Kinesthetic Touches For a Theory of Computing Class

Judy Goldsmith
University of Kentucky
329 Rose St.
Lexington, KY 40509
goldsmit@cs.uky.edu

Radu Paul Mihail
Valdosta State Univeristy
1500 N Patterson St.
Valdosta, GA 31698
rpmihail@valdosta.edu

ABSTRACT
Theory of computing courses do not usually evoke memories of fun for most students. Instead, fragments of formalisms from automata theory and memories of the existence of rigorous proofs come to mind. In this paper we bring arguments from educational psychology and pedagogy that movement and fun can be a part of the teaching and learning process in theory of computing courses. Kinesthetic Learning Activities (KLAs) consist of physical interactions between the students and their environment in order to achieve or optimize an educational objective. We propose a set of KLAs and performances to assist teachers with topics that most students find hard to grasp. Our hope is to inspire CS educators to not only adopt and modify these KLAs, but to create new ones. Our greater hope is that words like “fun” and “games,” along with more accurate understanding of important results in theoretical computer science, come to mind when students are asked to describe their theory course.

1. INTRODUCTION
This is a paper about real practices in real Theory of Computing classes that bring motion into the teaching and/or learning process. We argue that there are pedagogical reasons to involve motion in what has often been taught in the best formal mathematics tradition.

Our university offers theory of computing courses at the undergraduate and graduate level. The syllabi for both include regular languages, deterministic and nondeterministic automata, and their equivalence, as well as proofs that there are non-regular languages; context-free grammars and pushdown automata, and their equivalence; Turing machines and their variants, a brief introduction to decidability, and to computational complexity. Our undergraduate course is considered part of the discrete math sequence, and also includes a section on logic, at the cost of depth and rigor in the other topics. For the undergraduate course, student learning outcomes include:

- Fluency in the elements of automata theory
- Basic understanding of properties of formal languages and associated concepts including grammars and regular expressions
- Basic understanding of models of computation including Turing machines

The undergraduate course is required for computer science majors; the graduate course is a core course, and can be replaced by an algorithms course. Thus, the graduate course is usually taken by students who are more comfortable with mathematical formalisms. The undergraduate course is an oft-postponed challenge for many of our students.

1.1 Order Matters
The standard order for a traditional theory of computing course is based on the Chomsky hierarchy of formal languages. It starts with regular languages (and automata theory), then introduces context-free languages, with their corresponding grammars and pushdown automata. Some courses then cover context-sensitive languages, but many have traded that topic for more time for Turing machines and their model variants, computability, and then a brief introduction to computational complexity, ending with NP-completeness.

It is common to end the course with NP-completeness in the last week of classes, and the halting problem soon before that, although anyone who has taught for a few years, or has recently been a student, knows that those last few weeks are the time when students are least likely to absorb and understand hard concepts. In other words, computability and complexity often go flying past even the better students while those students are busy with major programming projects, emergency extra-credit assignments, and general end-of-semester panic.

We have taken to introducing the course with Turing machines. This has two major advantages. First, it eases them into the notion of formal models of computation with something that is “Turing complete,” meaning that it can compute whatever their laptops/watches/supercomputer accounts can compute. Secondly, it introduces the hard parts of the course (undecidability and NP-completeness) before their brains are full from the rest of the semester.

One side effect of ending the course with CFLs and pushdown automata is that the students can make connections to courses on compiler design, parsers, and other basic computer science concepts, so they go out believing that this is a relevant course for the curriculum. A secondary effect is that the beliefs that they can master the (recent) material, and that it is relevant, seem to have a positive effect on teaching evaluations (at the end of the semester).

2. KINESTHETIC LEARNING

Bloom's taxonomy of educational objectives [13] remains an important tool to improve student learning through systematic analysis and research of learning goals and process. This taxonomy, and later revisions, e.g., [4], suggest that learning can be divided into three domains: cognitive, affective and psychomotor, or kinesthetic. Anderson [4] and others in the educational psychology community (e.g., [36, 20, 35, 23, 37]) encourage educators to consider all three domains, and include aspects thereof, when designing lessons.

