Opening up the Collaborative Problem-Solving Process to Solvers

Tyler Robison

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Steven L. Tanimoto, Chair

Stephen Kerr

Richard Anderson

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

Opening up the Collaborative Problem-Solving Process to Solvers

by Tyler Robison

Chair of Supervisory Committee:
Professor Steven L. Tanimoto
Department of Computer Science and Engineering

In software systems, having features of openness means that some of the internal components of the system are made available for examination by users. Researchers have looked at different effects of open systems a great deal in the area of educational technology, but also in areas outside of education. Properly used, openness has the potential to teach users about the system, and about themselves, without drowning the user in a sea of confusing information.

This work explores features of openness that I have added to the collaborative problem-solving system known as CoSolve. Collaborative problem-solving software allows several solvers, often remotely and possibly asynchronously, to work together to create a solution to a problem known to all of them. For the purposes of this work, the problem being solved is well defined. In addition to making remote and asynchronous solving more practical, collaborative problem-solving systems can enhance the

problem-solving process by providing additional services to solvers, such as record-keeping, suggestion and evaluation tools.

The contributions I present here are 1) the addition of features of openness to CoSolve, and an evaluation of those open features, and 2) a collection of tools for analyzing and understanding CoSolve behavior. The first contribution improves CoSolve by making it easier for solvers to collaborate. The second introduces a way to understand the seemingly chaotic and complex behavior of solvers in the CoSolve system.

TABLE OF CONTENTS

Acknowledgments

I’d like to acknowledge the unwavering support of my family: my parents, brother, sister, brother-in-law, sister-in-law and three (soon to be four) nieces.

Partial support from the National Science Foundation under grants IIS-0537322 and 0613550 is gratefully acknowledged.

Chapter 1: Introduction

We've all had the unfortunate experience of struggling with a computer program in order to get it to do what we want, whether it's locating a particular feature, determining whether or not an action is possible, or telling us why some behavior did or did not occur. If we were communicating with a person instead of with a program, things would likely be much easier: we could merely say "I want to turn off auto-correct," or "I want to know whether I can compile several word processing files together into a single document" or ask "Why did the formatting on this file change?" and the person would weigh the request or question in light of the context and figure out what specifically should be done, or what to answer. Modern computers, being more constrained in their ability to interpret and reason (at least in the sense of how humans do it), generally need to be informed more precisely what to do, either by the user or the programmer. And so a user's experience may be one of frustration as he† tries to figure out how to get the computer to perform a task that he understands clearly, and that he could easily convey to a human.

This state of things should come as no surprise, and enhancing the quality of interaction between computers and their users is, and has long been, a large area of research. And of course no one research project will be able to solve this massive problem, but the hope is that by approaching the problem from different directions and chipping away at it we can gradually make progress towards smoother interaction.

What if the system could teach users as they used the tool? Not just teach them about the tool via a standard 'help' interface stating "This button does such-and-such," or "Activate this feature in this menu," but instead help teach knowledge about the task itself, knowledge whose relevance could in theory extend beyond the tool. Perhaps 'teach' is not the right word here, as people using tools are often focused on being productive; a user may not be responsive to a lecture on proper spelling when hurriedly typing a paper on a tight deadline. Rather, such an interface makes it easier for users to learn.

† In this work, the word 'their' will be used to mean the possessive form of 'he' and 'she,' as well as the possessive form of 'they.' Also, when writing about solvers, 'he' and 'she' will be used in alternating chapters.

Instead of only simplifying the tool or adapting the tool to individual users—though both are valid techniques—I am interested in ways for users to 'look inside' the tool to learn how the tool works, learn how to accomplish the task, and perhaps learn about their own knowledge and behavior as well. This goes beyond the static materials of traditional help systems and allows users to inspect the 'moving parts' of the tool, as well as perhaps catch their own reflection. These ideas are known as 'openness' or 'transparency,' and can allow the user to learn more about the task, the tool, and himself.

1.1. A Definition of Openness

For the purposes of this work I define openness in software systems as follows:

> "A software system is said to be *open* if details of the system's state or design, though unnecessary for the successful usage of the system, are made available for inspection by the user."

Rather than viewing a system as simply open or not open, it is more useful to view openness as a spectrum. Most software systems possess at least a small degree of openness: consider a tutorial that shows the learner his score; this information isn't strictly necessary for using the tutorial, though the learner may appreciate seeing it. Further along the spectrum would be a similar tutorial that also lets the learner see how that score was calculated. Even further would be a tutorial shows the details of how the score is calculated, the reasoning behind that process, and how the tutorial program uses this information to guide the tutoring process.

