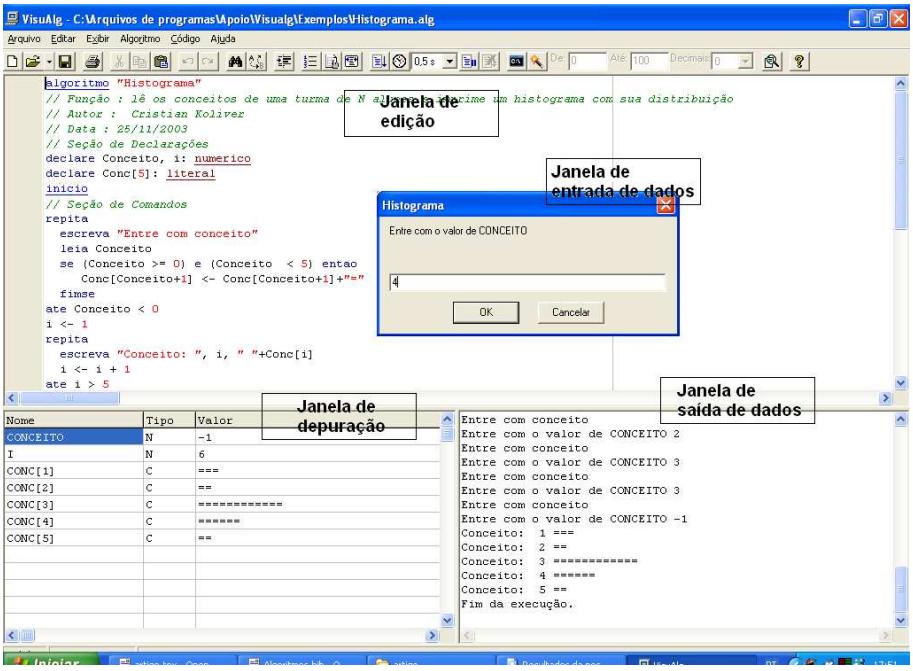

Figura 1: Lendo o valor de uma variável.

As principais razões para o uso de um ambiente desse tipo foram: (1) permitir que os estudantes verificassem a corretude das soluções propostas; (2) permitir que os estudantes verificassem a dinâmica da mudança dos valores das variáveis e do fluxo de controle; (3) acostumar os estudantes a realizar o teste de mesa, através de ferramentas automatizadas; (4) familiarizar os estudantes com o uso do computador e ambientes de programação; e (5) aumentar a motivação dos estudantes em relação à disciplina.

3.1. Metodologia

Foram utilizadas três turmas, referenciadas como A, B e C. A turma A consistia inicialmente de 39 estudantes, a maior parte vinculados ao curso de Engenharia de Produção, sendo as aulas dessa turma ministradas à noite; a turma B (vespertino) consistia inicialmente de 37 estudantes, a maior parte vinculados ao curso de Engenharia Mecânica; finalmente, a turma C (manhã) consistia inicialmente de 38 estudantes, do curso de Engenharia Ambiental. As aulas das turmas A e C foram ministradas em sala de aula; as aulas da turma B foram ministradas integralmente em laboratório, com uma máquina para cada dois alunos.

A despeito de apenas os estudantes da turma B utilizarem a ferramenta Visualg de forma sistemática, nós demonstramos seu uso e o utilizamos para exemplificar conceitos e mostrar o funcionamento de soluções algorítmicas, através de um canhão de projeção, nas três turmas. Também, durante todo o semestre, nós incentivamos que *todos* os estudantes (incluindo aqueles das turmas A e C) utilizassem o ambiente fora do horário de aula, bem como o correio eletrônico para demonstrar ou comunicar suas dúvidas em relação aos problemas propostos para resolução extra-classe ou pessoalmente, fora do horário de aula. Para esclarecimentos adicionais, constantemente incentivamos a procura pelos monitores (em cinco horários diferentes).

