Uma Análise do Sequenciamento Pedagógico no Ensino de Computação na Educação Básica

Nathalia da C. Alves[1], Christiane G. von Wangenheim[1], Jean C. R. Hauck[1]
Adriano F. Borgatto[1], Dalton F. de Andrade[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
88.049-200 – Florianópolis – SC – Brasil

nathalia.alves@posgrad.ufsc.br,
{c.wangenheim,jean.hauck,adriano.borgatto,dalton.andrade}@ufsc.br

Abstract. *Several curriculum frameworks have been proposed to guide computing education in K-12, including the SBC curriculum. However, observing significant differences regarding the sequencing of contents, a question that remains is to which regard the proposed sequence is appropriate for learning. In this context, this article conducts a large-scale analysis using Item Response Theory in order to classify contents by their complexity and compares these results with the SBC curriculum. The results of this research can be used by curriculum designers as well as instructors in order to improve computing education in K-12.*

Resumo. *Vários currículos de referência foram propostos de forma a guiar o ensino de computação na Educação Básica, incluindo as diretrizes da SBC. No entanto, observando diferenças significativas em relação ao sequenciamento de conteúdos, levanta-se a questão se a sequência proposta é apropriada para o aprendizado. Neste contexto, este artigo realiza uma análise em larga escala utilizando a Teoria da Resposta ao Item para analisar os conteúdos em relação à complexidade e compara estes resultados com as diretrizes da SBC. Os resultados desta pesquisa podem ser usados por pedagogos, bem como professores, de forma a melhorar o ensino da computação na Educação Básica.*

1. Introdução

No mundo inteiro propõe-se iniciar o ensino de computação na Educação Básica. O ensino de computação na Educação Básica inclui especificamente o pensamento computacional. O pensamento computacional refere-se aos processos de pensamento envolvidos na criação de soluções algorítmicas, ou passo-a-passo, que podem ser executadas por um computador [Wing 2006]. Recentemente, no Brasil, o MEC homologou a Base Nacional Comum Curricular (BNCC) abordando o desenvolvimento do pensamento computacional dentro da área de Matemática [MEC 2018].

> "A área de Matemática, no Ensino Fundamental, centra-se na compreensão de conceitos e procedimentos em seus diferentes campos e no **desenvolvimento do pensamento computacional**, visando à resolução e formulação de problemas em contextos diversos." [MEC 2018, p. 471].

Existem diversas maneiras de desenvolver o pensamento computacional, uma delas é por meio de atividades práticas de programação, como as atividades da *Hour of Code* (code.org) ou por meio do desenvolvimento de jogos, animações e aplicativos pelos alunos [Grover e Pea 2013; Lye e Koh 2014; Ortiz e Pereira 2018; Santos et al.

DOI: 10.5753/cbie.sbie.2019.1

2018]. Para esse fim, tipicamente se adotam linguagens de programação visual [Santos et al. 2018; Dagostini et al. 2018], como, por exemplo, App Inventor para desenvolver aplicativos móveis [Daniel et al. 2017].

Visando a inserção do ensino do pensamento computacional nas escolas, necessita-se de diretrizes de currículo para guiar o sequenciamento pedagógico definindo as etapas nas quais os conteúdos devem ser abordados. Portanto, surge a necessidade de currículos de referência que definam um conjunto orgânico e progressivo de aprendizagens essenciais que todos os alunos devem desenvolver ao longo da Educação Básica, descrevendo a progressão de aprendizagem dentro dessa área de conhecimento [Hlebowitsh 2010].

Nesse contexto, várias entidades têm realizado esforços para o desenvolvimento de diretrizes e currículos de ensino de computação na Educação Básica definindo o sequenciamento de conteúdos. Abordando especificamente o pensamento computacional, foi desenvolvido o *K-12 Computer Science Framework* nos Estados Unidos [CSTA 2016], o *Computing at School* no Reino Unido [CAS 2015], o *Australian Curriculum, Assessment and Reporting Authority* [ACARA 2015] e no Brasil, a Sociedade Brasileira da Computação (SBC) criou diretrizes para ensino de computação na Educação Básica [SBC 2018]. Além disso, o Centro de Inovação para a Educação Brasileira também desenvolveu o Currículo de Referência em Tecnologia e Computação [CIEB 2018]. As diretrizes de currículo da SBC abordam a aprendizagem de conceitos e práticas básicas de computação, incluindo o eixo de pensamento computacional. A partir das competências, essas diretrizes também definem objetivos de aprendizagem e o sequenciamento pedagógico para o ensino de computação para os diferentes níveis da Educação Básica. De forma geral, as diretrizes e currículos abordam conceitos básicos parecidos, como algoritmos e programação. No entanto, nota-se uma grande diferença no sequenciamento de conteúdo entre as diversas propostas.