Given our physical senses, we perceive information visually (e.g., white/black board, overhead projector with text, pictures, diagrams, videos), auditorily (sounds) and kinesthetically (taste, touch and smell). A growing body of literature suggests that as people mature, the majority specialize in one input modality (i.e., they become visual, auditory or kinesthetic learners), though some rely on multiple input modalities [17]. Teachers tend to focus their teaching methods on the input modality that works best for them [6]. Thus, those students whose primary modality is not in use in the classroom are at a real disadvantage.

Kinesthetic Learning Activities (KLAs) involve physical interaction between the students and their environment. Begel et al. [7] argue that research on kinesthetic learning in higher education is sparse, and even more sparse in engineering related fields. KLAs offer students a change to literally and physically participate in their own education, and thus have potential to energize an otherwise static group of learners. For the interested reader, Begel [7] puts forward a few guidelines on designing KLAs, and recommends that future research be directed toward assessing KLAs and building a set of best practices.

Mathematics provokes anxiety for a variety of students. One might think that teaching abstract, rigorous mathematical thinking, by its nature, must be dry of artistic expression. Karl Schaffer and Erik Stern [27] are determined to change that. The two men perform and choreograph where modern dance intersects mathematics. Combinatorics and set theory exhibit symmetries and structures that Shaffer and Stern bring to life using dance. For example, one of their routines involves attempting a hand-shake that fails numerous times, resulting in the dancers falling past each other. In a recursively defined set whose basis clause is the dancers, and the inductive clause joins two arbitrary elements using a hand-shake, the dance demonstrates the various possibilities and constraints of set building.

In their book [32], Shaffer and Stern propose dance activities and practical activities for teachers to use in the classroom. For example, distinguishing symmetries induced by reflection, rotation and translation is practiced using simple movements and relative positioning of a group leader and a group of followers.

A motivated teacher, excited to explore kinesthetic learning, faces several obstacles: students' reluctance to participate, students' failure to understand the point of the exercise, or the teacher's own failure from losing control of the activity [30]. Student reluctance to participate in general problem-solving, according to Felder and Brent [16], is in part due to a sudden withdrawal of a support structure given to students by teachers from the first grade on. Pollard et al. [30] argue that this problem can be circumvented by adding elements of kindergarten activities such as toys and play. The idea is simple: toys materialize abstract concepts and playing with the toys can help build internal representations of complex ideas or algorithms.[1]

Attempts at enriching computer science students' learning through kinesthetic enhancements has been done in the past. For example, Silviotti et al. [33] suggest KLAs for a distributed computing course, where algorithms are enacted by people and data structures built from people. The authors argue that these activities promote the understanding of concurrency through simultaneously active agents, and locality of scope through physical limits. The authors describe several distributed algorithms enacted kinesthetically.

Most of us have seen videos of folk-dance troupes performing sorting algorithms [1]. The site csteaching tips points out that YouTube abounds in videos of CS students doing similar exercises, and algorithms teachers can bring these dances to their classrooms [24].

Dr. Craig Tovey, at Georgia Tech, brings to life Dantzig's simplex method [14] for finding the optimal solution(s) to linear programs,[2] using students as columns of a matrix and rubber bands for the edges of a simplex (hypertetrahedron). He illustrates the concept of pivoting through an edge of the rubber band simplex that remains static as one of the simplex's vertices moves to a better choice. This lesson is publicly available on YouTube at: https://www.youtube.com/watch?v=Ci1vBGn9yRc.

Friss de Kereki [15] proposes and evaluates a set of KLAs for an object oriented (OO) flavor of CS1. The experimental design had two control groups and one intervention group. The author reports higher motivation, satisfaction and passing rates for the intervention group. Similarly, Adorjan et al. [2] propose improvements to an OO CS1 using Gardner's taxonomy of intelligence [19], dubbed multiple intelligence (MI). Adorjan et al. provide a list of activities suitable for CS1, along with a mapping to MIs. For Gardener's MI4 (bodily-kinesthetic intelligence), Adorjan proposes five activities.