This work explores the idea of openness in the area of collaborative problem-solving in an attempt to help alleviate some of the difficulties faced when a group of solvers comes together—perhaps remotely and asynchronously—using computers to work on a shared problem.

1.2. Contributions of this Work

At a high level, the aim of this work is to explore openness in collaborative problem-solving tools, and in order to do so, features of openness have been added to CoSolve, a collaborative problem-solving system. These open features are collectively referred to as the CoSolve Consultant. The resulting system was then evaluated by having solvers use the system with and without the Consultant to collaboratively solve a problem. From this comparison I was able to develop a picture of how solvers used the features of openness, and how the openness impacted collaboration and problem-solving.

In addition, I present a series of tools for analyzing the use of CoSolve by solvers. Independent of the openness features above, these analysis tools can provide information on how solvers collaborated, what their patterns in solving behaviors were, and how these behaviors changed over time. These analysis tools are then used to inspect solvers using CoSolve in the evaluation of the CoSolve Consultant. While potentially useful to solvers themselves, these analysis tools are designed for use by those who are interested in understanding the solving process and individual solver behavior: researchers and instructors.

To state this more concisely, the contributions of this work are:

1) The implementation and evaluation of features of openness in Cosolve, a collaborative problem-solving environment
2) A suite of tools for analyzing and understanding the behavior of solvers in CoSolve

The former contribution sheds light on how openness can enhance collaboration and guide solvers, while at the same time highlighting some potential pitfalls it can cause. The latter contribution provides additional tools for understanding solvers in CoSolve, and yields additional ways to evaluate the problem-solving process.

While the contributions here are presented in the context of CoSolve, I believe that they can be used in other collaborative problem-solving systems as well. This topic is discussed in chapter 6.

1.3. Thesis Organization

In chapter 2, relevant background information will be discussed. In order for the discussion of this thesis to take place, we first need to touch upon a number of areas of research that are of some relevance, including collaborative problem-solving, open learn models and the educational theory of Cognitive Load. The chapter also describes my earlier work in the area of openness. The chapter will present the context for the discussion to come.

Chapter 3 describes my earlier work in the area of open systems. I describe different open systems I implemented, tracing the ideas that were used to add features of openness to CoSolve.

Chapter 4 describes CoSolve, an online environment for collaborative problem-solving and design developed by our research group at the University of Washington. It is important to go into CoSolve in some detail as my later research revolved around developing for the CoSolve environment. The chapter will then go on to describe the features of openness I added to CoSolve, collectively referred to as the CoSolve Consultant. The chapter also describes the study performed to evaluate how the CoSolve Consultant was used by solvers, and how it affected solvers, and to develop an understanding of solvers in CoSolve.

Chapter 5 explores the results of the study described in chapter 4. Analysis of these results and discussion of their meaning are presented in this chapter. Also presented, in the context of developing this analysis, are tools developed for understanding solvers using the CoSolve environment.

In chapter 6 I will present a summary of the contributions and findings of this work, and argue the case for the application of these techniques beyond CoSolve, to other collaborative problem-solving systems.

Chapter 2: Related Work

In this chapter I will explore the background necessary for the discussion of my work. In order to build a solid foundation for the rest of the thesis, a number of topics need to be covered, including issues of open learner modeling and collaborative problem-solving systems. While much of the literature covered here falls into the domain of Computer Science, it is also necessary to investigate work outside of Computer Science, including metacognition and Cognitive Load Theory.

At the heart of my work is the concept of openness, that is, revealing aspects of the internals of a system to its user. The discussion will start with the concept of openness in educational technology, as that area contains many of the inspirations for this work. Then the discussion will move to examples of openness in non-educational systems. After a foundation is laid for openness, the area of collaborative problem-solving will be addressed, including previous attempts at openness and opportunities for improvement, and then Cognitive Load Theory will be presented so that it can be used as a tool for managing complexity in the interface.