O sequenciamento de conteúdo para atingimento de objetivos de aprendizagem é tipicamente feito com base em fundamentos pedagógicos que compreendem a carga cognitiva e a complexidade do conteúdo a ser abordado [Wasson 1990]. O conteúdo deve ser sequenciado de acordo com o nível de dificuldade, iniciando a aquisição do conhecimento com etapas simples e amplo suporte didático, em direção a etapas mais complexas com menor necessidade de suporte (*scaffolding*) [Silva et al. 2018]. A diminuição do suporte de forma gradual auxilia no desenvolvimento dos domínios afetivo, psicomotor e cognitivo do aprendiz [Ormrod 2018]. Assim, observando as diferenças de sequenciamento pedagógico entre as propostas de diretrizes e currículos, se questiona até que ponto a sequência proposta é adequada.

Para analisar a dificuldade de um conteúdo e sistematicamente definir o seu sequenciamento, técnicas da teoria da medida podem ser usadas. A teoria da medida propõe abordagens para analisar a dificuldade de conteúdo, tais como Teoria da Resposta ao Item (TRI) e Teoria Clássica dos Testes (TCT) via análise de itens. Para a aplicação de ambas as teorias é necessário criar itens relacionados ao conteúdo que se quer analisar.

A TCT propõe medidas nas quais os parâmetros do item são dependentes dos aprendizes e a medida depende do conjunto de itens como um todo. A TRI propõe modelos probabilísticos para calibração de parâmetros que, diferentemente da TCT, não são dependentes de um conjunto específico de itens. Para cada item podem ser

calibrados: o parâmetro *a* (inclinação ou discriminação), os parâmetros *b* (dificuldade) e o parâmetro *c* (acerto casual). Devido ao foco nas propriedades individuais de cada item, a TRI permite o posicionamento dos itens numa escala de proficiência [Andrade et al. 2000]. A escala de proficiência distingue o que é mais fácil ou difícil do ponto de vista dos aprendizes. Além disso, a TRI permite fazer uma comparação entre diferentes testes que abordam o mesmo conteúdo [Andrade et al. 2000]. Assim, a escala de proficiência pode ser utilizada como uma base sistemática para propor um sequenciamento do conteúdo em relação à dificuldade de aprendizagem.

Analisando-se as propostas de currículos e diretrizes, identifica-se uma falta de evidências científicas em relação ao sequenciamento de conteúdos. Algumas das propostas não apresentam informações sobre como foram desenvolvidas. Outras adotaram processos baseados no consenso da comunidade como o *K-12 Computer Science Framework* [CSTA, 2017]. Alguns trabalhos analisam de forma pontual etapas ou conceitos específicos da progressão do pensamento computacional na Educação Básica. Franklin et al. (2017) analisou o desempenho de alunos entre 9 e 12 anos para entender o sequenciamento conceitual em um currículo baseado em blocos. Rich et al. (2017) analisou a trajetória de aprendizagem especificamente dos conceitos de sequência, repetição e condicional do pensamento computacional. Seiter e Foreman (2013) propõem um *framework* para entender e avaliar o pensamento computacional nos primeiros anos da Educação Básica. Porém, não foram encontrados artigos que apresentem uma análise sistemática com base em dados referentes ao sequenciamento de conteúdo do pensamento computacional no contexto da Educação Básica brasileira. Assim, o objetivo deste estudo é analisar o sequenciamento de conteúdo das diretrizes de currículo propostas pela SBC para o eixo de ensino do pensamento computacional na Educação Básica, especificamente no Ensino Fundamental.

2. Diretrizes de currículo de ensino de computação na Educação Básica

No Brasil, a SBC criou diretrizes que incluem o pensamento computacional na Educação Básica [SBC 2018] (Figura 1). O eixo do pensamento computacional refere-se à capacidade de compreender, definir, modelar, comparar, solucionar, automatizar e analisar problemas de forma sistemática por meio da construção de algoritmos [SBC 2018]. O eixo de cultura digital refere-se ao letramento em tecnologias digitais para comunicação e expressão, e o eixo mundo digital refere-se à apropriação dos processos do mundo digital para compreensão e crítica de tendências [SBC 2018].

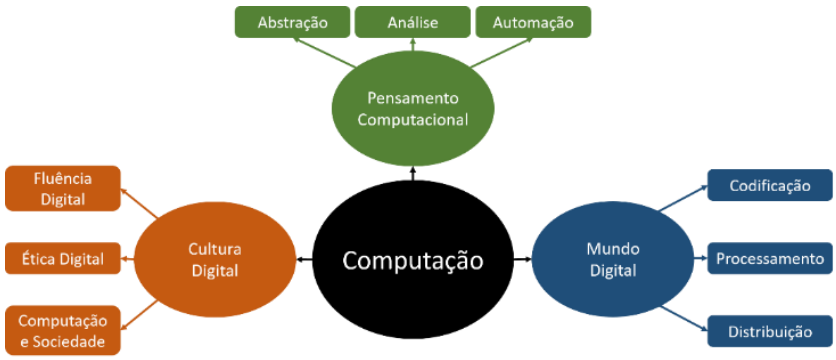

Figura 1. Eixos das Diretrizes da SBC (2018).

