CHAPTER 1

INTRODUCTION

One's object is then to have a clear mental picture of the state of the machine at each moment in the computation. This object can only be achieved with a struggle. (Turing, 1950, p. 459)

Algorithms, the building blocks of computer software, are inherently abstract entities. Indeed, aside from the electronic pulses that flow through the computer as they execute, they lack a concrete representation in the natural world. Their lack of a tangible real-world representation distinguishes them from other objects of scientific inquiry, such as clouds, wildebeests, and amoebas; it also makes them notoriously difficult to teach and learn.

As anyone who has ever had to explain an algorithm to someone else will attest, the use of graphical notations in the explanation of algorithms seems both extremely natural and particularly appropriate. A survey of the pedagogical literature and algorithms research bears this out; ever since the publication of the first volumes of Knuth's seminal series *The Art of Computer Programming* (Knuth, 1973), graphical illustrations of algorithms have become commonplace as visual aids in computer science textbooks and journal articles.

Consider, for instance, Knuth's illustration of a data structure called the *deque* (i.e., the double-ended queue, pronounced "deck"). A generalization of stacks and queues, the deque is a linear list in which all insertions and deletions are done at the ends of the list. Inspired by E.W. Dijkstra's illustrations of stacks, Knuth (1973) depicts the deque as a railroad switching network (see Figure 1). Railroad cars to be added to the deque must pass through a switch; depending on how the switch is set, cars roll either directly to the head of the deque, or around a loop to the tail of the deque. Likewise, depending on how the switches are set, railroad cars may be removed either from the head or tail of the deque.

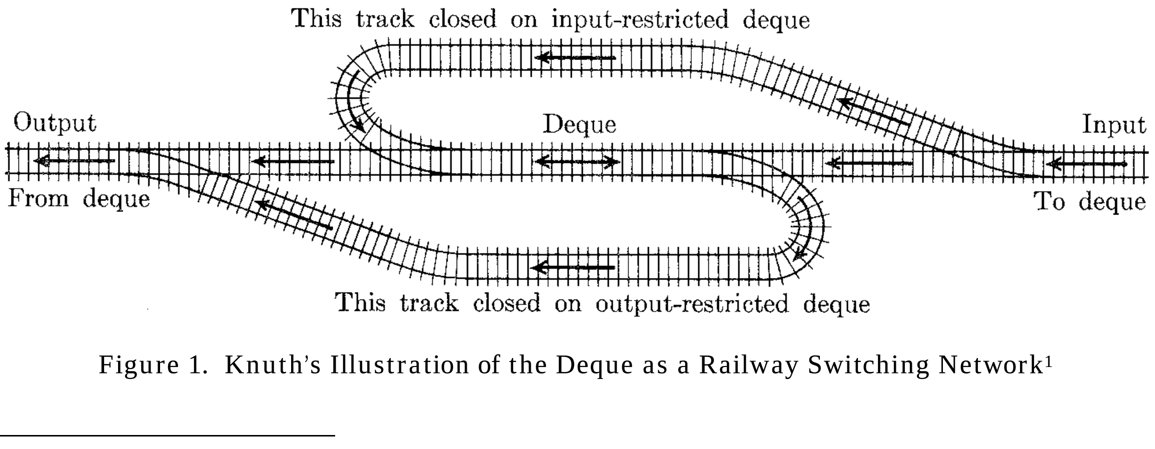
This track closed on input-restricted deque
Output
Deque
Input
From deque
To deque
This track closed on output-restricted deque

Figure 1. Knuth's Illustration of the Deque as a Railway Switching Network[1]

[1]From *The art of computer programming, vol. 1 – 2nd edition* (p. 236), by D. Knuth, Menlo Park, CA: Addison Wesley Longman. ©1973 Addisson Wesley Longman Inc. Reprinted with permission.

The advent of computer graphics technology in the late 1970s and early 1980s brought with it new opportunities to illustrate algorithms. Illustrations moved from paper to computer screen, as computer scientists developed the first computer systems to facilitate the creation of so-called *algorithm animations*—animated illustrations of algorithms in action. Due to the limits of the technology of the time, the first system could do little more than aid in the production of algorithm movies (Baecker, 1975). With the help of that system, Baecker produced *Sorting Out Sorting* (Baecker & Sherman, 1981), a legendary instructional film on sorting algorithms (see Figure 2).

