Lumberjack Summer Camp: A Cross-Institutional Undergraduate Research Experience in Computer Science*

Patricia Johann
Department of Mathematics and Computer Science, Dickinson College
Carlisle, PA 17013 USA
johannp@dickinson.edu

Franklyn A. Turbak
Department of Computer Science, Wellesley College
Wellesley, MA 02481 USA
fturbak@wellesley.edu

Abstract

This paper describes our experiences in leading Lumberjack Summer Camp, a ten-week undergraduate research experience in compiler-based optimization techniques for functional programs, held during the summer of 2000. Like many undergraduate research experiences, Lumberjack Summer Camp was designed to provide an opportunity for students and faculty to work closely together toward a common research goal. But Lumberjack Summer Camp was designed around an additional aim as well: to bring together a critical mass of researchers from two small liberal arts colleges to pursue individual research projects situated within one overarching, collaborative, cutting-edge research endeavor. We explore some important consequences of this design choice, ultimately offering Lumberjack Summer Camp as an unusual, but very workable, model for undergraduate research experiences.

*To appear in *Computer Science Education*, Volume 11, Number 4 (December 2001).

1 Introduction

The educational value of undergraduate research in the sciences cannot be overestimated. In addition to providing an opportunity for students to develop close, academically-based mentoring relationships with faculty, involvement in scientific research develops students' capacity for intellectual discipline and independent thought, fosters the ability to synthesize knowledge from seemingly disparate fields of inquiry, and teaches the importance of critically analyzing one's work relative to established theory and scientific evidence. It also helps students understand the role that research plays in their own educations — both directly, by determining the content of the curricula they study, and indirectly, by creating new knowledge and keeping faculty active in, and excited about, their disciplines. Perhaps most importantly, involvement in research enables undergraduates to make the transition from course-directed consumption of scientific knowledge to participation in the production of knowledge. Indeed, *doing* science, rather than learning science, allows students to experience — exhilaratingly, and often for the first time — their chosen discipline as a living, breathing, growing body of knowledge in whose evolution they can meaningfully participate, rather than as a finished product long since developed by a small contingent of dead geniuses.

Undergraduate research in computer science offers these benefits to students and faculty no less than does research in other sciences. Involvement in undergraduate research in computer science has changed the course of student and faculty careers alike, spurring the former on to further pursuit of the discipline, and adding new dimensions to the professional lives of the latter. But while other sciences — most notably biology, chemistry, geology, and psychology — and engineering have rich traditions of undergraduate involvement in research, computer science does not. Moreover, surprisingly little has been written about those efforts that *have* been made to involve undergraduates in computer science research. (Recent articles in this area include (Lopez & Messa, 1994, Sturm & Glassman, 1996, Bard, Berque, & Dershem, 1996, Koelzer, 1997, Tesser, Al-Haddad, McClaugherty, & Frame, 1998, Dima, Parent, Briggs, & Dickerson, 1998, Passos, 1999)).

In this paper, we report on our experiences in leading Lumberjack Summer Camp (LSC), a ten-week intensive research experience for six undergraduates from Bates and Wellesley Colleges, held at Wellesley College during the summer of 2000. Participants in LSC investigated the practical effectiveness of a particular class of compiler-based optimization techniques for functional programs known as *fusion* algorithms (Launchbury & Sheard, 1995). (Since fusion algorithms achieve their effects by eliminating certain tree-like data structures from programs, fusion is sometimes punningly called *deforestation* (Wadler, 1990). As persons committed to removing trees from programs, we press this pun still further, calling our summer research experience "Lumberjack Summer Camp," and referring to ourselves as "lumberjacks.") Like many undergraduate research experiences at small liberal arts colleges, LSC was designed to provide an opportunity for students and faculty to work closely together toward a common research goal. However, several features of LSC distinguish it from more typical such experiences:

- LSC was a *cross-institutional initiative*. Faculty and students from two small liberal arts colleges (Bates and Wellesley) pooled resources at the host institution (Wellesley) for the ten week duration of the research experience. The faculty leading LSC worked collectively, and without regard to institutional affiliation, to guide the research of each of the student participants.

- LSC involved a relatively *large group of students and faculty*, with a *critical mass of researchers coming from each institution*. Although some undergraduate research experiences involve six or more students, those at small liberal arts colleges typically involve only one or two students under the guidance of a single faculty member. Even when larger numbers of students are involved in a research experience — as might be the case in an NSF-funded Research Experience for Undergraduates, for example — those students are usually drawn one each from a variety of home institutions. While this can lend some diversity to the research experience, it also makes it more difficult to preserve the sense of intellectual community the experience is intended to stimulate once the participants have all dispersed to their home institutions.

- LSC fostered *both individual and collaborative research*. The research experience we provided was centered around a single, collaborative research project, a critical piece of which each student worked on individually (with faculty guidance). An important feature of LSC was that the students' individual research was always done in the context of this larger project. With a few notable exceptions (Koezler, 1997, Dima, et al., 1998, Passos, 1999), undergraduate research experiences — especially those at small liberal arts schools — are often unable to provide this kind of dual focus for scientific work. Instead, they tend to be structured around individual faculty-student research (e.g., (Lopez & Messa, 1994, Sturm & Glassman, 1996)). This may be because a research project scoped to be carried out successfully with the available resources cannot offer multiple entry points for undergraduates. It may also be that there simply aren't other student researchers on the project with whom to collaborate.

This paper explores some important consequences of these features of LSC. Based on our experiences, we offer LSC as a somewhat unusual, but very workable, model for undergraduate research experiences.