2.1. Openness in Educational Technology

One of the primary inspirations for this work came from ideas of openness seen in some areas of educational software. On a high level, openness in educational software exposes some of the inner-workings of the system to the learner. The view may, for instance, reveal to the learner what the educational system believes about her or how her scores are being calculated [Bull 2004, Bull and Kay 2005, Bull and Nghiem 2002, Cook and Kay 1994, Greer and Zapata-Rivera 1999, Kay 1997, Mabbott and Bull 2006, Mitrovic and Martin 2002, Morales, Pain and Conlon 2001]. This sort of information shown doesn't have to be knowledge in the domain being learned, and so won't necessarily boost learning of the subject material, but can provide other benefits such as giving the learner a better awareness of her own knowledge [Morales, Pain, and Conlon 2001, Zapata-Rivera and Greer 2000], helping to develop a better understanding of the knowledge space, or teaching skills that could transfer to other settings, such as ability to assess herself. For instance, a learner using an algebra tutorial may find out about her strengths and weaknesses in the area of algebra: this

knowledge won't by itself help her solve algebra problems, but can be of benefit less directly in guiding learning efforts.

There are several aspects of educational systems which can be made open to the learner. One of the most relevant ideas explored by the research community is that of open learner models, which embrace the notion of exposing the system's beliefs regarding the learner to the learner for inspection, and possibly opening up the model to greater interaction as well. Many educational systems already maintain a model of the learner's knowledge level, scores, et cetera. A simple example of this would be a learner using algebra instruction software, and seeing the program's diagnosis that the learner understands concept A very well and concept B very poorly. Open systems, at their simplest, merely reveal that learner model to the learner.

Note that the uses of learner models are not restricted to educational systems: a person using any system (software or otherwise) will necessarily have some understanding of what that system is, what it can do and how to use it. This model could vary from a very simple 'black box' model of "I run the program and it generates the correct results" to a more complex one involving numerous features shortcuts to use and knowledge of bugs and how to avoid them. Educational systems very commonly use learner models to represent the user's progress so far, or what the user knows and does not know. For instance, the educational game Refraction, designed to teach users about fractions, keeps track of the learner's current stage in the game as well as the number of coins achieved for meeting additional challenges [Andersen et al, 2011].

Researchers have argued that there are several benefits to be gained from the use of open learner models (OLM), including the refinement of the learner model [Bull, Brna and Pain 1995, Bull and Pain 1995, Cook and Kay 1994] and the prospect of getting learners to reflect on their knowledge and learning [Bull and Pain 1995, Morales, Pain and Conlon 2001]. As such, numerous researchers have addressed various issues in this space, such as examining different ways of showing information to the learner and different types of information to show.

Historically, traditional tutoring systems did not let the learner access her learner model directly [Bull 2004, Morales, Pain and Conlon 2001]. The learner could often only see high level performance information, or infer the system's beliefs regarding her

through its behavior. This is likely due to the fact that adding open learner models would take additional time and effort for the designers and programmers. In 1990 John Self argued that learner models should be opened up to learners [Self 1990]:

> "Indeed it would be a salutary principle to insist that all student models be made open to the student. This might benefit ITS (intelligent tutoring systems) design by reducing the temptation to include crude, ad-hoc classifications, and, more importantly, may lead to educational benefits as it might well provoke the student to reflect on her own understanding."

Numerous reasons have been cited as to why open learner models should be included in intelligent tutoring systems. Self touches upon a couple of these points in the above quote: first of all that having a more transparent system will place pressure on designers to be more careful about the quality of their models and of the classifications inside of them. Cook and Kay expressed similar ideas, saying that such a view enforces programmer accountability [Cook and Kay 1994].

Secondly, Self believes that the learner having a view into her own knowledge (or, rather, into the system's beliefs of her knowledge) may encourage the learner to reflect on her knowledge, and thus enhance learning. Others have voiced the same hope; in their work on the Mr. Collins system, Bull and Pain argue that inspection of and interaction with the learner model can lead to mental reflection by the learner [Bull and Pain 1995]. Morales et al describes how learners who had seen their learner models were more able to articulate their strategies than those who did not [Morales, Pain and Conlon 2001]. Potential gains may be had as well for learners inspecting domain knowledge, such as an expert model [Anderson, Boyle et al 1990, Morales, Pain and Conlon 2001].