As progressões são feitas em anos letivos, conforme a organização da Educação Básica brasileira. O pensamento computacional é delineado pela abstração, análise e

automação (Figura 1). Enfocando a prática de programação como uma das formas de desenvolver o pensamento computacional, os conceitos iniciam com uma definição mais simples no primeiro ano do Ensino Fundamental aumentando de complexidade a cada ano (Tabela 1).

Tabela 1. Diretrizes da SBC para o Ensino Fundamental [SBC 2018].

Ano	Objeto de conhecimento	Habilidades
1 Ano	Organização de objetos	Organizar objetos concretos de maneira lógica utilizando diferentes características (por exemplo: cor, tamanho, forma, texturas, detalhes, etc.).
	Algoritmo: definição	Compreender a necessidade de algoritmos para resolver problemas.
		Compreender a definição de algoritmos resolvendo problemas passo-a-passo (exemplos: construção de origamis, orientação espacial, execução de uma receita, etc.).
2 Ano	Identificação de padrões de comportamento	Identificar padrões de comportamento (exemplos: jogar jogos, rotinas do dia-a-dia, etc.).
	Algoritmos: construção e simulação	Definir e simular algoritmos (descritos em linguagem natural ou pictográfica) construídos como sequências e repetições simples de um conjunto de instruções básicas (avance, vire à direita, vire à esquerda, etc.).
		Elaborar e escrever histórias a partir de um conjunto de cenas.
	Modelos de objetos	Criar e comparar modelos de objetos identificando padrões e atributos essenciais (exemplos: veículos terrestres, construções habitacionais, etc.).
3 Ano	Definição de problemas	Identificar problemas cuja solução é um processo (algoritmo), definindo-os através de suas entradas (recursos/insumos) e saídas esperadas.
	Introdução à lógica	Compreender o conjunto dos valores verdade e as operações básicas sobre eles (operações lógicas).
	Algoritmo: seleção	Definir e executar algoritmos que incluam sequências, repetições simples (iteração definida) e seleções (descritos em linguagem natural e/ou pictográfica) para realizar uma tarefa, de forma independente e em colaboração.
4 Ano	Estruturas de dados estáticas: registros e vetores	Compreender que a organização dos dados facilita a sua manipulação (exemplo: verificar que um baralho está completo dividindo por naipes, e seguida ordenando)
		Dominar o conceito de estruturas de dados estáticos homogêneos (vetores) através da realização de experiências com materiais concretos.
		Dominar o conceito de estruturas de dados estáticos heterogêneos (registros) através da realização de experiências com materiais concretos.
		Utilizar uma representação visual para as abstrações computacionais estáticas (registros e vetores).
	Algoritmo: repetição	Definir e executar algoritmos que incluem sequências e repetições (iterações definidas e indefinidas, simples e aninhadas) para realizar uma tarefa, de forma independente e em colaboração.
		Simular, analisar e depurar algoritmos incluindo sequências, seleções e repetições, e também algoritmos utilizando estruturas de dados estáticas.
5 Ano	Estruturas de dados dinâmicas: listas e grafos	Entender o que são estruturas dinâmicas e sua utilidade para representar informação
		Conhecer o conceito de listas, sendo capaz de identificar instâncias do mundo real e digital que possam ser representadas por listas (por exemplo, lista de chamada, fila, pilha de cartas, etc.).
		Conhecer o conceito de grafo, sendo capaz de identificar instâncias do mundo real e digital que possam ser representadas por grafos (por exemplo, redes sociais, mapas, etc.)
		Utilizar uma representação visual para as abstrações computacionais dinâmicas (listas e grafos).
	Algoritmos sobre estruturas dinâmicas	Executar e analisar algoritmos simples usando listas / grafos, de forma independente e em colaboração.
		Identificar, compreender e comparar diferentes métodos (algoritmos) de busca de dados em listas (sequencial, binária, hashing, etc.).
6 Ano	Tipos de dados	Reconhecer que entradas e saídas de algoritmos são elementos de tipos de dados.
		Formalizar o conceito de tipos de dados como conjuntos.
	Introdução à generalização	Identificar que um algoritmo pode ser uma solução genérica para um conjunto de instâncias de um mesmo problema, e usar variáveis (no sentido de parâmetros) para descrever soluções genéricas.
	Linguagem visual de programação	Compreender a definição de problema como uma relação entre entrada (insumos) e saída (resultado), identificando seus tipos (tipos de dados, por exemplo, número, string, etc).
		Utilizar uma linguagem visual para descrever soluções de problemas envolvendo instruções básicas de processos (composição, repetição e seleção).
		Relacionar programas descritos em linguagem visual com textos precisos em português.
	Técnicas de solução de problema: decomposição	Identificar problemas de diversas áreas do conhecimento e criar soluções usando a técnica de decomposição de problemas.
7 Ano	Automatização	Compreender que automatizar a solução de um problema envolve tanto a definição de dados (representações abstratas da realidade) quanto do processo (algoritmo).
	Estruturas de dados: registros e vetores	Formalizar o conceito de registros e vetores.
	Técnicas de solução de problemas: decomposição e reuso	Criar soluções para problemas envolvendo a definição de dados usando estruturas estáticas (registros e vetores) e algoritmos e sua implementação em uma linguagem de programação.
		Depurar a solução de um problema para detectar possíveis erros e garantir sua corretude.
	Programação: decomposição e reuso	Identificar subproblemas comuns em problemas maiores e a possibilidade do reuso de soluções.
		Colaborar e cooperar na proposta e execução de soluções algorítmicas utilizando decomposição e reuso no processo de solução.
8 Ano	Estruturas de dados: listas	Formalizar o conceito de listas de tamanho indeterminado (listas dinâmicas).
		Conhecer algoritmos de manipulação e pesquisa sobre listas.
	Técnicas de solução de problema: recursão	Identificar problemas de diversas áreas e criar soluções, de forma individual e colaborativa, usando algoritmos sobre listas e recursão.
		Empregar o conceito de recursão, para a compreensão mais profunda da técnica de solução através de decomposição de problemas.
	Programação: listas e recursão	Identificar o conceito de recursão em diversas áreas (Artes, Literatura, Matemática, etc.).