The remainder of this paper is structured as follows. In Section 2 we discuss our motivation for designing Lumberjack Summer Camp. Section 3 describes our goals for Lumberjack Summer Camp and some design decisions we made in planning it. Section 4 briefly describes the research project in which the participants in LSC were engaged, and Section 5 details the way in which the summer research experience was structured around that project to achieve our pedagogical and research goals. In Section 6 we reflect on some important lessons learned in leading LSC. We consider LSC a useful model for improving the research infrastructure at small colleges, and conclude in Section 7 with some LSC-inspired recommendations for colleagues considering undertaking similar undergraduate research initiatives at other such institutions.

2 Why Lumberjack Summer Camp?

LSC was conceived as an informal pilot study to determine the feasibility of cross-institutional undergraduate group research in computer science. It was led by two computer science faculty, one from each of Bates and Wellesley Colleges, and provided research opportunities for two Bates and four Wellesley computer science students: three rising juniors, two rising seniors, and one new graduate heading for graduate school. The research experience was funded through the NSF research grants of the individual faculty members involved, together with a grant from the Howard Hughes Medical Institute, administered by Bates College, and a Wellesley College Science Center summer research grant.

The organization of LSC grew out of three main considerations. First, we believe that providing experience with cutting-edge research is one of the most effective tools educators can employ to attract talented undergraduates to, and retain them in, scientific careers. In particular, we consider undergraduate involvement in research to be a fundamental aspect of science education at small liberal arts colleges. We further believe that these institutions are uniquely poised to offer opportunities for undergraduates to work side-by-side with faculty in research settings. Indeed, undergraduates at small liberal arts colleges do not compete for faculty attention with graduate students and postdoctoral research fellows. Instead, faculty — rather than graduate students or postdoctoral associates — take on the role of research mentor, providing undergraduate researchers the same close interaction and individualized guidance in research that they offer in other settings throughout the academic year.

Second, we believe that the intellectual stimulation derived from interacting with others pursuing related ideas is critical to the success of a research endeavor — for faculty as well as for students.

Good ideas rarely emerge from the mind of a single individual working without the benefit of outside influences. Accordingly, mitigating some of the effects of disciplinary isolation experienced by students and faculty based at small liberal arts colleges was one of our key goals in designing this research experience. Such institutions often have relatively few students undertaking advanced study in any particular discipline at any given time, and also tend to recruit new faculty with an eye toward broadening disciplinary coverage rather than deepening local research expertise in any particular area. Our aim was to bring together in one physical location a critical mass of scholars all focused on a common intellectual goal. In this way we hoped to maximize opportunity for the "accidental collisions of ideas" which Bruce Alberts, President of the National Academy of Sciences, maintains are essential to intellectual productivity (Boyer). We also hoped to instill among the participants of LSC a sense of intellectual community that would persist well beyond the ten week duration of the research experience.

Third, we were already pursuing independent, but closely related, research in compiler-based optimization of functional programs. The discovery that the technical goals of these projects nicely complemented one another, that we therefore stood to benefit from intensive and prolonged professional contact with one another, and that we share a commitment to involving undergraduates in our research, led us to consider the following questions: Could our two research projects be dovetailed to provide a single research experience suitable for undergraduate involvement? Could such a research experience be structured in such a way as to enable us to simultaneously achieve our research and pedagogical goals?

LSC emerged from our efforts to answer these questions. Through it, we sought to weave together the faculty involvement with undergraduates which is the hallmark of small liberal arts colleges and the benefits of disciplinary concentration found at larger, often more research-oriented, institutions.

3 Planning Lumberjack Summer Camp

Once we had decided to lead LSC, our first orders of business were to secure funding for student stipends, and to hire students into the research positions these stipends made possible. Although we sought to hire capable, interested, intellectually mature students with appropriate technical backgrounds, good work habits, and good interpersonal skills, our primary criterion for selecting LSC participants was that they exhibit a keen interest in the project. Lack of specific technical knowledge — beyond a fundamental familiarity with computer science as a discipline and a basic facility with functional programming — barred no one from participating. Of course, due to the nature of the research to be conducted, some knowledge of programming languages and compilers was a definite advantage for those that had it.

When approaching students about participating in the summer research we were careful to emphasize that conducting research is very different from working problems in a textbook or carrying out a pre-planned laboratory experiment. By contrast with those situations, in research there is not some already-understood answer or phenomenon that one is trying to verify or reproduce. Instead, research is concerned with the discovery of new knowledge — ideally, new to faculty and student researcher alike. Because of this, we felt it unrealistic for our students to expect us always to have ready answers to their questions or to be able to guide them unequivocally. We therefore explained to potential participants in LSC that they had to be prepared for the frustrating-but-exciting experience of understanding their individual research project as well as — if not better than — their faculty mentors and collaborators.

We were also careful to make explicit to potential recruits our main pedagogical goals. In addition to providing an opportunity for them to participate in the discovery of knowledge, we wanted them to emerge from the research experience with:

- the ability to acquire the background necessary to make new discoveries,

- the ability to communicate — in both written and oral form — new discoveries and their significance to specialist and non-specialist audiences alike,
- the ability to understand and explain to others the contents of research papers, and
- facility with project-specific tools, such as the HUGS Haskell interpreter and GHC Haskell compiler, as well as general purpose tools (e.g., Linux, Emacs, LaTeX) whose use has become standard in computer science research.

In addition, we wanted our students to learn the perseverance required to move a research project from conception to completion. We were, however, confident that circumstance would provide this lesson without any special effort on our part.

These goals explicitly guided our planning of LSC. To increase the likelihood of achieving them, we devoted considerable attention to two fundamental issues, namely identifying suitable collaborative and individual research projects, and structuring the research experience.