The CS Unplugged movement, started by Mike Fellows, has accumulated a book [12] and website [22] with many activities suitable for outreach to kids as well as for the CS classroom. Much has been written about the use of these and similar activities, e.g., [8, 9, 10, 11, 18, 21, 26], and some about the efficacy thereof [34].

Peacock [29] looks at potential teacher learner style mismatches for English as a foreign-language instruction. He cites Reid's [31] hypothesis that claims such mismatches cause learning failure, frustration and demotivation. Pea-

[1]One of the authors, old enough to remember the New Math, remembers using wooden blocks to count in various bases — though perhaps that was after kindergarten.

[2]Not to be confused with Nelder and Mead's [25] unconstrained optimization method based on the same geometric simplex.

cock gives teachers several recommendations to accommodate a balanced style. For example, kinesthetic learners should be given problem-solving activities, role-play and drama, and encouraged to actively participate in the learning environment. He also notes that Chinese learners prefer kinesthetic and auditory learning styles. This finding is also supported by Park [28].

Ambudkar [3] presents a dance activity to support student learning of the interoperability and communication between the layers in the open systems interconnections (OSI) model. The dance team consisted of seven rows (corresponding to the seven layers of the OSI model) and two columns (corresponding to the sender and receiver). The dancer's movements indicated how data moved between layers and between the sender and receiver.

Anewalt [5] includes toys, games and the Alice programming environment to foster an active learning approach for CS0 students. Algorithmic thinking and accuracy were reinforced by asking students to write instructions for making peanut butter and jelly sandwiches. The instructor later unveiled tools and materials, then randomly paired students were asked to modify their algorithms to use only the tools and materials available. The students then precisely followed their algorithms to make the sandwiches. She describes other activities related to basic programming principles, such as cookie-cutter and cut-out shapes to illustrate difference between classes and objects, "fishing" the argument (miming the action of fetching the argument value) to assign a parameter during a function call and the data transfers therein, as well as others.

Theory of computing courses tend to be devoid of physical activity,[3] and are thus optimizable by adding relevant and purposeful dramatic, visual, and physical elements to cater to kinesthetic learners. Our contribution to this aspect of computer science education consists of a set of kinesthetic learning activities, which we hope will be adopted by and improved upon by others.

3. KINESTHETIZING THE THEORY OF COMPUTATION CLASS

For this section, we assume that the reader has seen a theory of computing class, though details may have been forgotten. While the statement of the pumping lemma is destined to be forgotten, we speculate that drawing smiley faces for accept states on finite automata and frowning faces for reject states might make basic finite automata definitions more memorable.

3.1 Power Set Construction

In many classes labeled "Theory of Computing" or "Finite Automata and Formal Languages," regular languages are introduced early on. They might be defined by deterministic or (equivalently) nondeterministic finite automata, regular expressions, and/or regular grammars. Eventually, most classes claim or prove that whichever of those models were introduced are equivalent. The proof that any nondeterministic finite automaton N has an equivalent deterministic finite automaton D is referred to as the power-set proof. The construction starts by defining the start state of D, s_0^D, to be the set containing the start state, s_0^N, of N and any states reachable from s_0^N by ϵ-transitions (where ϵ is the empty string). Then, for each state s^D in D corresponding to a set S of states of N, and each element σ of the alphabet Σ, we create a state (if it isn't already in D) consisting of the set of states reachable from any state in S via σ followed by ϵ-transitions. We label the transition by σ.

[3]Here we discount the vigorous writing of mathematical formalisms with chalk or dry-erase markers that some instructors do, and the physical acts of note-taking in which we wish students would engage.

We then might prove that the two automata compute the same language. What we hope is that the students understand the construction, and we might ask them to perform it on a homework or exam.

Dr. Craig Tovey, at Georgia Tech, has his students enact the construction. First, they simulate a deterministic finite automaton. He writes each character of the string on a separate sheet of paper. Students holding one sheet each line up as the input string on one side of the room. Once a student's character is "read" the student throws the sheet into a trashcan and has to sit on the floor on the other side of the room. This conveys the one-time read process.