Reflection is considered one type of metacognition [Cox 2005], which is, to put it simply, "thinking about thinking" [Flavell 1979]. More specifically, the term refers to the inspection of one's own cognitive processes or abilities. For instance, if a learner realizes that she doesn't understand long division, then this is an instance of metacognition. While it may seem trivial, we often lack a complete picture of our own knowledge. Some of the initial work on metacognition was done with children being asked to assess their own knowledge [Flavell 1979]. In this study younger children had

optimistic, and less accurate, views of their own knowledge, which then hindered attempts to improve it. It wasn't just that the younger children possessed less knowledge of the task to be completed, but that they didn't have clear notions of the limitations of that knowledge (they rated themselves as being more competent than they were), and so had a harder time improving their skills.

The exact definition of metacognition, apart from the rather vague "thinking about thinking," is somewhat difficult to nail down [defining metacognition paper]. One quote from some of the original work on the subject is helpful in drawing the line between cognitive and metacognitive progress:

> "Cognitive strategies are invoked to *make* cognitive progress, metacognitive strategies to *monitor* it." [Flavell 1979]

According to this, rereading a textbook chapter to reinforce the information is a cognitive strategy, while testing oneself over the information is a metacognitive strategy as it is intended to assess one's knowledge, though in reality many strategies fall into both categories to some degree.

Metacognition is important in part because it helps us understand our strengths and weaknesses, and these can guide us towards improving. For instance, the realization that a student doesn't understand algebra on the night before an exam is likely to encourage additional studying. Similarly, while studying, a student's realization that she understands algebra topic A well and topic B poorly will allow her to use her study time more efficiently. Outside of an educational context, being aware of what they know can help users work productively: after adding an open learner model to the sam text editor, researchers saw some users utilize it to explore new features [Cook and Kay1994]. The ability to observe and regulate of one's own cognitive processes have been shown to improve learning [Cox 2005].

In addition to the potential for metacognition, other benefits of openness include helping to clean-up and improve the learner model [Dimitrova 2003, Self 1990] and increased motivation for usage [Bull, Mangat et al 2005]. These are especially important if we consider them in the context of a collaborative system, where

determining the contributions of individuals is helpful, and increased participation in the group can improve the result.

Although researchers have identified benefits of openness, there exists the downside that providing extensive openness has the potential to interfere with the learner's productive use of the system. One potential problem is that a learner using a system is likely to have a fair amount of cognitive load from simply using the system, and piling on more information, however useful, may hinder learning and productivity. Cognitive Load Theory explores this concept, and will be described later in this chapter.

There is the additional concern that the learner may be distracted by the open interface, and spend time using it which she could have otherwise spent using the tutorial. And there is always potential for the learner to become confused by the open information, or worse, misunderstand it. For instance, a learner reading her profile the wrong way may think she is doing an excellent job on the tutorial when in fact the opposite is true. And so it does not appear to be the case that open systems exist with purely positive effects. Therefore, choices need to be made; section 2.5 on Cognitive Load Theory will provide one tool for making such choices.

2.2. Openness Outside of Educational Technology

The concept of openness is not confined to the realm of educational systems. In attempting to use any system a user must learn about the interface: for instance, knowing about its uses and limitations or how to activate certain features. These become harder when a system is especially complex. Openness can assist the user in learning about the system, much in the same way as it can in educational technology. This section describes some uses of openness outside of educational technology.

Researchers opened up a user model for the text editor sam that kept track of features used by the user and approximate level of understanding of the editor [Cook and Kay1994]. The learner model used in this work was a hierarchy of text editor features, organized into categories such as 'basics' and 'powerful' to help users browse and locate relevant features. While use of the model was completely voluntary and not everyone used it, the researchers saw that some of those who did use it took an active interest in the model, and used it to explore the functionality of the editor in more depth.

In a word processing program built on the Crystal framework, users were able to query the system regarding events that did, or did not, occur [Myers, Weitzman et al 2006]. For instance, if the formatting unexpectedly changed the user could make use of a menu system to ask why that formatting change occurred. The system under the hood has sufficient knowledge built-in to it that it is able to reason about changes that occurred and trace their cause all the way back to a user's action or setting. Such a system is a step closer to human-like interaction; if asked why a word was formatted a certain way, a knowledgeable human may reply something along the lines of "The word is part of a figure caption, and so uses the figure-caption settings that you specified." This idea of being able to query the system's inner-workings is very much along the lines of the ideas of openness.

Similar in character is the Whyline, a tool to help programmers debug their applications by allowing them to ask questions of the output produced by the program [Ko and Myers 2004, Ko 2006]. Open systems such as Crystal and Whyline can be seen as showing users what is actually happening at the core of the system itself, which can help the user solve the current problem while also potentially help the user learn about the process of debugging.