Regarding the former, we specifically sought a collaborative project of sufficient technical merit to be of interest to the functional programming community, and of sufficient breadth to accommodate a group research effort. At the same time, we required the collaborative project to encompass a number of substantive self-contained subprojects, each suitable for investigation by a single student or pair of students. At the levels of both the collaborative and individual projects, we sought a research project that could encompass the paradox of "open-ended closed-endedness." That is, we sought a project which was closed-ended in the sense of allowing us to state clearly what our research aims were and understanding, in outline, how we might achieve them, but was simultaneously open-ended in the sense that we would still have much to discover, many technical problems to address, and more than one surprise to encounter on the path to filling in that outline.

The collaborative research we undertook is described in Section 4. The individual projects which the students ultimately decided on, as well as the relationships of these projects to the collaborative project, are also discussed there. The projects can be classified into four broad categories:

1. *Comparing different approaches to solving a problem and evaluating which approaches work best in which contexts.* The comparison process often highlights aspects of the compared approaches that would not stand out if the approaches were studied in isolation. In some cases, comparative studies naturally lead to the development of hybrid approaches that combine the best aspects of the systems studied.

2. *Testing and/or analyzing a program developed by another researcher to discover its strengths and weaknesses.* Being able to articulate weaknesses is critical to the development of increasingly robust programs. Ideally, the source code of the program should be available for study, as should papers or technical reports describing the theory behind the program and its implementation. But even if the source code is not available (as was the case in one of our individual projects), "black-box" testing can often reveal valuable information about the program.

3. *Implementing (or extending toy implementations of) theoretical results developed by another researcher.* It is common for researchers, especially Ph.D. and Masters students, to develop theory with no associated implementation, or with only toy prototypes. The development of "real" implementations is valuable, especially if they involve extensions not considered by the original developer. Furthermore, in the course of an implementation, it is common to encounter questions and problems that lead to new lines of research.

4. *Developing tools that enable other members of a collaborative project to work more effectively.* In group projects, it is often the case that some members could be more productive if they had certain tools, but they do not have the time to develop the tools themselves. In these cases, one or more group members can support other members by developing tools. Sometimes such tools are interesting enough in their own right to be useful beyond the scope of the project.

These kinds of projects have three important properties that make them particularly well-suited for research experiences like LSC. First, they are accessible to undergraduates, even in traditionally "high barrier" subdisciplines of computer science like programming language design and implementation. Second, they push forward the frontiers of computer science knowledge, and so constitute "real" research. This stands in contrast to many undergraduate projects, in which students often work on a problem whose solution is well-known by researchers (but not by them). While such projects are still valuable for the student, we strongly favor projects which lead to results which are new to everyone. Finally, projects like those described above are especially amenable to collaborative enterprises in which each student is working on part of a larger project. In particular, a project focusing on comparing different techniques naturally has many subprojects that involve testing, analysis, and implementations of the techniques. Comparison projects also create an environment that encourages students to share the knowledge and experience gained in individual subprojects with the larger group.

With regard to the structure of LSC we sought to accommodate a second paradox as well, namely that of "structured flexibility." This entailed providing enough structure for the research experience that the students didn't flounder, but enough flexibility that they achieved a reasonable degree of intellectual independence. Toward this end, we encouraged our students not to consider us the final arbiters of all things Lumberjack, but rather to learn to see themselves and one another as the first-class decision-makers and resources for one another that they eventually became. The way in which we structured LSC is discussed in more detail in Section 5.

4 The Research Problem

In this section we briefly describe the collaborative research project around which LSC was organized. A detailed explanation of the research problem is beyond the scope of this paper, but we want to give the flavor of the research we undertook, and also make clear how each student's individual research contributed to the collaborative project.

4.1 The Collaborative Project: An Overview

Modular program construction is widely regarded as an integral part of any reasonable software development process. One very general way of achieving modularity in functional languages is to construct large programs as compositions of smaller components. Each component in such a composition generates a data structure representing its output, and this data structure is consumed as input by the next component in the composition. These *intermediate data structures* emerge as an essential kind of "glue" that enables reusable component programs to be connected together in mix-and-match ways (Hughes, 1990).

Unfortunately, modular programs constructed in this fashion tend to consume more time and space resources than their non-modular counterparts. The main difficulty is that the direct implementation of compositional programs *literally* constructs, traverses, and discards its intermediate data structures — even when they play no computational role. When the sole purpose of an intermediate data structure is to specify the connection between the computations performed by two components, the computations can be woven together into a single component. The resulting non-modular programs may exhibit even order-of-magnitude efficiency increases over corresponding modular ones.

Fusion is a technique for automatically transforming modular programs into more efficient, non-modular equivalents via the elimination of intermediate data structures. Although the theoretical foundations of fusion are fairly well understood and several techniques have been developed, surprisingly little is known about their effectiveness in practice. An understanding of the issues which arise in practical applications of fusion is, however, necessary in order to determine whether or not it gives rise to sufficient program improvement to be incorporated into production-quality optimizing compilers for functional languages.

Toward this end, our summer research experience centered around evaluating and comparing the effectiveness of implementations of the three main state-of-the-art fusion techniques available at present. These are: Németh's implementation of warm fusion (Launchbury & Sheard, 1995, Németh, 2000), Chitil's type inference-based approach to deforestation (Chitil, 1999), and the HYLO fusion system of Onoue, Hu, Iwasaki, and Takeichi (Hu, Iwasaki, & Takeichi, 1996, Onoue, Hu, Iwasaki, & Takeichi, 1997).

4.2 Technical Issues and the Individual Projects

Measuring the effects of fusion on substantial application programs in the context of a real compiler is not an easy task. Significant technical issues must be dealt with in order to insure that the measurements obtained are meaningful.