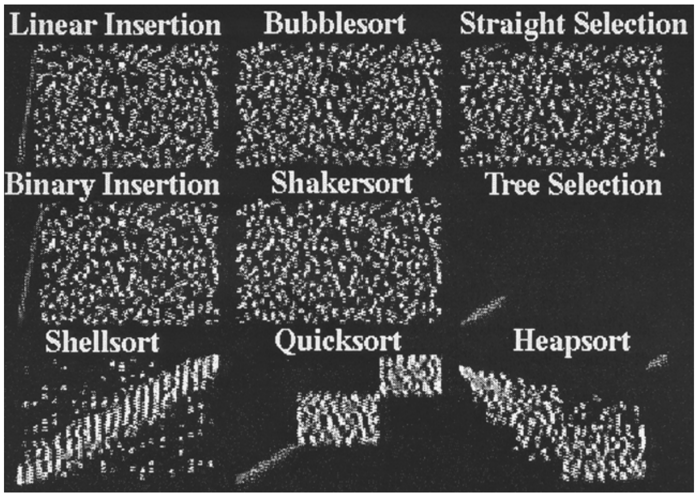

Figure 2. A snapshot of the “Grand Race” in *Sorting Out Sorting*[2]

By taking advantage of emerging graphical workstation technology, later systems supported *interactive* environments for exploring algorithm pictures and animations. For example, the seminal interactive algorithm animation system, BALSA (Brown, 1988), defined an environment in which users could (a) select sets of input data on which to view an algorithm animation; (b) choose an arrangement of alternative views defined for the animation; (c) start and stop the animation, and control the animation's execution speed; (d) zoom and pan an animation view; and (e) write an animation viewing session to a script for later use. Figure 3 presents a snapshot from an interactive session with BALSA.

[2]From *Software visualization* (p. 379), edited by J. Stasko, J. Domingue, M. Brown, & B. Price, Cambridge, MA: The MIT Press. ©1998 The MIT Press. Reprinted with permission. The grand race graphically compares nine alternative sorting algorithms operating on an identical 2500-element data set. Data elements to be sorted are represented as scatterplots of dots—one scatterplot for each sorting algorithm. As each respective sorting technique places elements in place, the corresponding scatterplot gradually transforms into an upward-sloping line.

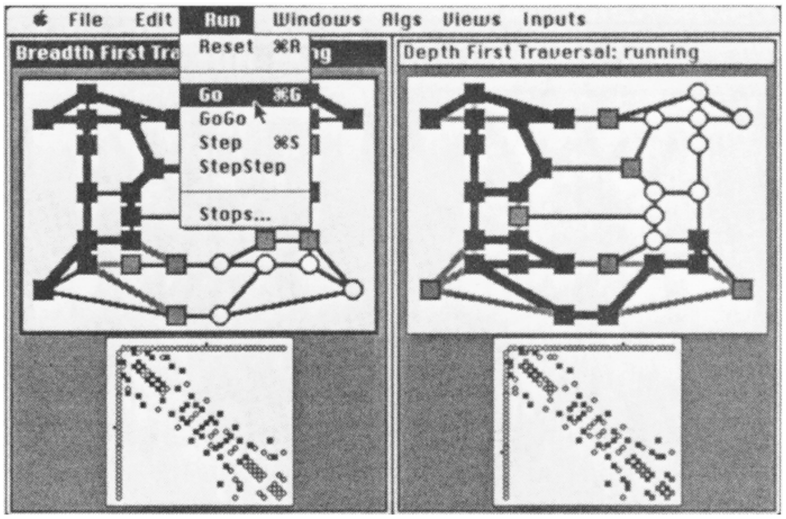

Figure 3. A Snapshot from An Interactive Session with BALSA[3]

To date, the most popular application of such systems aims to address the teaching and learning problem described above. So-called *algorithm visualization (AV) systems* enable instructors and students to create and explore graphical representations (both static illustrations and dynamic movies) of the algorithms under study (see, e.g., Brown, 1988; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989). Noting their increased flexibility as compared with blackboards, whiteboards, and overhead projectors, several computer science instructors have enlisted AV artifacts as visual aids in their lectures (see, e.g., Bazik, Tamassia, Reiss, & van Dam, 1998; Gurka & Citrin, 1996). In addition, some instructors have extolled their value in computer science laboratories (see, e.g., Naps, 1990), as student study aids (see, e.g., Gurka & Citrin, 1996), and as the basis for student assignments (see, e.g., Bazik, Tamassia, Reiss, & van Dam, 1998; Stasko, 1997).

1.1 Problems

Despite the enthusiasm and high expectations of their developers, computer-based AV artifacts have failed to enter mainstream computer science instruction as pedagogical aids (Baecker, 1998; Stasko, 1997). While those few educators who are also AV technology developers tend to employ their own AV technology, the majority of AV educators tend to stick to more traditional pedagogical aids, such as blackboards, whiteboards, and overhead projectors.

In light of the strong intuitive basis for AV artifacts, which is reinforced by the enthusiasm of AV technology developers, a puzzling research question arises:

Research Question: Why has computer-based AV failed to catch on?

[3]From *Algorithm animation* (p. 65), by M. Brown, Cambridge, MA: The MIT Press. ©1988 The MIT Press. Reprinted with permission. In this snapshot, the user is running a side-by-side comparison of breadth-first and depth-first search executing on an identical graph. The user has chosen to view each algorithm in terms of two identical views. The larger, top views depict the graph being traversed; vertices that have been visited are shaded black, vertices whose descendents have not been fully explored are shaded gray, and vertices that have not yet been visited are shaded white. The animations are presently paused; the user is about to restart them by choosing "Go" from the "Run" menu.