Related to the question of what to make open is the question of how to present data in general; this falls under the broad research area of information visualization [Herman 2000]. Many types of data can usefully be represented as graphs, that is, as collections of vertices and edges between those vertices. Converting this data from a pair of sets to a visual presentation allows those viewing it to make use of the sophisticated visual abilities of humans, and so allowing us to more easily make sense of the data and find patterns. This, however, leads to one of the main problems in information visualization, which is dealing with very large sets of data, given the challenges of showing it in limited screen real-estate and navigating it.

One of the most common representations for graph data is that of the tree layout, for graphs that satisfy the mathematical definition of a tree. Such trees are used commonly for everything from family trees to object-oriented programming designs, and give a simple and effective way to represent hierarchies of data. The tree layout is the

representation of problem-solving in this work, in the CoSolve system, as it precisely describes how work builds off of the work that came before it.

As described in sections 2.1 and 2.2, openness has been added to both educational and non-educational systems and has had a number of effects shown by researchers; these range from motivational effects to more productive work. Now we'll explore the problem domain to which I have added openness, that of collaborative problem-solving.

2.3. Collaborative Problem-Solving

For the purpose of this work I consider problem-solving to mean acquiring a satisfactory solution to a formally stated problem. What makes a solution satisfactory is dependent on the problem; for some there will be one correct answer (solving an algebra problem); for others it may be matter of trying to maximize some function, such as the amount of money earned. In a computational sense, the input to the system is the problem (including its complete rules, initial state and goal), and the output is a solution that satisfies that goal. The addition of the term 'collaborative' means that a team of solvers will work together to find a common solution; this is in contrast to solvers working independently or competitively on a problem. Modern computing technology has the potential to enhance collaborative problem-solving in several ways by connecting people remotely and by providing additional affordances for collaboration and solving.

A related area is that of crowd-sourcing, where solvers are given small pieces of the problem to solve, and these are combined together to create a full solution; often times solvers have economic incentives to complete tasks [Brabham 2008]. In contrast, collaborative problem-solving generally involves different solvers working together instead of on their independent pieces; solvers' awareness of each other is often important for the completion of a collaborative task.

Collaborative problem-solving has a great deal of potential due to how computer systems can connect people (with potentially diverse skills and knowledge) across the world, tapping into a vast resource pool that would likely be impractical using other technologies. There is also the opportunity for additional features, such as having the

computer make suggestions or check work. These capabilities can, however, introduce additional problems, as questions arise regarding how to manage and regulate a large project.

Researchers have taken many approaches towards creating computer programs that help human users solve problems. Fischer describes one way to think of the space of programs [Fischer 1990]. One example of a point in this space is that of an expert system, where the computer has the primary responsibility and power to solve the problem, and the user merely needs to ensure it is set up correctly and then start the process. This, while potentially useful for many problems, does not take advantage of the fact that a human user has many skills a computer does not, such as general image classification and abstract notions such as creativity. Ideally, a problem-solving system would take advantage of the abilities of both the human and computer solvers.

In the same work Fischer describes a theoretical framework for 'cooperative problem solving systems.' Here it should be noted that 'cooperative' indicates cooperation between a human and a computer, not cooperation between several humans through a computer system. One of the key ideas described is that the human user will almost certainly have a mental model of the problem that is different from how the computer represents and understands the problem: one of the key tasks of a cooperative problem-solving system is then to help translate the user's ideas and intentions correctly to the computer. Fischer describes several ways that this translation can be improved; the most relevant here are the ideas of 'restructuring' and 'reformulation.' In restructuring, the tool designers modify the tool's representation and understanding of the problem to be solved in order to more closely match that of the user, and therefore make it easier for users to convey their ideas. Reformulation is in some senses the opposite of restructuring; in reformulation, the tool is outfitted with functionality to help the user understand how the tool represents the program. A transparent into the tool's representation could be used as part of reformulation, for instance. Both restructuring and reformulation serve the same purpose, easing the translation of ideas from user to system, but each goes about it in a different way.

The work of Fischer mentioned above deals with human-computer interaction for problem-solving, but does not take into account collaboration by several humans using

the tool together. By considering ideas of collaboration with Fischer's notions of cooperative problem-solving we reach the area of computer supported collaborative work (CSCW). CSCW research also seeks to utilize computers to enhance the problem-solving capabilities of humans, but has emphasis on using computers to mediate solving between human solvers.