- To make meaningful comparisons of the three fusion implementations, they must all run in the same computing environment. We have chosen the most recent and stable version the Glasgow Haskell Compiler (GHC version 4.08) as the common platform for several reasons. GHC is itself a Haskell program that was specifically designed to support experimentation with new program transformations such as fusion. GHC also provides profiling tools for measuring the time and space requirements of compiled programs. These are essential for evaluating the efficacy of fusion on benchmarks. Another important advantage of GHC is that two of the fusion implementations (Németh's warm fusion engine and the HYLO system) are already implemented in GHC, albeit in slightly older versions.

  The differences between GHC versions are significant enough that updating the two existing implementations to GHC-4.08 requires them to undergo non-trivial modification. In addition, the third implementation (Chitil's type-inference based approach) was not developed in GHC at all. Instead, it runs in the Haskell interpreter HUGS and transforms programs written in a small language that is a pared down version of Core, GHC's intermediate language. Extending Chitil's implementation to handle full Haskell and porting the extended implementation to GHC is a significant project.

  Providing a common environment requires modifying all three implementations in ways which reflect detailed knowledge about their underlying algorithms, as well as about different versions of GHC. Three different students therefore took on the tasks of *understanding the three algorithms and their implementations*. Since extending Chitil's implementation is the biggest task and could clearly not be accomplished during the summer, it was undertaken by a student who would be continuing her work as part of an honors thesis during the following academic year.

- Our project requires significant program development, especially in the case of extending Chitil's implementation. Although it is essential for all the fusion implementations to eventually run in the same version of GHC, it is not necessary to do all program development using GHC. Indeed, there are several reasons why GHC is *not* an ideal environment for developing extensions to the fusion implementations. First, such development requires modifying the implementation of GHC. GHC is a very large and complex program, and adding new Core-to-Core transformations is tricky. Second, GHC compile times can be *very* long, especially when recompiling GHC itself. This results in painfully slow edit-compile-debug cycles. Finally, GHC currently does not support any sort of interactive mode, so that any testing/debugging has to be done in batch mode. This is very inconvenient.

  We would prefer to develop new code in a more friendly, interactive environment in which program development and experimentation can be accomplished both more easily and more quickly than in GHC. The HUGS Haskell interpreter provides precisely such an environment. However, unlike GHC, HUGS does not provide tools for manipulating Haskell programs in Haskell — a fact which makes it difficult to test HUGS-based fusion engines using real Haskell programs as input. Also, while HUGS is a good environment for developing programs, it is not a good environment for benchmarking them. Because interpreter-based approaches to program profiling do not model many complexities of real systems, they are not suitable for studies whose goal is to measure performance in practice.

  Together, these observations suggest that the development and evaluation of fusion implementations is best done using a combination of GHC and HUGS. HUGS can be used for those stages of development where interactivity and rapid turn-around time are important. In order to test HUGS-based fusion engines on Haskell programs as input, we imagine an *IR converter* that allows the intermediate representation (IR) of programs at various stages of compilation to be moved back and forth between GHC and HUGS. Development of such a converter was undertaken by one of our students.

- In order to measure the effectiveness of fusion, it is necessary to have good suites of *benchmark programs*. There is an existing suite of Haskell programs (the `nofib` suite (Partain, 1992)) that is commonly used to test the performance of Haskell implementations. This suite contains both simple toy programs as well as real applications. While we will use the `nofib` suite in our comparative study, additional test programs are also needed. There are many features of the fusion engines that will not necessarily be exercised by standard benchmarks, and so new benchmarks that specifically test these features must be written. Moreover, the existing benchmarks were not written with fusion in mind; their authors might have written them in a more modular style had they known that fusion would be performed. For this reason, it is important to design some new benchmarks specifically designed to showcase the benefits of fusion.

  Over the summer, one of our students began construction of a new `nosquares` benchmark suite containing toy programs from classic papers in the fusion literature, as well as more substantial application programs written specifically for the purpose of testing fusion implementations. As part of testing the HYLO system (to which we currently have only a black-box web interface), another student developed a `nofuse` suite of toy programs highlighting the successes and limitations of that system. We will use all three benchmark suites in our comparative study.

- Using existing application programs as benchmarks is essential for measuring the effects of fusion in practice. This means that the fusion engines must fully handle all features of Haskell. One important feature of real Haskell programs that is not handled effectively by any of the existing fusion engines is their expression as multiple source program modules. (Chitil has developed a technique to address cross-module fusion, but is has not been implemented.) The existing fusion implementations all assume that each of their input programs is written as a single module. In order to be able to test the fusion engines on existing application programs, one of our students took on the project of building a *demodulizer* that automatically transforms multi-module Haskell programs into single-module programs that can be processed by the fusion implementations we are studying.

In designing the projects sketched above, there was a tension between scoping the project so that it could be completed by the end of the summer and making the project a realistic part of the larger comparative study. We tended toward the latter, expecting that it would give the students a better sense of a real research project and make them more full-fledged participants in the project. While we intended that the students could make reasonable progress on the projects during their six weeks of individual research, we realized from the outset that none of the projects were likely to be completed by the end of the summer.

We had hoped that most of our students would continue working on their projects during the academic year following the summer. As discussed in more detail in Section 5, five of the six students continue to be involved with it in some capacity. Work on the IR converter, the demodulizer, and the test suites continues, as does the extension and incorporation into GHC of Chitil's type inference-based fusion implementation. Some preliminary measurements with Németh's implementation of warm fusion and the HYLO system have been taken, but substantial benchmarking remains to be done once all of the necessary machinery has been constructed and put into place.

5 Structuring the Research Experience

LSC can be seen as comprising three distinct phases: (1) a mini-course on compiler-based program optimization which occupied the first three weeks of the research experience; (2) the remaining seven weeks of the research experience during which the students chose and worked on their individual research projects; and (3) the continuing research effort. We briefly describe the salient characteristics of each phase.