While no extant empirical research specifically addresses that question, AV researchers have speculated three main obstacles to AV's widespread adoption:

Obstacle 1: The technology required to deploy AV may not be readily available (Gurka & Citrin, 1996). AV software runs on graphical workstations; college classrooms and laboratories may not be equipped with such workstations, they may not have the right kind of workstations, or they may not have the equipment needed to project an instructor's workstation onto a large screen that an entire class can view.

Obstacle 2: Creating visualizations for classroom use requires substantial time and effort (see, e.g., Brown & Hershberger, 1991; Duisberg, 1987; Helttula, Hyrskykari, & Raiha, 1989; Mukherjea & Stasko, 1994; Stasko, 1989; Stasko, 1991). Mapping an algorithm to an animated representation is a non-trivial problem; it requires careful thought and knowledge of a particular algorithm animation programming framework. Animation programming is especially difficult if the animation must be *general-purpose*—that is, if it must illustrate the algorithm for all of its possible inputs (Stasko, 1989).

Obstacle 3: The pedagogical value of using AV has not been empirically substantiated (Byrne, Catrambone, & Stasko, 1996; Gurka & Citrin, 1996; Lawrence, 1993; Stasko, Badre, & Lewis, 1993). Recently, AV technology developers have become increasingly interested in subjecting their systems to empirical evaluation. Unfortunately, as several researchers have noted, the results of experiments designed to substantiate the pedagogical benefits of AV technology have been far from convincing (Byrne, Catrambone, & Stasko, 1996; Gurka & Citrin, 1996; Hundhausen, 1997; Kehoe & Stasko, 1996; Stasko, 1997). Indeed, of ten controlled experiments that have aimed to substantiate AV's effectiveness as a pedagogical aid (Byrne, Catrambone, & Stasko, 1996, §2 and 3; Kann, Lindeman, & Heller, 1997; Lawrence, 1993, ch. 4 – 9; Stasko, Badre, & Lewis, 1993), only five have yielded statistically significant results in favor of AV technology. Adding to the indecisive nature of these results is the fact that two of the five significant results could not disentangle the benefits of AV from another factor.

Of course, speculating about these obstacles is all in the interest of addressing a second, closely related research question:

Research Question: Can we develop AV artifacts that overcome these obstacles, allowing AV technology to enter mainstream computer science education?

1.2 Paths to Solutions

Over the past decade, computer science educators, technology developers, and even psychologists have sought solutions to the problem of moving AV technology into the mainstream. Depending both on the obstacle they have chosen to address, and on the primary reason to which they have attributed AV's failure, these researchers fall into one or more of three research camps:

1. *Technology Camp*: "We need better AV technology." In response to Obstacle 2, the largest research camp has focused on the development of better AV technology (see, e.g., Baecker, 1975; Brown, 1988; Brown & Hershberger, 1991; Citrin & Gurka, 1996; Dominigue, Price, & Eisenstadt, 1992; Duisberg, 1987b; Gloor, 1992; Helttula, Hyrskykari, & Raiha, 1989; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989). For these

researchers, the concept of "better AV technology" has had a rather narrow definition, which stems from a widely held belief that current AV technology does not facilitate the rapid development of visualizations; instructors must invest substantial time and effort in order to create new visualizations with the technology. If creating new visualizations remains a time- and resource-intensive process, they reason, then AV educators are unlikely to adopt the technology. If, on the other hand, they can succeed in building AV technology that facilitates the rapid construction of new visualizations, then the widespread use of AV technology will follow.

2. *Pedagogy Camp*: "We need better AV-based pedagogy." Responding to Obstacle 3, a second research camp holds that if AV-based pedagogical exercises simply are not effective, then computer science educators are unlikely to adopt AV technology. Accordingly, several computer science educators have explored the development of improved AV-based pedagogy hand-in-hand with the development of improved AV technology (see, e.g., Brown & Sedgewick, 1984; Eisenstadt, Price, & Dominique, 1993; Kehoe & Stasko, 1996; Knox, 1996; Michail, 1996; Naps, 1990; Sigle, 1990; Stasko, 1997). Lawrence (1993; Lawrence, Badre, & Stasko, 1994), in fact, dedicated an entire dissertation to empirically comparing various pedagogical alternatives with and without AV technology. By providing evidence for and against the adoption of particular AV-based pedagogical approaches, these researchers aim not only to make a case for the value of AV technology as a pedagogical aid, but also to provide guidance for its effective use.