	Paralelismo	Compreender o conceito de paralelismo, identificando partes de uma tarefa que podem ser realizadas concomitantemente.
9 Ano	Estruturas de dados: grafos e árvores	Formalizar os conceitos de grafo e árvore.
		Conhecer algoritmos básicos de tratamento das estruturas árvores e grafos.
	Técnica de construção de algoritmos: Generalização	Identificar problemas similares e a possibilidade do reuso de soluções, usando a técnica de generalização.
	Programação: generalização e grafos	Construir soluções de problemas usando a técnica de generalização, permitindo o reuso de soluções de problemas em outros contextos, aperfeiçoando e articulando saberes escolares.
		Identificar problemas de diversas áreas do conhecimento e criar soluções, de forma individual e colaborativa, através de programas de computador usando grafos e árvores.

3. Metodologia de Pesquisa

O objetivo desta pesquisa é analisar o sequenciamento dos conteúdos referentes ao pensamento computacional conforme sugerido no eixo de pensamento computacional das diretrizes da SBC para o Ensino Fundamental. Para atingir esse objetivo, é realizado um estudo de caso para identificar a dificuldade observada dos objetos de conhecimento em um contexto particular seguindo Yin (2001) e Wohlin et al. (2012).

Definição do estudo. O objetivo do estudo é definido e decomposto de acordo com a definição de objetos de conhecimento da SBC (2018) e dos conceitos da área de algoritmos e programação, como uma das partes principais do ensino do pensamento computacional conforme *o K-12 Computer Science Standards* [CSTA 2017]. A definição dos itens é feita de forma sistemática por meio de uma rubrica (Tabela 2), a qual permite a definição de níveis de desempenho para cada item, relacionando-os a uma pontuação. Os itens são definidos enfocando conceitos de algoritmos e programação e conceitos presentes na linguagem de programação visual do ambiente App Inventor, uma das linguagens de programação mais populares na Educação Básica, cuja avaliação pode ser feita de forma automatizada a partir do código-fonte.

Tabela 2. Definição de rubrica relacionada ao pensamento computacional [Alves 2019].