3.2. Linguagem e Ambiente

A linguagem utilizada para representar as soluções algorítmicas disponibiliza apenas três tipos de dados: numérico (números reais), literal (sem distinção para caracteres isolados ou seqüências de caracteres) e lógico (valores VERDADEIRO ou FALSO). Os únicos tipos estruturados disponíveis são arranjos de uma (vetores) e duas dimensões (matrizes).

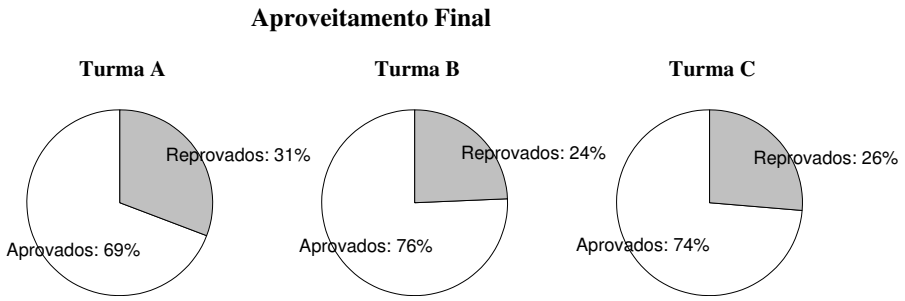

Figura 2: Índices de reprovação das três turmas.

Os operadores aritméticos são os quatro básicos (adição, subtração, multiplicação e divisão) mais os operadores de divisão inteira, resto da divisão e exponenciação. Os operadores lógicos são a adição lógica (ou), multiplicação lógica (e), negação (não) e adição lógica exclusiva (xou). Os operadores relacionais são os comuns (maior, menor, maior ou igual, menor ou igual, igual e diferente). Para literais, existe apenas a operação de concatenação de cadeias de caracteres, representada pelo sinal +.

Das construções de controle de fluxo disponíveis na linguagem, foram apresentadas apenas duas, em razão de elas serem suficientes para representação da solução da maioria dos problemas propostos [5]: se-entao-senão e repita-até.

3.3. Avaliação

Para avaliação do desempenho dos alunos, usou-se o método tradicional de aplicação de provas. O conteúdo das avaliações foi dividido em três áreas (Fundamentos, Algoritmos Seqüenciais e Condicionais, Algoritmos Iterativos e Algoritmos com Vetores), sendo aplicada uma prova escrita para cada área (todas de igual peso). Na turma B as provas foram também ministradas em sala de aula. As provas das três turmas foram semelhantes, no sentido de que exigiam que o estudante analisasse três problemas e propusesse soluções algorítmicas para eles, representadas através do pseudo-código usado. Uma quarta questão sempre envolvia o uso de teste de mesa, com o intuito de incentivar (de uma maneira mais "contundente") que o estudante passasse a utilizar, de fato, esse método para verificação de erros. Em geral, alguma solução possível para os problemas das provas podia ser obtida através da adaptação de soluções de problemas vistos em sala de aula. Foram aprovados os estudantes cuja média harmônica das três provas foi igual ou superior a seis (6.0). Aqueles que não alcançaram tal média puderam ainda prestar um exame, versando sobre todos os tópicos vistos. Neste caso, foi aprovado o estudante cuja nota foi igual ou superior a seis (6.0). Os conceitos, atribuídos conforme a média harmônica das três avaliações ou nota do exame, seguem a seguinte convenção: 0 ($nota < 6.0$), 1 ($6.0 \leq nota < 7.0$), 2 ($7.0 \leq nota < 8.0$), ($8.0 \leq nota < 9.0$), e 4 ($9.0 \leq nota$).