3. *Evaluation Camp*: "We need better evaluation methods." The third, and smallest, research camp offers a different response to Obstacle 3. This camp stresses the possibility that current AV technology may, indeed, offer real benefits, but that the methods we have employed to measure those benefits have not been sensitive to them (Gurka, 1996; Gurka & Citrin, 1996). Researchers in this camp hold that past failures to produce positive results may be largely a matter of inappropriate or incomplete evaluation methods. Thus, they argue for the need to reexamine critically the results of past empirical studies, with a particular focus on the ways in which these studies have been designed and carried out. By refining current empirical methods, researchers can, this camp believes, put themselves in a position to stage successful empirical studies; guidelines for proper AV technology design and pedagogy, as well as the increased adoption of AV technology, will follow.

1.3 The Thesis

I argue that the problem of underutilized AV technology has its roots in the *theory of effectiveness* that has guided the three camps just described. That theory, which I call Epistemic Fidelity (EF) Theory (after Roschelle, 1990), encompasses commitments to (a) a view of knowledge as representations in people's heads; to (b) a belief in the graphical medium as an effective encoder of knowledge; and, as a consequence, to (c) a view that AV technology is pedagogically effective because it is able to provide a faithful account (i.e., one with high epistemic fidelity) of an expert's mental model of an algorithm, thus enabling the robust and efficient transfer of that mental model to AV viewers. I hold that EF Theory is deficient; thus, progress along any one of the above solution paths effectively perpetuates the theory's deficiencies, leading to, at best, a temporary fix.

The solution, I contend, is to address the problem not at the surface, but at its roots. In other words, instead of tweaking our current design, pedagogy, and evaluation methods, we need to rethink the theory of effectiveness in which they are rooted. Only by proceeding from an alternative theoretical position—one that sheds new light on why AV technology is pedagogically valuable—do we have any hope of overcoming the obstacles that have stood in the way of AV technology's becoming a viable pedagogical aid.

The alternative theoretical position that I explore in this dissertation, *sociocultural constructivism* (see esp. Lave & Wenger, 1991; Wenger, 1998), differs fundamentally from EF Theory. Rather than viewing knowledge and learning at the level of the individual, as EF Theory does, sociocultural constructivism views knowledge and learning at the level of the *community of practice*. On this alternative view, learning is seen not as acquiring target knowledge structures, but rather as participating more centrally in the practices of the community. As I intend to illustrate, this alternative theoretical position has profound implications for the design, evaluation, and pedagogical use of AV technology. I argue that, by taking the position seriously, we have hope of overcoming the obstacles that have stood in the way of AV technology's adoption.

1.4 Brief Summary of the Dissertation

Clearly, one cannot hope to furnish a convincing empirical case for an alternative theory within the scope of a single dissertation. Accordingly, the more modest objective of this dissertation is to lay a preliminary foundation for the thesis, on which future research can build. The preliminary foundation I am striving for should both establish the plausibility of the alternative theoretical position, and generate excitement about and interest in its further development. I aim to establish the plausibility of the position by furnishing and analyzing a corpus of empirical data; I aim to generate excitement and interest by presenting a novel prototype AV system that differs radically from the systems that have been built so far.

Figure 4 presents a graphical summary of the dissertation. The labels (a) through (f) each map to a major component of the dissertation. In the subsections that follow, I present an executive summary of the dissertation by elaborating on these components in turn.

1.4.1 EF Theory, Its Stronghold on Past Research, and Its Inadequacy

My first objective is to describe EF Theory [abbreviated "EF" in Figure 4(a)] and to demonstrate its stronghold on past AV research through a review of the literature. By performing an informal meta-analysis of the experimental research whose results are summarized in my discussion of Obstacle 3 above, I make the case that EF Theory is inappropriate as a guiding theory of effectiveness—hence the slash through the EF Theory oval in Figure 4(a).

1.4.2 Choosing an Alternative Theory

My second objective [see Figure 4(b)] is to choose a more promising alternative theoretical foundation from which to proceed. In addition to arguing against the validity of EF Theory, my meta-analysis of past experimental research reveals a noteworthy trend: namely, that the more *actively* learners were involved with AV technology, the better they performed. Students who were actively involved did such things as (a) develop their own input data sets and observe the execution of the AV on those data sets; (b) program the algorithm while observing the AV; and (c) make explicit predictions regarding future states of the AV before seeing them. This finding, coupled with the observation that the target skills of an algorithms course are inherently social, argues for a shift to a theoretical position called *sociocultural constructivism* (see, e.g., Lave & Wenger, 1991). Sociocultural constructivism [abbreviated "SC" in Figure 4(c)] views knowledge not at the level of the individual, but at the level of the community of practice. On this alternative view, learning is seen in terms of participating more centrally in the practices of the community; AV technology is regarded as pedagogically valuable insofar as it facilitates access to increasingly "expert" community practices, and insofar as it mediates meaningful conversations among community members.