Item	Insuficiente – 0 pontos	Básico – 1 ponto	Intermediário – 2 pontos	Avançado – 3 pontos
I01. Eventos	Não usa eventos.	1 tipo de manipulador de eventos é usado.	2 tipos de manipuladores de eventos são usados.	Mais de 2 tipos de manipuladores de eventos são usados.
I02. Variáveis	Não usa variáveis	Modificação ou uso de variáveis predefinidas.	Criação e operação com variáveis originais.	
I03. Strings	Não usa strings.	Uso de string para alterar textos de componentes visuais.	Criação e operação com strings originais.	-
I04. Operadores	Não usa operadores.	Usa operadores aritméticos.	Usa operadores relacionais.	Usa operadores booleanos (lógicos).
I05. Nomeação	Menos de 10% dos nomes são alterados.	10 a 25% dos nomes são alterados do padrão.	De 26 a 75% dos nomes são alterados do padrão.	Mais de 76% dos nomes são alterados do padrão.
I06. Condicionais	Não usa condicionais.	Uso de condicional simples (if then).	Uso de condicional completo (if then else).	Uso de condicionais (if then else, if then).
I07. Sincronização	Não usa temporizador para sincronização.	Uso de temporizador para sincronização.	-	-
I08. Abstração	Não define procedimentos.	Existe exatamente um procedimento e sua chamada.	Existem procedimentos para organização (várias definições de procedimentos).	Existem procedimentos tanto para organização quanto para reuso (mais chamadas do que definição de procedimento)
I09. Laços	Não usa laços.	Uso de laço simples (while)	Uso de laço com variável simples (For each)	Uso de laço com item de lista (For each).
I10. Listas	Não usa listas.	Usa uma lista unidimensional.	Usa mais de uma lista unidimensional.	Usa uma lista de tuplas (*map*).
I11. Persistência de dados	Não faz persistência de dados.	Usa persistência em arquivo.	Usa um banco de dados local.	Usa um banco de dados web.

Execução do estudo. Para maximizar o tamanho da amostra, foi realizado o *download* de todos os 88.864 aplicativos públicos disponíveis na Galeria do App Inventor em maio de 2018, armazenando seu código-fonte. A extração de dados foi feita

de forma automatizada avaliando os aplicativos por meio da ferramenta CodeMaster [Alves 2019]. Foram analisados com sucesso 88.812 aplicativos, pois devido a dificuldades técnicas, 52 projetos não puderam ser analisados. Os dados extraídos foram usados para a calibração de parâmetros usando o Modelo de Resposta Gradual [Samejima 1969] da TRI [Andrade 2000]. A partir dos parâmetros calibrados, os itens foram posicionados em uma escala de proficiência que permite a distinção entre o que é mais fácil e mais difícil do ponto de vista dos aprendizes [Andrade et al. 2000].

Análise e interpretação do estudo. Os itens posicionados na escala de proficiência são analisados em relação a sua dificuldade. As relações de ordem do posicionamento dos itens são comparadas com o sequenciamento das diretrizes da SBC discutindo similaridades e diferenças.

4. Resultados

4.1. Calibração dos parâmetros e posicionamento na escala

Para calibrar os parâmetros dos itens é utilizado o Modelo de Resposta Gradual (MRG) proposto por Samejima (1969) da TRI [Andrade et al. 2000]. O MRG assume que as categorias de resposta de um item são ordenadas entre si [Samejima 1969]. A métrica é estabelecida fixando-se os parâmetros populacionais em média = 0 e desvio-padrão = 1. O modelo baseia-se no fato de que indivíduos com maior habilidade possuem maior probabilidade de acertar um item e que esta relação não é linear.

A análise foi feita usando a linguagem R para calibrar o parâmetro *a* (inclinação) e os parâmetros *b* (dificuldade) de cada item. Como a rubrica contém itens politômicos ordinais, vários parâmetros *b* são gerados para diferenciar a passagem de uma pontuação a outra: *b2* = significa a dificuldade de obter pontuação 1 em qualquer item, *b3* = significa dificuldade de obter pontuação 2 em qualquer item, *b4* = significa dificuldade de obter pontuação 3 em qualquer item. Consequentemente, os itens que não têm descrição para a pontuação 2, também não têm um parâmetro *b3* (exemplo: item sincronização). Na TRI, os dois parâmetros *a* e *b* podem, teoricamente, assumir qualquer valor real entre $-\infty$ e $+\infty$. Porém, não se espera um valor negativo para o parâmetro *a*, tipicamente valores acima de 1,0 são considerados bons, pois indicam que o item discrimina bem aprendizes com diferentes habilidades. Neste estudo os parâmetros *b* são os principais indicadores a serem analisados, pois indicam a dificuldade do item. Para os parâmetros *b* são esperados valores próximos ou dentro do intervalo [-5, 5], sendo que valores negativos indicam que um item tem dificuldade abaixo da média e valores positivos indicam uma dificuldade acima da média.

Em geral pode-se observar que a maioria dos itens calibrou bem, com valores do parâmetro de inclinação (*a*) acima de 1 (Tabela 3). Além disso, os valores dos parâmetros de dificuldade (*b2*, *b3* e *b4*) ficaram dentro do intervalo [-5, 5], sendo que apenas o item 10 apresentou o parâmetro *b4* um pouco acima de 5. Os erros-padrão (EP) de cada parâmetro *b* apresentaram resultados similares e ficaram em ordens de magnitude baixa, portanto, não apresentando problemas de calibração. Analisando os resultados, pode-se inferir que obter 1 ponto no item I01 é mais fácil do que nos demais itens, pois esse item possui o menor parâmetro *b* (*b2* = -1,650). Por outro lado, obter 3 pontos no item I10 é mais difícil do que nos demais, pois apresenta o maior valor para um parâmetro *b* (*b4* = 5,204).