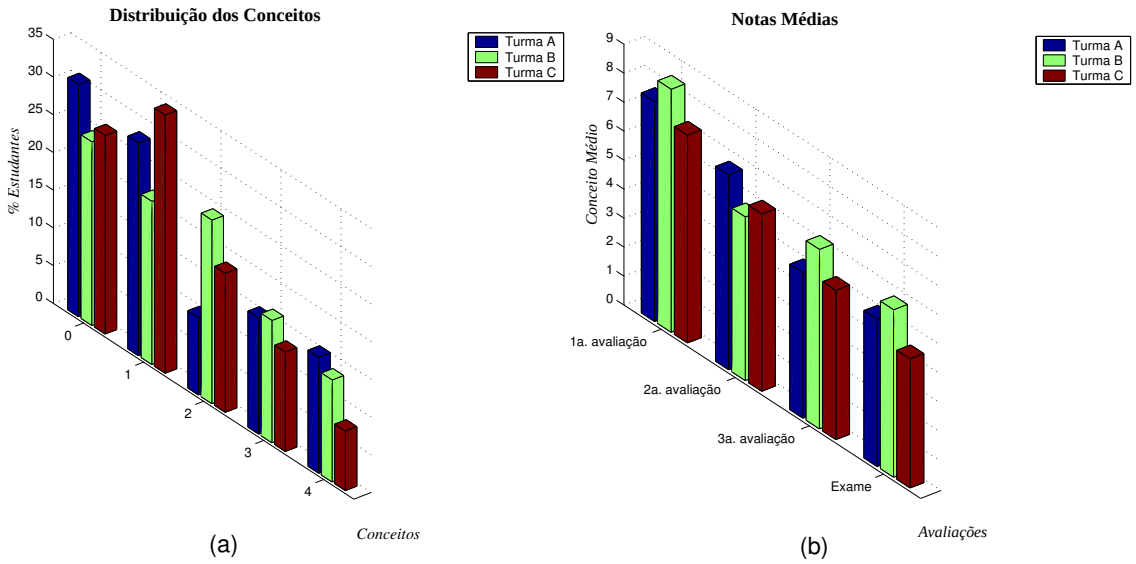

Figura 3: Desempenho das turmas: (a) distribuição dos conceitos finais; (b) notas médias nas quatro avaliações.

3.4. Resultados

A turma B, conforme o gráfico da Fig. 2 apresentou o menor índice de reprovação dentre as três. Além do índice de reprovação inferior aos das outras duas turmas, o conceito médio dos estudantes aprovados foi 2.2857, contra 2.2593 da turma A e 1.9286 da turma C. A distribuição dos conceitos finais das três turmas é mostrada no histograma da Fig. 4 (a). Também o desempenho da turma B foi superior ao das turmas A e C em três das quatro avaliações, conforme mostra o gráfico da Fig. 4 (b).

Nos histogramas da Fig. 4 podemos ver a distribuição do desempenho dos estudantes das três turmas, por intervalo de notas nas quatro avaliações. Enquanto na 1^a avaliação 79.48% dos estudantes da turma A e 69.44% dos estudantes da turma B obtiveram notas iguais ou superiores a 6.0, na turma B esse percentual alcançou 91.176%. Na 2^a avaliação, os percentuais das três turmas foram próximos: 61.11% para a turma A, 61.29% para a turma B e 65.71% para a turma C. Na 3^a avaliação, contudo, a turma B voltou a apresentar um desempenho muito superior ao das outras duas turmas: 64.51% dos estudantes dessa turma obtiveram notas iguais ou superiores a 6.0 contra 42.85% da turma A e 39.39% da turma C. A diferença aumentou mais ainda no exame: 66.66% contra 50% da turma A e 54.54% da turma C.

Com relação aos índices de desistência, todavia, a turma B foi a que apresentou o pior desempenho, conforme os gráficos da Fig. 5 (a). Porém, os índices de desistência dessa turma foram decrescentes ao longo do semestre, ao contrário das outras duas turmas nos quais eles oscilaram, como podemos ver no gráfico da Fig. 5 (b).