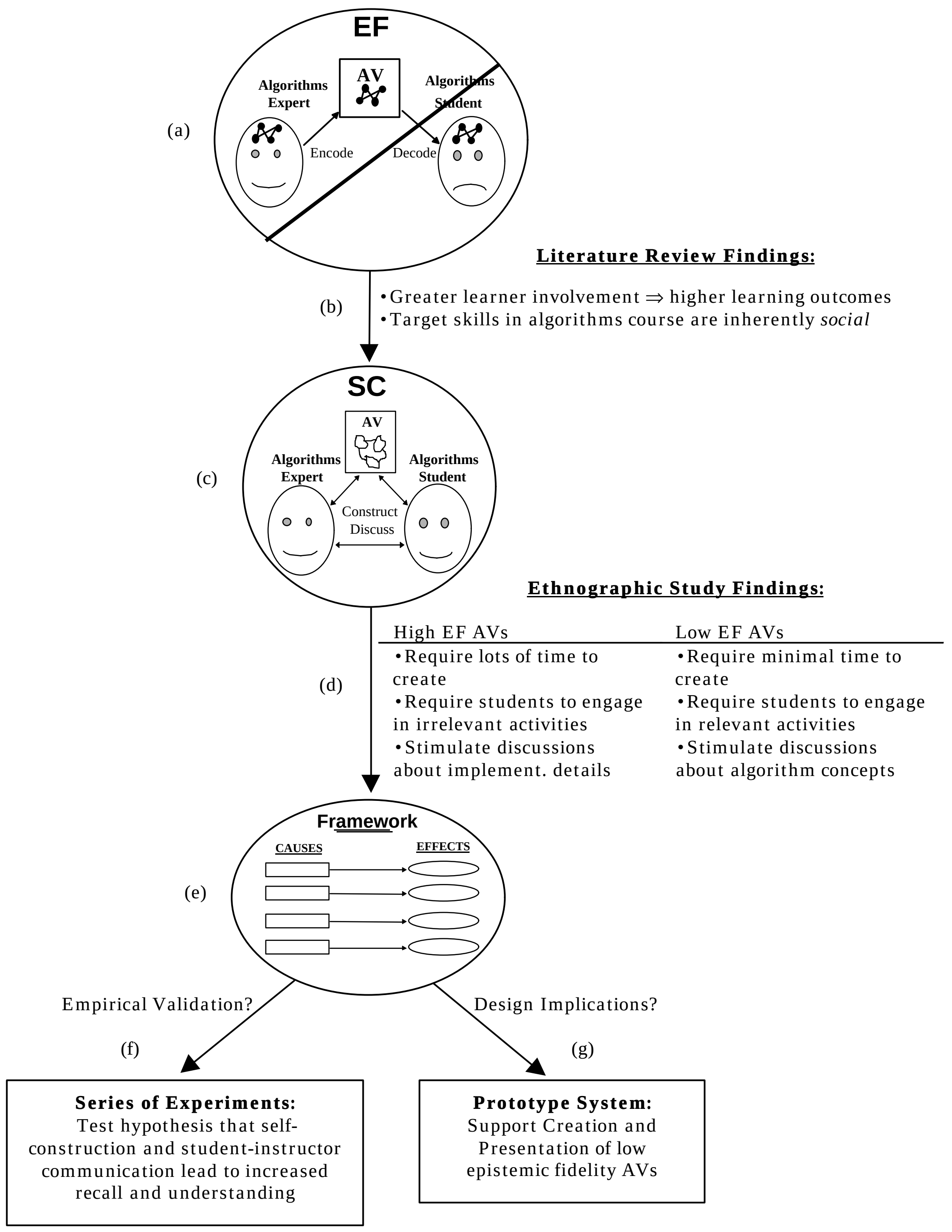
EF
(a)
Algorithms Expert
AV
Algorithms Student
Encode
Decode
Literature Review Findings:
(b)
•Greater learner involvement ⇒ higher learning outcomes
•Target skills in algorithms course are inherently *social*
SC
(c)
AV
Algorithms Expert
Algorithms Student
Construct
Discuss
Ethnographic Study Findings:
(d)
High EF AVs
•Require lots of time to create
•Require students to engage in irrelevant activities
•Stimulate discussions about implement. details
Low EF AVs
•Require minimal time to create
•Require students to engage in relevant activities
•Stimulate discussions about algorithm concepts
Framework
(e)
CAUSES
EFFECTS
Empirical Validation?
Design Implications?
(f)
(g)
Series of Experiments:
Test hypothesis that self-construction and student-instructor communication lead to increased recall and understanding
Prototype System:
Support Creation and Presentation of low epistemic fidelity AVs

Figure 4. Graphical Summary of the Dissertation

1.4.3 Empirically Exploring the Sociocultural Constructivist Approach

My third objective is to adapt the sociocultural constructivist position so that it accounts for the pedagogical role of AV technology within the community of practice being reproduced through an undergraduate algorithms course. To do so, I present a pair of ethnographic studies of an undergraduate algorithms course in which AV software was used as a means of facilitating students' more central participation in the community. Specifically, students were asked to use both AV software, and simple art supplies, to construct and present their own visualizations—two activities commonly performed only by community experts (algorithms instructors).