CSCW systems are often categorized by the dimensions of place and time. A collaborative effort can occur with its participants inhabiting the same place or with them spread out and interacting remotely. Similarly, participants can be working together at the same time (synchronous) or at different times (asynchronous). Collaboration often occurs with participants working together in the same location and at the same time, but technological advances can help remove these restrictions. For an example of asynchronous and remote collaboration, consider people working together on an online message board to solve some problem: the solvers can be located anywhere in the world and still work together, and need not be online at the same time.

Such a categorization scheme, though useful, only describes two high level (though important) aspects of CSCW systems, and as such, more in-depth taxonomies have been proposed [Penichet 2007]. Additional dimensions suggested include those of coordination, communication and sharing information. Features for coordination help solvers organize their work, which may improve efficiency. Features of communication support solvers speaking to each other, perhaps via written word. Features for sharing information allow solvers to pass around documents and products; this is similar to communication, but concerns sharing artifacts of the collaboration instead of messages. These dimensions allow us to more precisely categorize CSCW systems, and will be used to help describe CoSolve, the collaborative problem-solving system on which I have primarily built my work.

CoSolve is unique as a collaborative problem-solving system in that shows solvers their entire design process as a design tree, and allows solvers to remotely and asynchronously contribute to any portion of the design history, past or present [Fan, Robison and Tanimoto 2012]. According to the above taxonomy, CoSolve possesses affordances for communication, coordination and information sharing. CoSolve is

described in more depth in chapter 4, and this categorization will be explained in more detail.

Collaborative problem-solving systems offer a number of advantages over face-to-face collaborations: they can take place remotely, asynchronously, and can offer a number of computational features, such making suggestions. Yet at the same time, there are still a number of difficulties to using collaborative problem-solving systems, including the following, both of which I have addressed by adding openness to the system:

- Difficulty in understanding contributions: correctly attributing work to particular group members is important so that, say, recognition can be given for good work, but understanding who has done what may be difficult to communicate to solvers, even if it is fully recorded in the system
- Difficulty in keeping solvers up-to-date with the current state of the system; this is especially problematic in the case of the asynchronous collaboration

My work focuses less on the design of collaborative problem-solving systems, and more on adding openness to an existing collaborative problem-solving system, and how openness can contribute to solving these sorts of problems. We will now examine some existing work of researchers into adding openness to collaborative problem-solving systems.

Researchers have looked at adding a number of features of openness to collaborative problem-solving systems. Engelmann and Hesse looked at the effects of showing representations of team members' knowledge to each other [Engelmann and Hesse 2010]. As collaborative efforts often combine talents from different areas of expertise, and the team members may not be well acquainted with their individual specializations, there is a need to convey what each team member knows to the others in order for them to more effectively coordinate and solve. Engelmann and Hesse approach this by showing team members knowledge maps of themselves and their team-mates, effectively letting the team see the knowledge possessed by its members. Their work has shown that sharing this information with team members can speed up the beginning time for the problem-solving activity, as there is no longer a need to probe

each other for their respective expertise. They also found that the sharing of team knowledge improved the correctness of the team solutions.

Their work provides evidence that openness between team members can enhance the collaborative process. The work does not, however, look at conveying information beyond the knowledge team members bring to the session; the information shared is static throughout the session. A question of greater interest in this work is the effect of sharing information about the current problem-solving session: what each team member has done and contributed.

Janssen et al have examined issues more directly related to this idea of opening up the information regarding a problem-solving session to its team members [Janssen, Erkens et al 2006]. Their work has students working together to answer questions on the subject of historical witchcraft by referring to a number of historical documents and communicating with each other via a chat interface. The experimental group was additionally given the Participation Tool, which inspects the text communications of group members and graphically displays them (as well as those of other teams in the class) to convey the frequency and length of messages sent by team members. After some initial training, team members were able to, at a glance, get an impression of how they have communicated with their team members compared to the behavior of their team members and of the rest of the class. The visualizations created did result in the experimental group participating more, though it did not improve the quality of answers or improve knowledge awareness. Communications were also tagged based on their role in the collaboration, and the experimental group was shown to have made more communications aimed at social coordination than the control group.