5.1 The Early Days

LSC kicked off with an informal social gathering over dinner, followed early the next morning by our first official meeting. We began the meeting by first reminding the students of our research and pedagogical goals for the collaborative project in which we would all be engaged, and then providing a technical overview of the project. Over the next few days we identified a number of possible individual projects and their relationships to the collaborative project. This information was summarized on the LSC web page (`http://cs.wellesley.edu/~deforest`), which we encouraged the students to consult frequently as they decided which of the projects they were interested in undertaking. Students finding none of the projects we described appealing were encouraged to propose their own. By the end of the fourth week of LSC, we explained, they were to have settled on a project and begun work on it.

To prepare the students for selecting their projects we gave a three week mini-course covering the background information necessary to understand, at a relatively high level of abstraction, what each of the proposed individual projects would entail. We were particularly concerned that the students be able to identify those projects that were most in accordance with their own intellectual goals for the summer, that they were most interested in, that they felt best prepared to undertake, and that they thought they would most enjoy. Consideration of these issues was necessary for each possible project individually, since these varied widely in terms of the background and skills they required, the degree to which they were on the critical path of the collaborative research endeavor, and the positions they occupied on the practical-theoretical continuum. The mini-course served another important purpose as well, providing us and our students a common language and point of departure for the technical discussions that were to come.

A key goal of the mini-course was familiarizing students with Haskell programming and issues in transformation-based program optimization. While all six students had some experience programming in functional languages, only two had significant Haskell experience, and none had significant experience with writing programs that transform other programs. We gave brief lectures on these topics and posed program transformation exercises related to our collaborative research project. Students worked in pairs to implement program transformers of increasing sophistication. Working in pairs (which changed over time) helped the students to get to know each other, and was an effective means of having students share their background and experience. Interspersed with machine time were group sessions during which students discussed the problems or presented their solutions for peer review.

A second, but equally important, goal for the mini-course was to get the students up to speed on the fusion literature. Towards this end, we established a "reading group," a series of two- to three-hour sessions held twice a week in which we read aloud and worked through — often under student leadership — the details of research papers on various topics related to fusion. Making significant progress in this area proved to be rather difficult given the three week time frame with which we had to work. It involved helping the students make the transition from (perhaps) having read textbooks and worked problems on programming language and compiler design to reading research papers pushing the limits of knowledge about them. The leap in technical difficulty and assumed background between the two media did not go unnoticed by our students, who came to understand that learning to read research papers at all — even when one has the background to make sense of them — is itself a significant achievement.

Broadly speaking, then, the mini-course was designed to help the students make the transition from learning science to doing science. Of course, no amount of discussion of the research process can actually impel a student to make that transition; indeed, the entire premise on which undergraduate research programs are based is that only first-hand engagement in the research process and the discovery of knowledge can have that effect! But we believe that it is possible to create an environment in which the transition from student to student researcher is not only possible but highly likely, and in which it is a pleasurable and exciting, rather than disorienting, experience.

5.2 A Day in the Life

LSC became much less structured after the mini-course had ended. The students spent the first week after the mini-course investigating and choosing an individual research project that they would work on for the remaining six weeks of the summer. Some group activities from the mini-course did continue, however; reading group, for example, continued to meet twice a week for several more weeks. We also instituted a weekly group meeting. These provided a time for all of us to catch up on the progress others had made during the week, as well as to generate new ideas and help one another move forward on ideas already under exploration. Still, the time devoted to whole group activities decreased dramatically after the third week of the research experience, and the amount of time spent engaged in independent work, or in individual or small group conversations focused on specific technical aspects of the project, increased proportionally.

Recognizing that flexibility in work hours can often lead to a more productive work environment (and also that research sometimes requires extended periods of long hours which can be compensated for at other times), strict work hours were not set. Nevertheless, all eight of us were typically present in our research area from about 9 a.m. to 5:30 p.m., Monday through Friday, and often during evenings and weekends as well. A typical day at LSC might include a group meeting or a reading group session, a small group or independent consultation with one or both of us, collaborative activities between students working on related projects, and independent reading, programming, and/or thinking.

Throughout the research experience, students attended talks, panels, and social events organized as part of the Wellesley Science Center summer research program. They also kept research notebooks, which we reviewed on a weekly basis. At the end of the research experience each student submitted a formal written document to be posted on the project web page detailing the specific project they had undertaken, together with their findings. These documents took shape over the final five weeks of LSC; their polished versions were the result of a rigorous process of writing and revising based on the numerous and detailed comments we provided. Each student also prepared a poster for the poster session which marked the end of Wellesley's summer research program. The Bates students' posters were also presented at the Student Poster Session held during the 2000 Celebrate Bates weekend.

5.3 The Story Continues

While significant progress was made on all components of the project during the summer, the scope of the project was large enough that no part was complete by the end of the summer. Although we had not carefully planned for this scenario, it is fortunate that most of the summer students have been able to continue to work with us on their pieces of the collaborative project.

One student is continuing her work extending Chitil's implementation as her honors thesis project. Another wanted to continue work on the IR converter as a thesis project but, due to sabbatical scheduling, cannot; instead he is continuing his work on the project for pay. Two more students are continuing their work as paid research assistants during the academic year, while another is planning to continue her work in an independent study course in the spring. Only our newly graduated student is no longer closely associated with the project, reluctantly forsaking it for the demands of graduate school. We are, of course, delighted that all of the students found their summer experiences worthwhile enough to want to continue to be involved in the project. Again due to sabbatical scheduling, it is unclear at present whether or not the project will continue next summer.