For the power set construction, he brings in cardboard squares with state numbers, and draws a nondeterministic automaton on the board. Students volunteer to "be" the states of the nondeterministic automaton.

Tovey then asks the start state to climb onto a table. For each symbol in the string, for each student on the table, the student points out those states that are reachable from himself via that symbol. The reachable states then take their place on the table and the others climb down.

"Does it get crowded on the table?" we asked.

"Yes."

"Isn't there a danger that the table will flip over?" we asked.

"That's what makes it so exciting," he said.

We have tried this exercise. to the amusement of the class. Tovey also simulates a pushdown automaton in class. It's the same except for an extra line of people to be the elements in the stack.

3.2 Proving a Language is not Regular

There are two approaches to showing that a language is not regular. The more commonly taught one is the Pumping Lemma, which says that a language L is regular iff there is an n [the size of the minimal deterministic finite automaton for L] such that for all strings $s \in L$ of length $> n$, there are x, y, z such that $|xy| \leq n$ and $|y| > 0$ such that for all $i \in N$, $xy^iz \in L$.

Got that? The quantifier sequence after the "iff" is

$$\exists n \forall s \exists x, y, z \forall i.$$

This is really hard for most students to grasp, much less apply. However, the underlying idea is a simple application of the pigeonhole principle: if we have a finite automaton with n states, and a string with more than n symbols, then when the automaton processes the string, at least one state must repeat. Therefore, the path from that state back to itself can be repeated arbitrarily many times (or excised, when $i = 0$) before the string leads to an accept state.

In order to cement the idea of a repeated loop, we ask all the students in the class to draw a circle in the air with their fingers. Since we have not yet tried the Power Set Construction exercise, this is usually our first class participation exercise. At first many are reluctant, but are willing to humor the crazy teacher.

The second approach to showing a language is not regular is via the Myhill-Nerode Theorem. This begins by defining an equivalence relation R_L for a language L. We say xR_Ly iff for all strings z, $[xz \in L \Leftrightarrow yz \in L]$. It is difficult to show this, but not so difficult to show that x and y are not equivalent. For instance, for the language $L = \{1^n0^n | \ n \in N\}$, consider $x = 1^i$ and $y = 1^j$ for $i \neq j$, and let $z = 0^i$. Then $xz \in L$ but $yz \notin L$.

The Myhill-Nerode Theorem states that L is regular iff R_L has finitely many equivalence classes. (These correspond to states in a minimal finite automaton that accepts L.) To show that L is not regular, one shows (as in the previous example) that there are infinitely many non-equivalent strings. In the previous paragraph, we have argued that for any two distinct natural numbers i and j, 1^i is not L-equivalent to 1^i.

Dr. Mike Fellows says that he explains the notion $[xz \in L \Leftrightarrow yz \in L]$ by starting his hands in different places (representing x and y), making the same series of moves (z), and ending up in the same place. We conjecture that having the students do this, and also have them end up in different places, might help make the equivalence relation more memorable, if not clearer.

3.3 Formal Grammars and Buffalo

It is well known that the word "buffalo" can be a noun, verb, and adjective. For instance, "Buffalo buffalo Buffalo buffalo" might be parsed as "Bovines harass other bovines from a city in upstate New York." Thus, one can illustrate a parse tree for English without having to resort to complex vocabulary. However, one might rightly be concerned about shorter sentences, since this verb is transitive: the subject of the sentence must have someone or something to buffalo. However, there is a tap dance sequence called a buffalo, because the rhythm of the taps matches that of the word. Thus, "Buffalo buffalo" is a valid sentence.

Teachers eager to learn the step are advised to consult YouTube or their local tap dance instructor. It is worth noting that (a) the move involves a shuffle step, and (b) it is often used in tap routines as an exit. Thus, one can shuffle off with buffalos.