The work of Janssen et al is intriguing as it shows team members live collaboration information and reports increased communication as a result. The information conveyed, however, is extremely simple (number and length of messages sent as well as content of messages and some basic tagging), and only scratches the surface of collaborative metrics: what about the degree to which they built on one another's work, and what portion of the solution is due to each team member? There is more going on than is captured by these low level metrics, and ideally the openness could show some of it.

The work of Balakrishnan et al takes a different approach to openness in collaborative problem-solving [Balakrishnan, Fussell and Kiesler 2008]. In their study, two team members worked together in the role of detectives to solve a series of murders. In these rules they had access to information on victims and suspects, including transportation patterns and occupation, and details on the crimes committed. The control group had access to spreadsheets containing relevant information, while the three experimental groups had visualization tools showing the same information. The experimental groups differed in the degree of openness of the visualization tools, from none (each team member maintained a separate visualization), to read-only views of partners' visualizations, to a visualization fully shared between team members. The experimental version using the full shared visualization had the best results, with better performance than the other groups. Interestingly, providing read-only peer visualizations actually resulted in worse performance than entirely unshared visualizations.

While providing additional evidence for the power of opening up the collaborative process, Balakrishnan et al's work is limited in that the problem being solved has a single predetermined solution; similar to educational practices, but less relevant for more general problem-solving where the solution is actually unknown. Furthermore, the information shared is fairly domain dependent, and not a series of general values common to any team of solvers.

The work of Jermann and Dillenbourge [Jermann and Dillenbourg 2007] looks at not only showing live collaboration information to solvers, but the effect of context in terms of what is desirable behavior. By supplying this context, they saw increased participation. While interesting, these too are fairly simple metrics.

My work addresses the problems of understanding contributions and keeping solvers up to date by showing teams live, up-to-date information on their activities, not only of simplistic low-level metrics but of potentially more useful metrics of which the solvers may not have even been aware. Finally, the openness in this work is designed to be applied to a range of different problems; it is not constrained to one or two domains.

2.4. Metrics for Collaborative Problem-Solving

One of the most intriguing aspects of openness is the idea of opening up the solver's model to the solver (and the solver's team); very much along the lines of open learner model research, discussed previously. The question is what type of metrics we should compute and display to solvers; the work described above focuses on simple metrics like the number of messages sent, and on domain specific information. The nature of these metrics will of course vary depending on the type of collaboration and problem-solving in the system. Researchers have suggested a number of metrics for collaboration, and these will be described here.

One useful source is the work of Collazos et al, who compiled a list of collaborative metrics [Collazos, Guerrero et al 2007]. Their metrics are based on a series of low level values obtained from text messages via tagging schemes, marking them as discussing strategy, coordinating, etc. The metrics listed are primarily low level, being simple functions of hand-tagged values. They focus on the extent to which solvers discuss strategy, coordination, etc. Because these values were hand-tagged, they weren't suitable to automatic computation and being displayed to solvers, but similar ideas were used to analyze solver behavior in CoSolve, which will be discussed in chapter 5.

Barron examined the nature of discussions of teams of three solvers working on math word problems [Barron 2003]. Upon dividing teams into 'successful' and 'unsuccessful' groups, Barron saw two interesting results. First, when faced with a correct proposal for how to solve the problem, successful groups were more likely to accept or discuss it, while unsuccessful groups were more likely to ignore or reject it. That is, it wasn't that successful groups simply suggested more correct answers, but that they were more willing to consider those answers when they were presented. Secondly, proposals in more successful groups tended to follow from the previous discussion, whereas in less successful groups they were more out of synch with the discussion. Unfortunately, these results were obtained by coding messages by hand, which makes them less suitable to be computed automatically and shown to solvers in real time. Nonetheless, metrics in the same vein were found that could be computed automatically; these will be described in chapters 4 & 5.

The work of these researchers serves as a useful beginning for understanding solvers as they work in collaborative problem-solving environments. Part of my work builds on these to improve and expand measures of how solvers work and collaborate.

2.5. Cognitive Load Theory

In adding openness to a system, there are a number of decisions that need to be made, such as how to make the tradeoff between the benefits of showing more information with the downside that it could be distracting or confusing. Cognitive Load Theory is used in this work as a tool for addressing these types of challenges [Sweller 1988, Artino 2008, Kalyuga 2007, Kalyuga, Ayres et al 2003, Mayer and Moreno 2003, Moreno 2004].