Tabela 3. Calibração dos parâmetros.

Item	*a*	EP(a)	*b2*	EP(*b2*)	*b3*	EP(*b3*)	*b4*	EP(*b4*)
I01. Eventos	2,877	0,022	-1,650	0,009	-0,902	0,006	-0,473	0,005
I02. Variáveis	2,971	0,022	-0,830	0,006	-0,009	0,005	-	-
I03. Strings	1,656	0,012	-0,568	0,007	0,942	0,008	-	-
I04. Operadores	3,081	0,024	-0,055	0,005	0,210	0,005	0,475	0,005
I05. Nomeação	1,680	0,012	-0,313	0,006	0,067	0,006	1,887	0,012
I06. Condicionais	2,323	0,017	0,344	0,005	0,796	0,006	1,571	0,009
I07. Sincronização	2,809	0,029	0,892	0,006	-	-	-	-
I08. Abstração	3,184	0,034	0,988	0,006	1,081	0,006	1,189	0,007
I09. Laços	1,766	0,027	2,138	0,021	2,293	0,023	2,575	0,027
I10. Listas	1,243	0,014	1,489	0,014	1,996	0,019	5,204	0,070
I11. Persistência	1,572	0,020	1,821	0,016	1,901	0,017	3,356	0,036

Com base nos parâmetros de dificuldade calibrados, os itens são posicionados numa escala (0,1), isto é, com média = 0 e desvio-padrão = 1 (Figura 2). A escala de habilidade é uma escala arbitrária onde o importante são as relações de ordem existentes entre seus pontos e não necessariamente sua magnitude. Os itens foram dispostos nos pontos da escala de acordo com os parâmetros de dificuldade (*b2, b3 e b4*) calibrados, conforme apresentados na Tabela 2. Exemplificando, o I01 (Eventos) possui parâmetro *b2* = -1,650, portanto, foi posicionado no ponto -1,5 da escala.

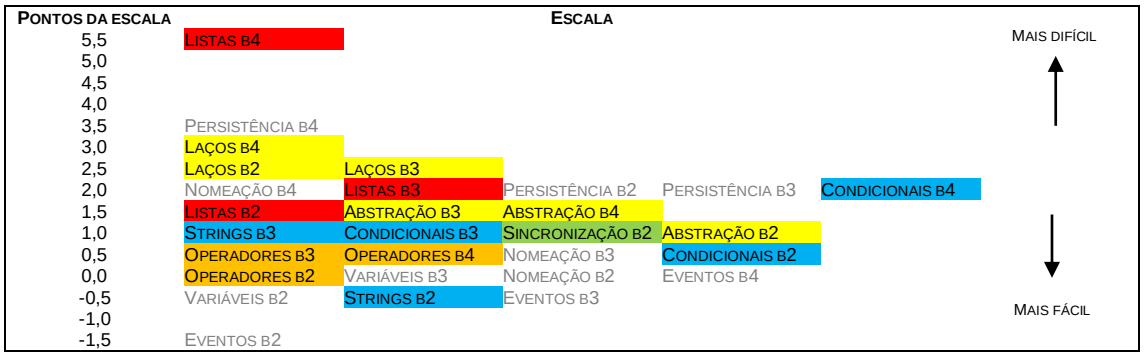

Figura 2. Posicionamento dos itens na escala.

Pode-se inferir a partir do posicionamento dos itens na escala que um item com um parâmetro *b* calibrado em 1,5 por exemplo, está 1,50 desvios-padrão acima da habilidade média. Assim, esse item é mais difícil do que todos os itens que foram posicionados abaixo do ponto 1,5 da escala. No contexto de programação de aplicativos com App Inventor, os itens mais fáceis de aprender incluem eventos, variáveis e strings (Figura 2), pois esses itens apresentam parâmetros *b* negativos (abaixo da média). Esses parâmetros estão semanticamente coerentes, haja vista o App Inventor estimula a criação de variáveis e o uso ilimitado de eventos [Turbak et al. 2014].

Os itens mais difíceis incluem listas, persistência e laços (*loop*). A pontuação 3 para o item lista (*map*) apresenta o maior parâmetro de dificuldade (Listas *b4*), sendo a mais difícil de atingir entre todas as demais definidas. Apesar de o item laços também ser considerado difícil, cabe ressaltar que o uso de comandos de laços nos programas App Inventor é pequeno, pois muitos processos iterativos que seriam expressos com laços em outras linguagens de programação são expressos como um evento que executa uma única etapa da iteração toda vez que é acionado [Turbak et al. 2014].