Como fator motivador, nossos resultados preliminares também indicam a eficiência do uso dessa ferramenta computacional ou similar. Na avaliação institucional dos cursos de graduação promovida por nossa universidade semestralmente, no quesito "Adequação dos recursos didáticos (textos, filmes, transparências, recursos computa-

cionais, etc.) às situações de ensino-aprendizagem", a nota média atribuída pelos estudantes da turma B que realizaram a avaliação foi 8.25, em um intervalo de 0 a 10, contra 6.00 da turma A e 5.60 da turma C. No quesito "Eficácia da metodologia de ensino da turma", a nota média atribuída pelos estudantes da turma B foi 8.00, contra 7.38 da turma A e 7.20 da turma C. Também em enquete *on-line* promovida para os estudantes da turma B, os resultados foram animadores, apesar de deverem ser observados com ceticismo devido à baixa participação dos estudantes: na 1^a questão, relativa à motivação do aprendizado, todos os seis estudantes que a responderam consideram que as aulas em laboratório ajudaram a aumentar muito a motivação para o aprendizado de algoritmos; com relação à eficiência do aprendizado, dos dez estudantes que responderam à questão, oito consideram que as aulas em laboratório auxiliaram muito na compreensão de algoritmos contra dois que consideram que as aulas em laboratório não auxiliaram nem prejudicaram; com relação às qualidades da ferramenta, dos seis estudantes que responderam à questão relacionada, cinco a consideram uma ferramenta boa ou razoável para visualização do funcionamento de um algoritmo. Dos seis estudantes que responderam à questão relacionada ao uso da ferramenta, cinco a utilizaram conforme o esperado, ou seja, através da execução passo-a-passo, acompanhando as mudanças nos valores das variáveis .