3.4 Enacting Turing Machines

When we teach about Turing machines, we often start by writing a TM program to decide the language of palindromes over a small alphabet, say, $\{0, 1\}$. This is convenient because the language of palindromes is not regular, and also (as we later mention) can be recognized in time $\mathcal{O}(n^2)$ with a one-headed, one-tape TM, but in time $\mathcal{O}(n)$ by a two-tape or two-headed TM. We later show that there's a simulation of a 2-tape TM by a 1-tape TM with an $\mathcal{O}(n^2)$ time blowup; the language of palindromes is an example where we see the quadratic speedup directly.

In order to describe the TM, we act out the states: "I'm remembering I saw a 1. Is this a blank? No. Move right. I saw a 1. Is this a blank?...." (This is said while moving in the same direction, one step per "Move right". Ideally, that direction should be to the students' right, rather than the instructor's.)

We have found, over the years, that having students attribute such statements to states and state/symbol/action tuples cuts down on the instances of students randomly writing down states and transitions just to write something.

3.5 The Infinite Loop Dance

One of the central concepts in teaching about Turing machines (or your favorite sequential model of computation) is the notion of an infinite loop. For instance, to show that a decidable language L is semi-decidable, one can modify the Turing machine that decides L by replacing every "No" output by an infinite loop. But if a professor asks students, "can you write an infinite loop," a surprising number of them refuse to admit that they can — meaning that they won't admit that they *have* done so.

The two simplest Turing machine infinite loops are the one-state loop that ignores the input and moves the read head right, and the one that ignores the input and moves right, left, right, left, etc.

Both can be demonstrated in choreography. The first should involve the professor side-stepping at least as far as the door, if not out into the hallway. The second, which does not require leaving the room, is to step right, then left, then right, then left, then right, then left, then involves a moue and a vocalization of the "etc." It doesn't hurt to explain out loud that the "etc." is part of the Infinite Loop Dance.[4]

3.6 Undecidability

The proof that the Halting Set is undecidable is both elegant and, at first introduction, usually baffling. We feel strongly that the students should see it, but we don't think that it is terribly persuasive. In fact, to many students, it looks like a conjuring trick. We spend some time setting up another argument that not all languages are decidable. The argument is as follows.

1. There are at least two sizes of infinity, countable and uncountable.

[4]For one of us, there is a soundtrack to this dance. It is the Romanian folkdance tune Alunelul. The pattern, if you include stamps in the step count, is 7,7,7,7,4,4,4,4,2,2,3,2,2,3. Or, to the right 5 steps, stamp stamp, to the left 5 steps, stamp stamp, repeat; to the right 3 steps, stamp, to the left 3 steps, stamp, repeat; to the right 1 step, stamp, to the left 1 step stamp, to the right 1 step, stamp stamp, to the left 1 step, stamp, to the right 1 step, stamp, to the left 1 step, stamp stamp.

2. The set of finite strings over a finite alphabet Σ of size $|\Sigma| > 1$ is countable, but the set of infinite sequences over Σ is uncountable.

3. Turing machines can be encoded as strings over a finite alphabet. Therefore there are countably many TMs, and thus, countably many decidable languages.

4. There are uncountably many infinite binary sequences, and thus uncountably many languages. Therefore, not all languages are decidable.

To set this up, we start with questions about Hotel Infinity.[5] First, "One dark and stormy night, you arrive at Hotel Infinity. It's at the edge of the universe. You are told that all the rooms are full, but you need a room to sleep in. Can you find a room without making anyone leave?"

Each question is left hanging until the next class period, or until someone asks (sometimes several classes later). After they get the idea that everyone can move from room n to room $n+1$ and still have a room, they usually find a way to house "you, and infinitely many of your friends" by sending folks in room n to room $2n$. They often get stuck on "and each of your friends has a disjoint set of infinitely many friends," but you can remind them that powers of primes form infinitely many disjoint sets. (There are many proofs that there are infinitely many primes. The one where you assume there are only finitely many, $\{p_0, p_1, \ldots, p_q\}$ and then consider the number $(\Pi_{i \leq q} p_i) + 1$ — a number not on that list, and not divisible by any prime — allows us to dissect a proof by contradiction. It's useful to set up that structure, from "Spoze not" to the $\Rightarrow\Leftarrow$ contradiction mark, as a template.)