4.2. Comparação do sequenciamento da SBC com os resultados da análise via TRI

A partir dos resultados do posicionamento da escala, pode-se analisar o sequenciamento do conteúdo proposto pelas diretrizes da SBC (2018) em relação ao sequenciamento dos objetos de aprendizagem. Em geral podem-se observar discrepâncias entre o grau da

dificuldade percebido pela análise via TRI e a alocação dos objetos ao longo do Ensino Fundamental (Tabela 4).

Tabela 4. Comparação do sequenciamento da SBC com a TRI.

Sequenciamento do eixo Pensamento Computacional das diretrizes da SBC no Ensino Fundamental			Sequenciamento baseado na TRI										
			← Mais fácil										Mais difícil →
			I01. Eventos	I02. Variáveis	I03. Strings	I04. Operadores	I05. Nomeação	I06. Condicionais	I07. Sincronização	I08. Abstração	I09. Laços	I10. Listas	I11. Persistência
Mais fácil →	1 Ano	Organização de objetos											
		Algoritmo: definição											
	2 Ano	Identificação de padrões de comportamento											
		Modelos de objetos								X			
		Algoritmos: construção e simulação									X		
	3 Ano	Definição de problemas											
		Algoritmo: seleção				X		X			X		
		Introdução à lógica											
	4 Ano	Estruturas de dados estáticas: registros e vetores										X	
		Algoritmo: repetição									X		
	5 Ano	Estruturas de dados dinâmicas: listas e grafos										X	
		Algoritmos sobre estruturas dinâmicas										X	
	6 Ano	Tipos de dados			X								
		Linguagem visual de programação						X			X		
		Introdução à generalização								X			
		Técnicas de solução de problema: decomposição								X			
	7 Ano	Automatização								X			
		Estruturas de dados: registros e vetores										X	
		Técnicas de solução de problemas: decomposição e reuso								X			
		Programação: decomposição e reuso								X			
← Mais difícil	8 Ano	Estruturas de dados: listas										X	
		Programação: listas e recursão										X	
		Técnicas de solução de problema: recursão											
		Paralelismo							X				
	9 Ano	Estruturas de dados: grafos e árvores											
		Programação: generalização e grafos								X			
		Técnica de construção de algoritmos: Generalização								X			

Em termos do sequenciamento dos objetos de conhecimento, observa-se que as diretrizes não seguem a relação de dificuldade encontrada na análise via TRI. Por exemplo, a proposta indica abordar conceitos de listas já no 5º ano, mesmo sendo um dos conceitos considerados mais difíceis. As diretrizes também preveem o ensino de conceitos como recursão (8º ano) e grafos (9º ano) no Ensino Fundamental, considerados complexos cuja abordagem tipicamente é feita somente a partir do Ensino Superior. Além disso, as diretrizes propostas pela SBC não abordam completamente todos os conceitos referentes a algoritmos e programação, tais como eventos.

Outros *frameworks*, como o proposto pelo CSTA (2017), apresentam um sequenciamento mais alinhado à escala de proficiência criada de acordo com a estimação da dificuldade dos conteúdos. Diferentemente das diretrizes da SBC, esses *frameworks* propõem o ensino de itens considerados mais fáceis, tais como variáveis e strings, antes de itens mais difíceis, como abstração. Itens considerados complexos, como listas, são abordados apenas nos anos finais do Ensino Fundamental ou no Ensino Médio e não nos anos iniciais do Ensino Fundamental conforme aborda a SBC.

4.3. Ameaças à validade

Foram identificadas ameaças potenciais à validade, às quais este estudo está sujeito e aplicadas estratégias de mitigação para minimizar seu impacto nos resultados. Um risco é relacionado ao agrupamento de dados de diferentes contextos. Os programas do conjunto de dados vêm de diversos contextos da comunidade mundial do App Inventor,

e nenhuma informação adicional sobre o histórico dos criadores dos projetos da Galeria App Inventor está disponível. No entanto, como o objetivo é a relação de ordem entre as dificuldades dos conteúdos, e não sua magnitude, isso não é considerado um problema. Outra ameaça referente à possibilidade de generalizar os resultados está relacionada ao tamanho da amostra e ao uso de código de somente uma linguagem de programação. Sendo uma das principais linguagens de programação visual que contém grande parte do conteúdo de algoritmos e programação abordado na Educação Básica e similar a outras linguagens utilizadas, como Scratch, esse risco é minimizado utilizando uma quantidade significativa de aplicativos (mais de 88 mil). Assim, o tamanho de amostra é considerado satisfatório, permitindo a geração de resultados significativos. Em relação à validade do construto foram definidas medidas de forma sistemática e a extração de dados foi feita de forma automatizada eximindo erros de uma extração manual. A técnica estatística utilizada para a análise foi escolhida com base na literatura, sendo uma das técnicas indicadas para esse fim. Cabe ressaltar que os resultados deste estudo são aplicáveis no contexto de ensino do pensamento computacional usando uma linguagem de programação visual ao invés de atividades desplugadas e similares.