The two key findings of these studies [see Figure 4(d)] not only support the sociocultural constructivist position, but also offer valuable practical recommendations for how best to implement AV construction assignments in an undergraduate algorithms course. First, the studies found that assignments in which students were required to use conventional AV software to construct *high epistemic fidelity* AVs actually distracted them from the focus of the course. Here, the term "high epistemic fidelity" denotes AVs that (a) are capable of depicting the target algorithm for general input, and (b) resemble the polished figures found in algorithms textbooks. Constructing high epistemic fidelity AVs was distracting not only because it took large amounts of time (over 30 hours on average, according to data I collected), but also because it required students to engage in low level implementation activities—plainly not the focus of a course that stresses the conceptual foundations of algorithms. On the other hand, when students used simple art supplies to construct low epistemic fidelity AVs, they spent far less time overall, and the time that they did spend was devoted to more relevant activities. Thus, low epistemic fidelity technology enabled students to participate more extensively in community-relevant practices—an important condition for learning, on the sociocultural constructivist view.

The second key finding of the studies was that, when students presented their AVs and discussed them with their instructor and peers, high epistemic fidelity AVs tended to generate conversations about implementation details, whereas low epistemic fidelity AVs tended to generate more relevant conversations about algorithm concepts. It appears that high epistemic fidelity AVs were so difficult to create that students preferred to discuss the toil they put into implementing them. On the other hand, the rough, unfinished look of low epistemic fidelity AVs seemed to invite conversations about the concepts they represented. Once again, this finding resonates with the sociocultural constructivist view, which regards the presentation and discussion of algorithms as a key form of participation in an algorithms course, and which sees AV technology's value in terms of its ability to facilitate such participation.

1.4.4 A Framework of Cause and Effect

My fourth objective is to use these findings as a basis for developing a set of specific hypotheses regarding the effectiveness of AV technology in an undergraduate algorithms course. The result is a framework of cause and effect [see Figure 4(e)]. In my ethnographic studies, I identified six causal factors as strongly influencing the effectiveness of AV technology, and four distinct measures emerged as appropriate gauges of the effectiveness of these factors. In addition to elaborating on these factors and measures, the framework proposes four specific hypotheses, each of which links one or more causal factors to a specific dependent measure:

1. The Activity Relevance Hypothesis: Low input generality, low typeset fidelity, and direct graphics cause high activity relevance.

2. The Communication Effectiveness Hypothesis: Low epistemic fidelity causes high communication effectiveness.

3. The Understanding and Recall Hypothesis: Self-constructing AVs with a story line, and then presenting them to an instructor for feedback and discussion, causes high recall and understanding.

4. The Community-Building Hypothesis: Self-construction and high instructor communication cause high community-building.

The framework constitutes the beginnings of an alternative theoretical position that the remainder of the dissertation begins to explore and flesh out, and that future research will need to develop and refine further. Resonant with constructivist learning theory, the position stresses (a) the value of students' constructing and refining their own personal representations of the material they are learning; and (b) the importance of students' participating more centrally in algorithms practice by presenting their self-constructed AVs to their peers and instructor for feedback and discussion. Moreover, in radical departure from extant AV technology's obsession with high epistemic fidelity AVs, the position underscores the value of low epistemic fidelity AVs in focusing students' on algorithms and how they work, and in promoting effective communication about algorithms.

1.4.5 Two Research Directions

Aside from refocusing the ongoing dialog on AV effectiveness, the hypotheses posited by the framework raise numerous research questions that are grist for further investigation. The final objective of the dissertation is to present original research that addresses two of the most important of these questions. The first research question [see Figure 4(f)] concerns the viability of the hypotheses: Can they be empirically validated? To begin to answer this question, I present an experiment that puts to the test a key piece of the Understanding and Recall Hypothesis (see above), whose confirmation should prove to be of great interest to computer science educators:

Hypothesis: On a test of procedural understanding and recall, students who construct their own AVs will outperform students who view and interact with an AV constructed by an expert.

Notice that, while EF Theory would not predict this hypothesis, the hypothesis follows directly from constructivist theory.

To test this hypothesis, the experiment employed a between-subjects design with two treatment groups:

Self-Construction: Students use simple art supplies to construct their own homemade visualization of an algorithm. They are responsible for using their visualization to step through the execution of the algorithm for at least five input data sets.

Active Viewing: Students interact with a visualization of predefined visualization of the same algorithm constructed by an expert. They are responsible for watching the visualization execute on at least five input data sets of their choosing.