By this time, the students are reasonably comfortable with infinity. They have shown that $\aleph_0 \times \aleph_0 \cong \aleph_0$.[6]

That's when we prove that the set of subsets of $\mathbf{N}$, or the set of infinite binary sequences, or ... is uncountable.

This is usually in the second week of class. We promise that this is as hard a concept to wrap one's head around as any in the class. They disbelieve, but hope.

Later, when we introduce the notion of a universal TM, we introduce the notion of a TM as a (finite) string. We stop and ask how many TMs there are. After some discussion, they usually conclude that the answer is "countably infinitely many". They are often pleased to be using this notion that they thought was just introduced, earlier in the semester, to amuse and/or baffle them.

When we have completed the standard Halting Problem proof by contradiction, we say, "But you already knew that not all languages are decidable." We walk them through the proof that there are uncountably many languages (the same proof by diagonalization that we gave to show that there was an uncountable set), and remind them that there are only countably many decidable languages.

[5]This is not original to us, and we are not sure of the origin.
[6]$\aleph_0$ is the cardinality of $\mathbf{N}$, the set of natural numbers; $\aleph_0$ is the size of any countably infinite set.

4. CONCLUSIONS

We have not done controlled experiments to see whether tracing a loop with their fingers helps students comprehend or remember the pumping lemma. We can, however, say with great confidence that the students woke up when asked to do so. We can also say that our own students reference the infinite loop dance during and after class. In fact, when we taught it to a colleague's class, the colleague put it into his final exam. ("For 1 point, get up and do the infinite loop dance.") Not only did all the students, one by one, do it, but they asked their professor to do it as well. For those students with math (or formalism) anxiety, the performance aspect of choreographed theory, and the chance to move around in goofy ways in the class, gives them a reason to come to class.

5. REFERENCES

[1] `Algo-Rythmics`. http://algo-rythmics.ms.sapientia.ro/ and https://www.youtube.com/user/AlgoRythmics, 2013. Accessed August 2, 2015.

[2] A. Adorjan and I. Friss de Kereki. Multiple intelligence approach and competencies applied to computer science 1. In *Frontiers in Education Conference, 2013 IEEE*, pages 1170–1172. IEEE, 2013.

[3] B. Ambudkar. Introducing network design to students via a dance activity. In *Technology for Education (T4E), 2013 IEEE Fifth International Conference on*, pages 123–126. IEEE, 2013.

[4] L. W. Anderson, D. R. Krathwohl, and B. S. Bloom. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon, 2001.

[5] K. Anewalt. Making CS0 fun: an active learning approach using toys, games and alice. *Journal of Computing Sciences in Colleges*, 23(3):98–105, 2008.

[6] W. B. Barbe and M. N. Milone Jr. What we know about modality strengths. *Educational Leadership*, 38(5):378–80, 1981.

[7] A. Begel, D. D. Garcia, and S. A. Wolfman. Kinesthetic learning in the classroom. In *ACM SIGCSE Bulletin*, volume 36, pages 183–184. ACM, 2004.

[8] T. Bell, J. Alexander, I. Freeman, and M. Grimley. Computer science without computers: new outreach methods from old tricks. In *Proceedings of the 21st Annual Conference of the National Advisory Committee on Computing Qualifications*, 2008.

[9] T. Bell, J. Alexander, I. Freeman, and M. Grimley. Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1):20–29, 2009.

[10] T. Bell, P. Curzon, Q. Cutts, V. Dagiene, and B. Haberman. Introducing students to computer science with programmes that don't emphasise programming. In *Proceedings of the 16th Annual Joint conference on Innovation and Technology in Computer Science Education*, pages 391–391. ACM, 2011.

[11] T. Bell, F. Rosamond, and N. Casey. Computer science unplugged and related projects in math and computer science popularization. In *The Multivariate Algorithmic Revolution and Beyond*, pages 398–456. Springer, 2012.