5. Conclusão

Foram analisadas as diretrizes da SBC, especificamente para o eixo do pensamento computacional, a partir de objetos de conhecimento relacionados a algoritmos e programação. Observou-se que as diretrizes estão coerentes com as competências gerais da BNCC. No entanto, a dificuldade de aprendizagem observada na prática usando uma linguagem de programação visual não é refletida no sequenciamento dos objetos de conhecimento. Assim, os resultados dessa análise podem ser aproveitados para sistematicamente discutir e melhorar o sequenciamento pedagógico das diretrizes adotando técnicas de *scaffolding* e comparações com outros *frameworks* de referência.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e do Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil (CNPq).

Referências

ACARA (2015) “Australian Curriculum, Assessment and Reporting Authority”, https://www.acara.edu.au/

Alves, N. da C. (2019) “CodeMaster: Um Modelo de Avaliação do Pensamento Computacional na Educação Básica através da Análise de Código de Linguagem de Programação Visual”, Dissertação (Programa de Pós-Graduação em Ciência da Computação (PPGCC)) – Universidade Federal de Santa Catarina.

Andrade, D. F., Tavares, H. R., Valle, R. C. (2000) “Teoria da Resposta ao Item: Conceitos e Aplicações”. São Paulo: Associação Brasileira de Estatística.

CAS (2015) “Computing at School”, https://www.computingatschool.org.uk/

CIEB (2018) “Currículo de Referência em Tecnologia e Computação”, Centro de Inovação para a Educação Brasileira, http://curriculo.cieb.net.br/

CSTA (2016/2017) “Computer Science Framework/Standards”, Computer Science Teachers Association.

Dagostini et al. (2018) “URI Online Judge Blocks: Construindo Soluções em uma Plataforma Online de Programação”, In: Anais do Simpósio Brasileiro de Informática na Educação, Fortaleza, Brasil.

Daniel, G. T. et al. (2017) “Ensinando a Computação por meio de Programação com App Inventor”, In: Anais do Computer on the Beach, Florianópolis, p. 357-365.

Franklin, D. et al. (2017) “Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum”. In: Proc. of the Technical Symposium on Computer Science Education. New York, EUA.

Grover, S., Pea, R. (2013) “Computational Thinking in K–12 A review of the state of the field”, Educational Researcher, v. 42, n. 1, p. 38-43.

Hlebowitsh, P. (2010) “Scope and Sequence, in Curriculum Development”. In: Encyclopedia of Curriculum Studies, SAGE Publications.

Lye, S. Y., Koh, J. H. L. (2014) “Review on teaching and learning of computational thinking through programming: What is next for K-12?”. Computers in Human Behavior, v. 41(C), p. 51-61.

MEC (2018) “Base Nacional Comum Curricular”, Brasil.

Ormrod, J. E. (2018) “Essentials of educational psychology: big ideas to guide effective teaching”, New York: Pearson (5 ed.).

Ortiz J. S. e Pereira R. (2018) "Um Mapeamento Sistemático Sobre as Iniciativas para Promover o Pensamento Computacional", In: Simpósio Brasileiro de Informática na Educação, Fortaleza, Brasil.

Rich, K. M, et al. (2018) “K-8 learning trajectories derived from research literature: sequence, repetition, conditionals”, ACM Inroads, vol. 9, n. 1, p. 46-55.

Samejima, F. A. (1969) “Estimation of latent ability using a response pattern of graded scores”. Psychometric Monograph, v. 17.

Santos, P. S. C., Araujo, L. G. J., Bittencourt, R. A. (2018) “A Mapping Study of Computational Thinking and Programming in Brazilian K-12 Education”. In: Proc. of the 48th Annual Frontiers In Education Conference, San Jose, EUA.

SBC (2018) “Diretrizes para ensino de Computação na Educação Básica”, Sociedade Brasileira de Computação.

Seiter, L., Foreman, B. (2013) “Modeling the learning progressions of computational thinking of primary grade students”. In: Proc. of the 9th Annual Int. ACM Conference on International Computing Education Research, New York, EUA.

Silva, R. C. S. et al. (2018) “Adaptabilidade de Objetos de Aprendizagem usando Calibragem e Sequenciamento Adaptativo de Exercícios”. Revista Brasileira de Informática na Educação - RBIE, vol. 26, n. 1, p. 70-90.

Wasson, B. (1990), “Determining the focus of instruction: Content planning for intelligent tutoring systems”, Ph.D. thesis, University of Saskatchewan, Canada.

Wing, J. M. (2006) “Computational thinking”. Communications of the ACM, v. 49, n. 3, p. 33–35.

Wohlin C. et al. (2012) “Experimentation in Software Engineering”. Berlin: Springer.

Yin, R. K. (2017) “Case study research: design and methods”, (6ª Ed.) Thousand Oaks: SAGE Publications.