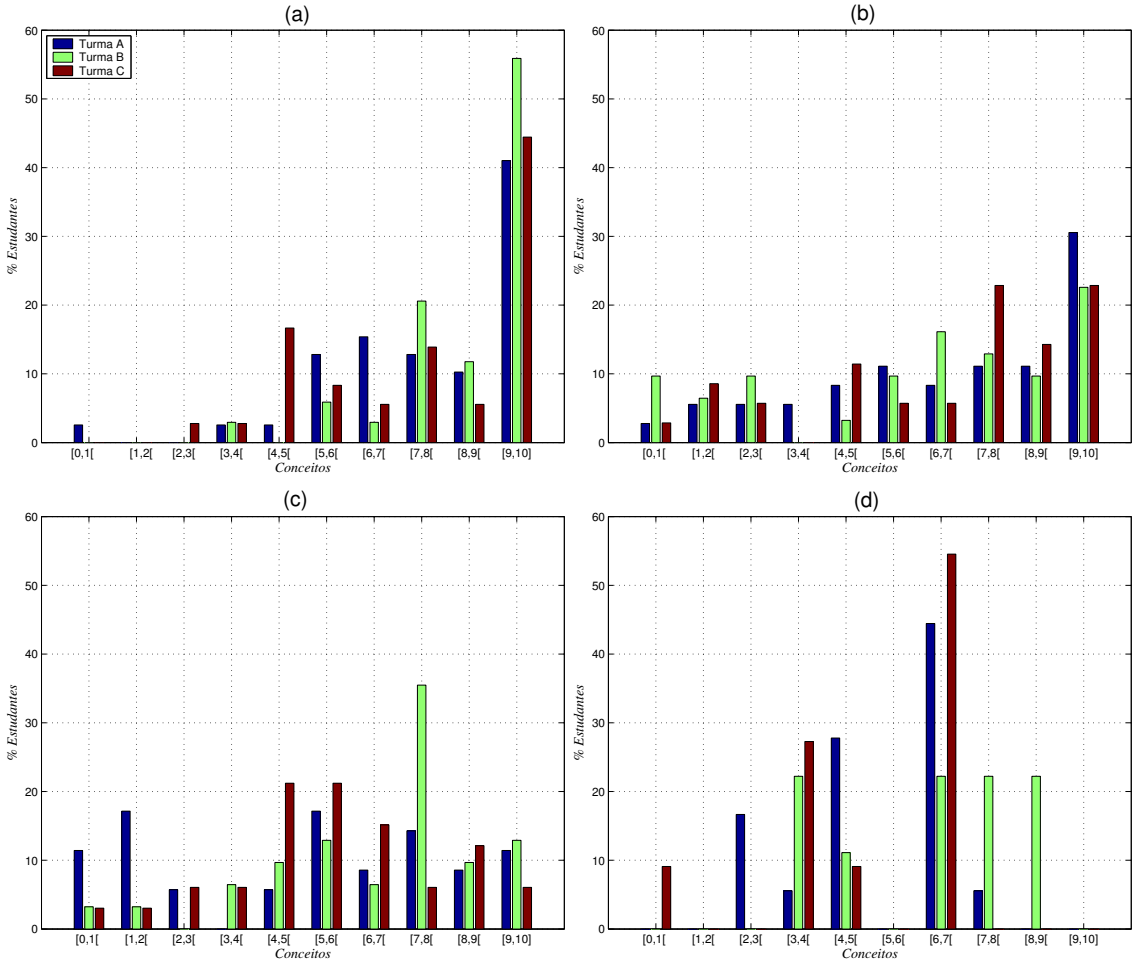

Figura 4: Histogramas de distribuição das notas: (a) 1^a avaliação; (b) 2^a avaliação; (c) 3 ^{a} avaliação; (d) exame.

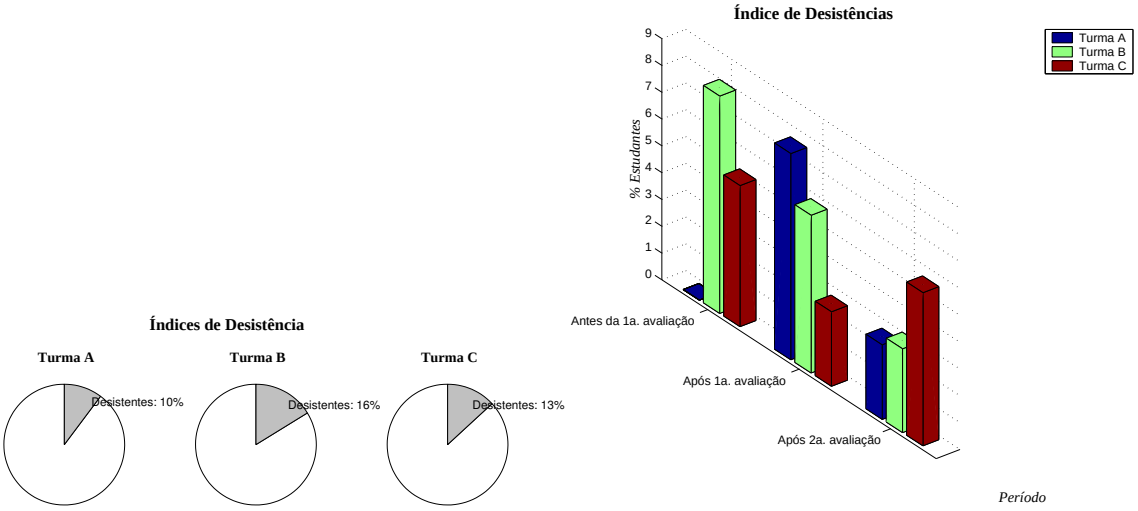

Figura 5: Distribuição das desistências: (a) percentuais de desistências; (b) desistências por período no semestre.

4. Conclusões

Neste trabalho, nós discutimos algumas das dificuldades relacionadas ao ensino da disciplina Algoritmos. Muitas dessas dificuldades decorrem da inexistência de estudos empíricos que demonstrem as vantagens ou desvantagens de determinadas abordagens metodológicas e pedagógicas na disciplina, dentre as quais: profundidade do pré-requisito matemático necessário, forma de representação e uso de laboratório. Para esses aspectos, emitimos a nossa opinião baseada apenas em uma experiência *ad hoc*, sem qualquer rigor científico.