When the empirical studies of the three deforestation techniques are complete, we plan to submit a journal paper comparing the techniques co-authored by all LSC participants. We are also encouraging students to present their individual work in other forums. For example, four of our students submitted abstracts for poster presentations at the Northeast Conference of the Consortium for Computing in Small Colleges in April 2001. All four were accepted, and were chosen as among "the top submissions" by the conference's Student Poster Co-Chairs. We expect that several of the individual projects will be worthy of additional workshop presentations or conference publications.

One of LSC's grandest moments came this past September at the 2000 Haskell Workshop and the International Conference on Functional Programming. Four of our students accompanied us to the meetings. We were eager to introduce them to the larger functional programming community and have them meet a larger group of people actively thinking about ideas similar to those with which they worked all summer. Our students — who, incidentally, are the only undergraduates we have ever seen at these conferences — and our informal descriptions of the technical and pedagogical goals of our research project were exceedingly well-received. Involving undergraduates in research in programming languages is virtually unheard of, and a number of researchers — including some very big names in the field — were extremely enthusiastic about it. Their praise for our efforts to educate early the next generation of researchers left us feeling we're doing something of significance for the functional programming community as a whole, as well as for our students and ourselves.

6 Lessons Learned

Leading LSC taught us a good deal about the feasibility of undergraduate group research in computer science. We came away from the experience with a number of thoughts about its successes and failures, the most significant of which we discuss in this section.

We were pleased with the logistical structure — mini-course, group meetings, reading group, etc. — of the research experience. Moreover, we found that six students and two faculty members was a very good number of each to have, and that the variety of prior experience and expertise — at both the student level and at the faculty level — afforded by the cross-institutional nature of the research experience was of considerable importance to its success. We saw our students take advantage of many opportunities to teach and learn from one another, and to see how their peers (rather than their faculty mentors) understand and experience computer science and computer science research. Each of us also found working with another faculty member with similar professional and pedagogical interests extremely stimulating. Precisely because our line of work can be so relentlessly solitary — especially for those of us housed in small departments — we very much appreciated the opportunity to engage in research and pedagogy with another faculty member.

The biggest problem we faced with LSC was the short duration of the summer research experience. Given our starting point for the project, ten weeks simply was not enough time to make the technical progress we would like to have made. This was due to several factors. First, the amount of time allocated for individual projects was very short. Discounting three weeks for the mini-course, one week for choosing a project, and one for preparing posters and final papers at the end left only five weeks of "prime time" research for the students. Moreover, even during these five weeks, significant time was spent in group activities such as group meetings and reading group.

Second, as already mentioned, the individual research projects did not constitute tidy units of research that could easily be completed during the summer. Rather, they were sizable chunks of a larger ongoing collaborative project. We feel that the size and scope of the collaborative project was necessary to accommodate six students' individual research projects and a variety of technical interests. However, a drawback of decomposing the collaborative project into sensible parts was that the size of those parts made it unlikely for a student to achieve closure on such a part during the summer.

A third factor was that we did not properly prepare our computing environment prior to the start of the research experience. An unexpectedly large fraction of our time was spent installing and understanding GHC, tracking down the source code for various fusion implementations, and getting these implementations running. In many cases, individual progress was blocked because some part of the computational environment was not working as expected. The fact that LSC began just two days after the end the semester contributed to the lack of preparation, but we should have better anticipated these sorts of difficulties. Learning from past missteps, we will be careful in the future to have as much of the necessary software as possible running before our research experiences begin. Doing so for LSC may not have completely alleviated the time crunch we faced, but we feel it would have helped to large degree. Most importantly, it would have freed us to spend the time we did have together with our students focused on fusion, rather than on the delicacies of software installation.

The time crunch we faced had consequences for the structure of LSC. For example, although we found reading group a particularly good device for helping students develop the ability to work through research papers, for establishing a common technical vocabulary, and for generating a body of shared knowledge and experience, it was very time-intensive. In fact, reading group consumed the equivalent of nearly one full day's work each week. Toward the end of the research experience we reluctantly discontinued it in favor of devoting the little remaining time we had available to us to the project itself. We similarly dropped our initial plan to have the students give oral presentations of their work to the group at the end of the summer, but given the emphasis we had placed on oral and written communication throughout the research experience, and given that each student presented their work orally at Wellesley's summer research poster session, we did not consider this particularly problematic.

It is worth remarking that, even had we made more technical progress during the summer, work on the collaborative project would still be continuing. But because the students' research was really just getting going when the intensive research experience ended, it has turned out to be important that all but one of them are willing to continue their projects into this academic year. Even were this not the case, we suspect that having the same students involved in the same research project for two summers, and perhaps during the academic years as well, would be ideal. Regardless, it may be wise to prepare students for the summer research experiences in specially tailored independent study courses during the preceding academic year, and to follow up with independent studies or theses whenever possible.

We feel that LSC successfully enabled each of our students to make the transition from coursework to research. Each student was able to acquire the technical expertise required to make progress on his or her project, as well as to demonstrate the expertise acquired both orally (in individual conversations with us and one another, in their contributions to reading group and group meetings, and with the public during the wrap-up poster session) and in writing (in their notebooks and formal project documents). Nevertheless, some students made more significant progress on their projects than did others. In some cases, lack of student progress was largely attributable to our failure to make the necessary tools available to them early enough in the summer; in others, student initiative was lacking. In addition, half of our students lived off campus, and this made it somewhat difficult for them to join us in working late in the way that we had imagined.

Despite this, all of the students exhibited a reasonable degree of commitment to the project, and some were very enthusiastic indeed. All worked independently when appropriate, and yet seemed always aware of themselves as contributors to a collaborative effort. We are immensely pleased that such a strong spirit of camaraderie, inspired by the work at hand, seemed to infuse LSC; certainly there was no shortage of technically-based jokes or good-natured teasing. The sense of intellectual community we tried so hard to foster really did blossom and assume a life of its own.