For this experiment, I recruited 24 students out of a later offering of the same undergraduate algorithms course in which I conducted ethnographic fieldwork. Participants were randomly assigned to one of the two treatments such that the two treatment groups were optimally matched with respect to self-reported grade point average. Participants had two and a half hours either to construct their own visualization (Self Construction group), or to interact with a predefined visualization (Active Viewing group), of QuickSelect, a divide-and-conquer algorithm that had been covered in a course lecture prior to the study (see, e.g., Cormen, Leiserson, & Rivest, 1990, pp. 153–155, 185, 187–188). Upon completing the learning phase of the study, participants completed two tasks designed to assess their procedural understanding and their recall ability:

1. a tracing task in which they had to trace through the algorithm for a novel set of input data; and

2. a programming task in which they filled in a partially-completed C++ or Java implementation of the QuickSelect algorithm.

Participants had 25 minutes to complete each of four tracing tasks, and 35 minutes to complete the programming task. I scored participants' traces and programs by comparing them against perfect traces and programs.

Although the Self Construction group outperformed the Active Viewing group on both tasks, statistical analyses failed to yield any significant differences between the two groups. A thorough review of the experimental materials and procedure did not reveal any flaws in the design; participants appear to have had an excellent chance of doing well on all tasks. This is borne out in the data, which, although extremely variable, include seven scores above 80% on both the tracing and programming tasks. In the absence of experimental design flaws, the most plausible reason that no significant differences were detected is that the self-construction factor, as manipulated in this experiment, is simply not reliable enough to produce an effect.

On a practical level, however, the results of this experiment can be placed in a positive light. Using conventional (high epistemic fidelity) AV technology to prepare visualizations for classroom use is notoriously difficult (see, e.g., Brown & Sedgewick, 1985). In contrast, having students construct their own low epistemic fidelity visualizations using art supplies is cheap, easy, and requires a relatively small investment of time on the part of instructors. Thus, instructors looking for a low-overhead way to incorporate AV technology into their curricula can take comfort in the finding that students who use "low tech" art supplies to construct their own visualizations may learn algorithms just as well as students who interact with AVs developed by their instructor with conventional "high tech" AV technology.

The second research question explored by this dissertation addresses the implications of the framework's hypotheses for technology design [see Figure 4(g)]: If one takes the hypotheses as constraints on the design space of effective AV technology, what kind of AV system emerges? I pursue this question by designing and prototyping (a) SALSA (<u>S</u>patial <u>A</u>lgorithmic <u>L</u>anguage for <u>S</u>torybo<u>A</u>rding), a high-level, interpreted language for programming low epistemic fidelity AVs; and (b) ALVIS (the <u>AL</u>gorithm <u>VI</u>sualization <u>S</u>toryboarder), an interactive graphical user interface for programming in SALSA. In line with the framework's hypotheses, SALSA and ALVIS aim to meet two key functional requirements:

1. Support the quick and easy construction of *low epistemic fidelity* AVs; and

2. Support the interactive presentation and discussion of those AVs.

Notice that these requirements differ markedly from those that follow from EF Theory, which has prompted extant AV systems to support the creation, but not the presentation, of *high epistemic fidelity* AVs.

To meet the first of the above requirements, SALSA and ALVIS aim to make constructing a low epistemic fidelity animation (i.e., a "storyboard") as easy as, and ideally easier than, constructing a homemade animation out of simple art supplies. To do so, ALVIS and SALSA are firmly rooted in the physical metaphor of art supply storyboard construction. Indeed, in past empirical studies (Chaabouni, 1996; Douglas, Hundhausen, & McKeown, 1995, 1996), and in my ethnographic fieldwork, creating homemade visualizations out of simple art supplies proved incredibly natural, quick, and easy.

An important component of this metaphor is the concept of *cutouts* (or *patches*; see van de Kant, Wilson, Bekker, Johnson, & Johnson, 1998): pieces of construction paper that may be cut out and drawn on, just like real construction paper. SALSA/ALVIS users create homemade animations by cutting out, sketching on, and arranging cutouts on a static background, and then specifying the ways in which the cutouts should be animated over time.

The ALVIS interactive environment includes both a "SALSA script" window, which contains the SALSA code that drives a homemade animation, and a "Storyboard" window in which the animation actually unfolds. Users can create, place, and animate cutouts either by typing SALSA commands directly into the "SALSA script" window, or by directly manipulating elements in the "Storyboard" window (which, in turn, automatically generates and inserts SALSA commands into the "SALSA script" window).

Observe that specifying an AV with ALVIS and SALSA differs markedly from specifying an AV with conventional AV technology. Whereas conventional AV technology (see Roman & Cox, 1993 for an overview) requires one to specify explicit mappings between an underlying driver program and an animation, ALVIS and SALSA enable one to specify animations that drive themselves; the notion of an external driver program is jettisoned altogether. To support self-driving animations, SALSA enables the logic of an animation to be specified in terms of the animation's *spatiality*—that is, in terms of the *spatial relations* (e.g., *above*, *left of*, *between*) among cutouts in the animation.