Outra grande dificuldade com a qual os instrutores têm se defrontado nessa disciplina refere-se à dificuldade inerente ao próprio estudante em desenvolver as habilidades necessárias para a construção de soluções algorítmicas, independentemente de sua bagagem de conhecimentos. Para tentar superar essa dificuldade, temos dedicado a maior parte da carga horária da disciplina à demonstração de técnicas de projeto e resolução de problemas, esperando que, ao seu final, os estudantes estejam preparados para resolver problemas genéricos não cobertos nas aulas através da adaptação de soluções de problemas conhecidos e similares. Essa abordagem decorre de nossa crença de que da mesma forma que uma Escola de Belas Artes não pode formar um artista, mas apenas ensinar e desenvolver nos estudantes a habilidade de aplicar as técnicas existentes, um estudante que cursou Algoritmos não terá que, necessariamente, estar apto a construir soluções altamente criativas para problemas que ele porventura se defronte já que é sabido, por experiência, que um projetista ou programador típico dificilmente cria algo totalmente original.

É nossa crença, também, que o uso de recursos computacionais das mais diversas naturezas pode ajudar a contornar ou minimizar algumas das dificuldade relacionadas a essa disciplina. Em uma etapa inicial, verificamos a utilidade do uso de uma ferramenta computacional específica (no caso, um ambiente para execução passo-a-passo e depuração de algoritmos) em três turmas. Nossos resultados iniciais indicam que o uso

dessa ferramenta pode melhorar o desempenho dos estudantes, em termos de aproveitamento e índices de aprovação, além de ser um fator motivador. Tais resultados, contudo, ainda são preliminares e não podem ser generalizados, uma vez que mais experimentos são necessários para diluir a influência de fatores externos como horário e natureza dos estudantes das turmas.

Trabalhos futuros envolvem mais experimentos através do uso da ferramenta supra-mencionada em mais turmas bem como o uso de outros recursos computacionais, como fórum de discussão e enquetes *on-line*.

Referências

[1] BAEZA-YATES, R. A. Teaching algorithms. *SIGACT News 26*, 4 (1995), 51–59.

[2] GHAFARIAN, A. Teaching design effectively in the introductory programming courses. In *Proceedings of the fourteenth annual consortium on Small Colleges Southeastern conference* (2000), The Consortium for Computing in Small Colleges, pp. 201–208.

[3] GRIES, D. What should we teach in an introductory programming course? In *Proceedings of the fourth SIGCSE technical symposium on Computer science education* (1974), ACM Press, pp. 81–89.

[4] HENDERSON, P. Modern introductory computer science. In *Proceedings of the eighteenth SIGCSE technical symposium on Computer science education* (1987), ACM Press, pp. 183–190.

[5] HENDERSON, P. B. Anatomy of an introductory computer science course. In *Proceedings of the seventeenth SIGCSE technical symposium on Computer science education* (1986), ACM Press, pp. 257–264.

[6] MCGILL, T., AND VOLET, S. An investigation of the relationship between student algorithm quality and program quality. *SIGCSE Bull. 27*, 2 (1995), 44–48.

[7] MUKHERJEA, S., AND STASKO, J. T. Applying algorithm animation techniques for program tracing, debugging, and understanding. In *Proceedings of the 15th international conference on Software Engineering* (1993), IEEE Computer Society Press, pp. 456–465.

[8] ON COMPUTING CURRICULA IEEE COMPUTER SOCIETY ASSOCIATION FOR COMPUTING MACHINERY, T. J. T. F. Computing Curricula 2001 Computer Science. Tech. rep., IEEE-CS and ACM, December 2001.

[9] PETRE, M. Expert Programmers and Programming Languages. In *Psyclology of Programming*, J. M. Hoc, T. R. G. Green, R. Samurcay, and D. J. Gilmore, Eds. Academic Press, London, 1990.

[10] POLLARD, S., AND FORBES, J. Hands-on labs without computers. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (2003), ACM Press, pp. 296–300.

[11] SURAWEERA, F. Getting the most from an algorithms design course: a personal experience. *SIGCSE Bull. 33*, 4 (2001), 71–74.