7 Recommendations

Based on our experiences leading LSC, we offer the following recommendations to colleagues considering undertaking similar undergraduate research initiatives at other small colleges.

7.1 Collaborative Research

In many areas of computer science — and certainly in the area of programming languages — modern research projects are rarely solitary efforts. Instead, a project will typically concentrate the efforts of a critical mass of researchers, each of whom is pursuing one or more aspects of it. Many benefits derive from such collaborative research. For instance, groups of researchers can collectively undertake projects whose size and scope are beyond the reach of any one individual. In addition, the different experiences, perspectives, and expertise that individual researchers bring to collaborative projects can be crucial to their success. This is especially the case for modern research projects in computer science, since these are increasingly likely to span several traditional subdisciplines. Moreover, researchers working in collaboration with others reap the benefits — most notably, learning from and teaching one another — of involvement in what is effectively a small, focused research community.

One important aspect of undergraduate research experiences is that they offer students a sense of what it is like to pursue research in their chosen disciplines. Since research in graduate school and professional settings is increasingly likely to involve collaborative research, it is important for undergraduate research experiences to reflect this. We therefore recommend structuring undergraduate research experiences around collaborative projects capable of accommodating groups of faculty and groups of students whenever possible.

Even when it is not possible to involve more than one faculty member in a project, many of the benefits of collaborative research can still be attained by involving a group of students. It is essential that new researchers learn to share ideas with one another, to mentor one another, and to distill and solve problems together. In our view, the kind of student/student peer mentoring relationships to which these interations give rise are every bit as important as the highly touted faculty/student mentoring relationship.

The broad scope and large number of participants in collaborative research projects make their persistence beyond a single summer research experience quite likely. As with LSC, this can provide researchers an opportunity to work on problems which cannot be completed within a ten-week time frame. Longer-term involvement with a research project helps students to develop a better understanding of the intricacies of their projects, as well as to make more progress toward their completion. It also helps students understand the kind of commitment required to successfully pursue research. Finally, long-term involvement with a research project helps build community bonds between participants. Ideally, a long-lasting project can assume a life of its own, surviving even the student turnover inherent at undergraduate institutions which do not have graduate programs.

7.2 Cross-Institutional Experiences

Our experiences with LSC have convinced us that cross-institutional undergraduate research collaborations offer many rewards for both students and faculty. This is in large part because such experiences amplify the "community" benefits of collaborative projects discussed above. We strongly recommend that colleagues planning undergraduate research experiences in computer science explore possible collaborations with like-minded faculty in their areas, especially with those in their geographic areas.

Faculty who are fortunate enough to have departmental colleagues with whom they share research interests should consider collaborating with them. Some of the very best aspects of LSC derived from having more than one faculty member involved in the same project. But although working with colleagues from one's home institution can simplify many logistical problems associated with cross-institutional research experiences, it is still worth looking outside the home institution for collaborative partners. Faculty and students from other schools often bring with them different backgrounds and cultures, making Alberts' "collisions of ideas" all the more likely.

Cross-institutional collaborations require careful planning. It is important for the collaborating faculty to meet — ideally several times and far in advance of the planned research experience — to iron out as many details as possible. Finding common intellectual and pedagogical ground and planning an experience well-suited to faculty and students from all participating institutions can take a significant amount of time. Securing funding for both faculty and student researchers is also a high priority. In our case, we already had some funding at our disposal when LSC was conceived but we still needed to seek out additional student stipends. It is worth noting that a number of funding agencies encourage or have special programs supporting cross-instititional collaborative research.

Although it might be possible to organize a collaborative research experience based on frequent visits between groups of researchers stationed at their home institutions, we strongly recommend that all participants be present at the same location during the core part of the experience. Physical proximity is a key factor in establishing a sense of camaraderie and intellectual community. Of course, it also raises a number of challenges, such as choosing a host institution and finding housing for all participants near the host institution. We are nevertheless confident that the benefits of physical proximity are well worth its costs.

7.3 Preparation

Successful undergraduate research experiences require significant planning and preparation. It is necessary to work out logistical details — such as funding, timing, housing, and number of participants — well in advance of the planned experience. Designing a project of suitable scope and focus, and which is accessible to undergraduates, also requires careful consideration. We have found the kinds of projects highlighted in Section 3 particularly amenable to undergraduate research.

Early selection of student participants is extremely important to the success of a research experience. This is in part because the backgrounds of the particular individuals to be involved in a research project must be taken into account when designing it, and in part because high-paying jobs often lure the best and brightest students away from research experiences. One way to counteract the latter is to identify promising students early in their undergraduate careers and prime them for eventual participation in research experiences through relevant coursework and independent study projects. Since time in a typical summer research experience is short, helping students gain facility with important concepts and tools before the experience begins is a worthwhile endeavor.

In LSC we learned the hard way that appropriate support stuctures — especially software applications and program development environments — should be in place *before* the start of a research experience. Again, time during a research experience is extremely valuable, and it should not be squandered on software installation or other tasks that divert attention from the research itself.

7.4 Activities

Although the activities appropriate for any particular research experience depend crucially on the nature of the project around which it is organized and the abilities and backgrounds of its participants, there are some activities whose benefits are fairly universal. Frequent meetings between faculty and students are certainly an essential part of an undergraduate research experience. Additionally, we recommend having students produce several different artifacts — indeed, several different *kinds* of artifacts — during a research experience. Students might, for example, be required to keep notebooks or design journals, write papers which summarize the research they have undertaken and the results they have obtained, and/or present posters or give oral presentations describing their research. We find that the production of such artifacts encourages students to reflect on their experiences, and also helps them monitor their research progress. It is essential that the artifacts produced evolve over the entire duration of the research experience and that their production becomes an integral aspect of it, rather than just being generated in a mad dash at the end. Faculty members can emphasize the importance of the artifacts — both to the students' development as researchers and to the future of the project itself — by offering frequent, plentiful, and thoughtful feedback on them.