The following brief example illustrates the kinds of spatial logic that SALSA supports. Recall that the idea behind the *insertion sort* algorithm is to successively insert elements in the unsorted portion of the array into the sorted portion of the array. The pseudocode presented in Figure 5 describes the algorithm.

```
 1:  INSERTION-SORT(A)
 2:  for xindex ← 2 to n - 1 do
 3:    x ← A[xindex];
 4:    j ← xindex - 1;
 5:    while (j > 0) and (A[j] > x) do
 6:      A[j+1] ← A[j];
 7:      j ← j – 1;
 8:    end while;
 9:    A[j+1] ← x;
11:  end for;
12:  end INSERTION-SORT;
```

Figure 5. Pseudocode Description of Insertion Sort Algorithm

One possible animation of the algorithm represents sorting elements as blocks, and uses a special arrow to mark the dividing line between the sorted and unsorted portions of the list (Figure 6). Each time the outer loop in the insertion sort is executed (lines 2 to 11), the arrow moves one position to the right. Notice that the internal logic governing the insertion sort *algorithm* is distinct from, but parallel to, the internal logic governing the insertion sort *animation*. For instance, consider the way in which each decides when the array is sorted. In the insertion sort algorithm, we continue sorting until the outer loop index (*xindex*) reaches $n – 1$—that is, until it is one less than the size of the array *a*. By contrast, in the insertion sort animation, we continue with the animation until the arrow slides to the right of the row of blocks. Whereas in the algorithm, the internal logic is mathematical ($xindex = n – 1$), in the animation, the internal logic is spatial (*arrow is right of edge of row of blocks*). This is precisely the kind of spatial logic, which might be thought of as analogy between the algorithm and the animation, that SALSA is capable of expressing.

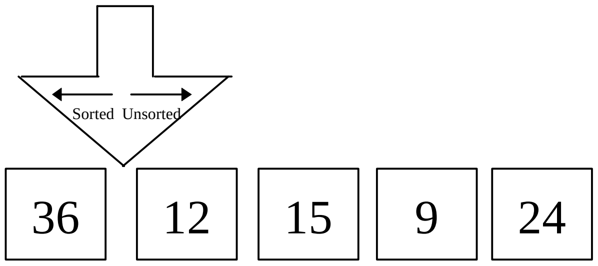

Figure 6. An AV of the Insertion Sort Algorithm

Recall that the second requirement of SALSA and ALVIS is that they must support the interactive presentation and discussion of low epistemic fidelity AVs. To meet this requirement, ALVIS supports the following two interface features:

1. *Fine-grained execution control*. Users may execute or single-step through an animation both forwards and backwards. At any point, users may pause, change the execution speed, or reverse the direction of execution.

2. *Interactive mark-up and modification*. Users may annotate the animation as it is executing with a "marking pen" tool; annotations may be erased at any point with an "eraser" tool. In addition, users may pause the animation and modify (i.e., reprogram) it at any point in its execution—without having to restart the animation from the beginning.

As was the case for animation specification, it is important to underscore that ALVIS and SALSA support a range of functionality for animation presentation and discussion that differs markedly from that of extant AV technology. To appreciate the contrast, consider the user interface that POLKA (Stasko & Kraemer, 1993) provides for controlling the execution of animations (see Figure 7); this interface is typical of extant AV systems. Much like the interface of a tape recorder, it allows users to start and pause the animation at any point, and to adjust the execution speed. Moreover, in POLKA, along with most extant AV systems, making changes to an animation requires one to quit the animation, modify code (which is typically low-level), recompile the animation, and restart the animation. Clearly, within the scope of an interactive presentation, performing this list of activities would be at best distracting, and at worst downright impossible.

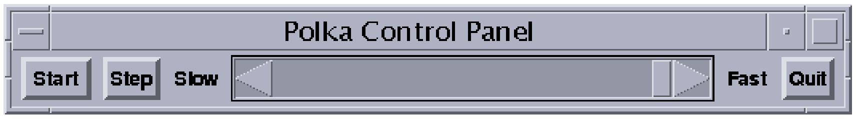

Figure 7. The POLKA User Interface for Controlling Animation Execution

1.5 Organization of the Dissertation

The dissertation summarized in the previous section is presented in the seven chapters that follow. Chapter 2 introduces EF Theory, reviews its influence on past research, and performs a meta-analysis of the experimental research designed to support it. That meta-analysis motivates the shift to Constructivist learning theory. Chapter 3 presents the cognitive and the sociocultural versions of Constructivism, and argues that, because the target skills of an algorithms course are inherently social, the sociocultural version is more appropriate as a guiding theory. Chapter 4 presents two ethnographic studies that explore the potential of the sociocultural constructivist pedagogical approach. Based on observations made in the ethnographic studies, Chapter 5 develops a framework of cause and effect—a specific set of hypotheses regarding the effectiveness of AV technology within an undergraduate algorithms course. Chapter 6 proposes a series of experiments for testing one of those hypotheses, and presents the results of the first experiment in the series.   Chapter 7 describes SALSA and ALVIS, a prototype language and system that explore the design implications of the hypotheses. Chapter 8 summarizes the work and its contributions, and outlines areas for future research.