[12] T. Bell, I. H. Witten, M. Fellows, R. Adams, J. McKenzie, M. Powell, and S. Jarman. *CS Unplugged*. Lulu.com, 2015.

[13] B. S. Bloom. *Taxonomy of Educational Objectives: The Classification of Education Goals. Cognitive Domain. Handbook 1*. Longman, 1956.

[14] G. B. Dantzig, A. Orden, P. Wolfe, et al. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

[15] I. F. de Kereki. Incorporation of kinesthetic learning activities to computer science 1 course: Use and results. *CLEI Electronic Journal*, 13(2), 2010.

[16] R. M. Felder and R. Brent. Navigating the bumpy road to student-centered instruction. *College Teaching*, 44(2):43–47, 1996.

[17] R. M. Felder and L. K. Silverman. Learning and teaching styles in engineering education. *Engineering Education*, 78(7):674–681, 1988.

[18] M. Fellows, T. Bell, and I. Witten. Computer science unplugged. *Computer Science Unplugged*, 2002.

[19] H. Gardner. *Multiple Intelligences: The Theory in Practice*. Basic books, 1993.

[20] T. F. Hawk and A. J. Shah. Using learning style instruments to enhance student learning. *Decision Sciences Journal of Innovative Education*, 5(1):1–19, 2007.

[21] P. Henderson. Computer science unplugged. *Journal of Computing Sciences in Colleges*, 23(3):168–168, 2008.

[22] S. Jarman, T. Bell, and I. Freeman. CS Unplugged. Accessed August 2, 2015.

[23] G. P. Krätzig and K. D. Arbuthnott. Perceptual learning style and learning proficiency: A test of the hypothesis. *Journal of Educational Psychology*, 98(1):238, 2006.

[24] C. Lewis, T. McKlin, T. Berry, and A. Schlesinger. csteachingtips. `http://csteachingtips.org/tip/use-physical-activities-demonstrate-sorting-algorithms-and-help-students-build-intuition-about`. Accessed August 2, 2015.

[25] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[26] T. Nishida, Y. Idosaka, Y. Hofuku, S. Kanemune, and Y. Kuno. New methodology of information education with "computer science unplugged". In *Informatics Education-Supporting Computational Thinking*, pages 241–252. Springer, 2008.

[27] S. Ornes. Math dance. *Proceedings of the National Academy of Sciences of the United States of America*, 110(26):10465, 2013.

[28] C. C. Park. Learning style preferences of Asian American (Chinese, Filipino, Korean, and Vietnamese) students in secondary schools. *Equity and Excellence in Education*, 30(2):68–77, 1997.

[29] M. Peacock. Match or mismatch? learning styles and teaching styles in EFL. *International Journal of Applied Linguistics*, 11(1):1–20, 2001.

[30] S. Pollard and R. C. Duvall. Everything I needed to know about teaching I learned in kindergarten: bringing elementary education techniques to undergraduate computer science classes. In *ACM SIGCSE Bulletin*, volume 38, pages 224–228. ACM, 2006.

[31] J. M. Reid. The learning style preferences of ESL students. *TESOL Quarterly*, 21(1):87–111, 1987.

[32] K. Schaffer, E. Stern, and S. Kim. Math dance. *MoveSpeakSpin, Santa Cruz*, 2001.

[33] P. A. Sivilotti and S. M. Pike. The suitability of kinesthetic learning activities for teaching distributed algorithms. *ACM SIGCSE Bulletin*, 39(1):362–366, 2007.

[34] R. Thies and J. Vahrenhold. On plugging unplugged into CS classes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 365–370. ACM, 2013.

[35] A. Vincent and D. Ross. Personalize training: determine learning styles, personality types and multiple intelligences online. *The Learning Organization*, 8(1):36–43, 2001.

[36] D. T. Willingham, E. M. Hughes, and D. G. Dobolyi. The scientific status of learning styles theories. *Teaching of Psychology*, 42(3):266–271, 2015.

[37] V. Zimmerman. Moving poems: Kinesthetic learning in the literature classroom. *Pedagogy*, 2(3):409–412, 2002.