We believe that several other of the activities we experimented with in LSC would be worthwhile for many collaborative projects. A mini-course can provide an excellent opportunity for researchers to learn to work with one another, as well as to ensure that everyone is "on the same page" with regard to the project. On the other hand, such a course necessarily takes time away from individual research, and so its length and scope must be carefully considered.

The benefits and drawbacks of reading groups are similar to those of mini-courses. But although reading technical papers is a skill that is essential to the research enterprise, it is very seldom taught. A reading group provides a forum in which those who have experience reading technical papers can mentor others in this critical skill. Reading groups can, of course, be used effectively outside the context of a collaborative research project. They may even be used to good effect in situations in which there are only loose connections between individuals' independent research projects.

Collaborative research experiences are enhanced by various kinds of group meetings. Regularly scheduled formal group meetings provide opportunities for researchers to report on progress made, to seek the advice and encouragement of other group members, and to learn about the work of other members of the group. Preparing to report on progress at group meetings requires that students develop the ability to describe their research activities to others. To build intellectual community and encourage camaraderie, it is also important to promote informal activities involving group members. In LSC, we often used food as a social lubricant to bring project participants together for both technical and non-technical discussions.

7.5 Long-Term Projects

Undergraduate research experiences are often limited to eight-to-twelve week periods during the summer. However, we have argued in this paper that there are advantages to designing longer-lived research projects that do not strictly adhere to this convention. The standard model for undergraduate research experiences is, of course, valuable, but it does not tell the whole story. We have found that this model can fruitfully be enhanced by engaging students in a variety of additional research-related activities before, during, and after the summer research experience. Such activities might include advanced courses, independent study projects, academic-year research projects, research talks, workshops, and conferences. We are convinced that extending the standard model for undergraduate research is essential if we are to make available to our students more varied and more realistic opportunities to do science.

Acknowledgments

We heartily thank the six students who participated in Lumberjack Summer Camp: Kirsten Chevalier, Nausheen Eusuf, Kate Golder, Holly Muenchow, P. Chris Staecker, and Aaron Wheeler. Our gratitude also goes to those committees and agencies that funded the research experience, to Wellesley College for hosting it, and to those faculty involved in the Wellesley Science Center summer research program who were kind enough to include us and our students in their activities. Patricia Johann is supported, in part, by the National Science Foundation under grant CCR-9900510, and was on the faculty at Bates College at the time of LSC. Franklyn Turbak is supported, in part, by the National Science Foundation under grant CCR-9804053.

References

Bard, G., Berque, D., & Dershem, H. (1996). Finding and developing research experiences for undergraduates in the small college setting. *The Journal of Computing in Small Colleges*, 12(2), 94–95.

The Boyer Commission on Educating Undergraduates in the Research University. Reinventing undergraduate education: A blueprint for America's research universities. Available at `http://notes.cc.sunysb.edu/Pres/boyer.nsf`.

Chitil, O. (1999). Type inference builds a short cut to deforestation. In *Proceedings of the 1999 ACM SIGPLAN International Conference on Functional Programming*, 249–260, ACM.

Dima, C., Parent, G., Briggs, A., & Dickerson, M. (1998). Experimental analysis of polygon placement problems: An undergraduate research project in computational geometry. *The Journal of Computing in Small Colleges*, 13(5), 13–24.

Hu, Z., Iwasaki, H., & Takeichi, M. (1996). Deriving structural hylomorphisms from recursive definitions. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming*, 73–82, ACM.

Hughes, R. J. M. (1990). Why functional programming matters. In David Turner, editor, *Research Topics in Functional Programming*, 17–42, Addison Wesley.

Koelzer, J. G. (1997). Undergraduate research in computer science at a small college: A case study. *The Journal of Computing in Small Colleges*, 12(4), 329–332.

Lopez, A. M. & Messa, K. C. (1994). An undergraduate research program in multi-paradigm software design. In *Selected papers of the twenty-fifth annual SIGCSE Symposium on Computer Science Education*, 271–275, ACM.

Launchbury, J. & Sheard, T. (1995). Warm fusion: Deriving build-catas from recursive definitions. In *FPCA '95, Conference on Functional Programming Languages and Computer Architecture*, 314–323, ACM.

Németh, L. (2000). *Catamorphism Based Program Transformations for Non-Strict Functional Languages*. PhD thesis draft, Department of Computing Science, University of Glasgow.

Onoue, Y., Hu, Z., Iwasaki, H., & Takeichi, M. (1997). A calculational fusion system HYLO. In *IFIP TC 2 Working Conference on Algorithmic Languages and Calculi*, 76–106, Chapman & Hall.

Partain, W. (1992). The nofib benchmark suite of Haskell programs. In *Glasgow Workshop on Functional Programming*, 178–194, Springer-Verlag.

Passos, N. L. (1999). Implementing a true undergraduate research experiment. *The Journal of Computing in Small Colleges*, 14(3), 86–94.

Sturm, D. & Glassman, R. (1996). Mentored undergraduate projects: A research program for women in computer science. *The Journal of Computing in Small Colleges*, 12(2), 19–23.

Tesser, H., Al-Haddad, H., McClaugherty, S. & Frame, J. (1998). Experience with an undergraduate research project: Software reuse and interoperability study. *The Journal of Computing in Small Colleges*, 14(1), 86–95.

Wadler, P. (1990). Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73, 231–248.