Cognitive Load Theory describes the human mind as a machine, not unlike a modern day computer, that has limited resources for storing and processing information in working memory, but arbitrarily vast storage space for items in long-term memory [Artino 2008]. Most useful reasoning and learning takes place in the working memory, but because of its limited size our minds can easily reach cognitive overload; a state in which learning is severely impaired due to insufficient cognitive resources. In computer terms, working memory is analogous to a computer's processor cache; things 'on its mind' are stored there, and the computer operates upon data found in that memory (conceptually; in reality the details of a computer's processing are a bit more complicated).

In this theory a person's working memory's size is greatly constrained, and so in order to accommodate the enormous amount of information we are confronted with, we as reasoning beings require more sizable storage; that's where long-term memory comes in. Long-term memory is the storage space for things we may potentially need later, and so can safely be set aside until they are later needed, at which time they can be pulled back into working memory to be used. In our computer analogy, long-term memory is like a computer's hard-drive, incredibly spacious, but in order to do anything useful with the information you need to pull it into working memory.

So far this model of our cognition is pretty much as expected; people have realized for a long time that the conscious mind does much of the interesting work, and

that we possess some kind of long-term storage which can be written to and read from. It is here that CLT begins to expand further and starts to become interesting. Given our limited working memory, the question of how we perform complex, multi-step actions comes to mind. For instance, reading this text: experienced readers effortlessly scan the symbols on this page, reading them into letters, bringing those letters together into words, then bringing out meaning from this combination of words. For experienced readers, most steps in this process do not require any conscious thought; one does not consciously reason that the first letter in this sentence is an 'F', nor that the meaning of the string of characters 'For'. This fairly substantial body of knowledge (character recognition, spelling, grammar, parsing semantic meaning, etc.) is condensed more compactly in the mind as we become better at it, and eventually no longer requires our conscious attention.

Cognitive Load Theory addresses this building from of prior knowledge via the concept of the schema [Artino 2008]. Loosely, a schema is an integration of several items of information together into a new piece of information; the resulting association is stored in long-term memory. Once learned, the schema can be applied automatically, without conscious thought, and counts only as one 'item' in working memory, despite how many items it consists of. Therefore, repeated construction of schemas is one way around the limits of working memory; we can't change the size of our work space, so we instead condense the size of the items we work with. In addition, after being learned, a schema is itself an item of information that can be combined into other schemas; a curve becomes a 'C', then a series of letters becomes the word 'Cognitive', which produces its own complex meaning. According to CLT, this is how we learn, by compacting frequently encountered and important trends into schemas, allowing us to more easily hold, recognize and reason about these trends.

The question then arises of how to construct schemas. CLT holds that our working memory can be put towards learning these schemas, but that same memory resource is also required for reasoning about the task and anything else that's 'on your mind.' These different uses of working memory in performing a task are described below:

- Intrinsic cognitive load: Intrinsic cognitive load is the burden on our mental resources of the material being learned; the size of this load depends on the complexity of the idea being learned and reasoned about. Conceptually, the only way to reduce the intrinsic load of a task is to learn schemas about it, and use those instead; other than that, you can't get around it.
- Extraneous cognitive load: Also called ineffective cognitive load, extraneous load is a burden placed upon one's cognitive resources that isn't part of the material itself (that is, isn't part of the intrinsic load), and doesn't contribute to learning schemas. In essence, extraneous cognitive load is just a waste of mental resources that could be used reasoning about or learning the material. An example of extraneous cognitive load would be trying to read some facts from a textbook when the textbook page is too 'busy': containing unnecessary images and text. The way material is presented (straight-forward versus convoluted) can decrease or increase the extraneous load of a task.
- Germane cognitive load: Germane cognitive load is the use of cognitive resources to build schemas. Over time, building schemas will reduce the intrinsic load of the task – reducing the overall load of the task, but first resources need to go to germane cognitive load to build the information into a schema.

It is important to note that these three types of cognitive load all share the same resources; a very 'busy' presentation of some information will likely result in additional extraneous load, which results in fewer resources to be used for understanding the material (intrinsic load) and building schemas for it (germane load), and since the intrinsic load has a set value (until schemas are learned), that means fewer resources for germane load, and thus slower learning. The theory thus suggests that reducing extraneous load is key to allowing resources for germane load, and thus for learning of schemas to take place. The notion of cognitive overload is when 'too much is going on' mentally, perhaps due to extraneous information or lack of schemas, and so the learner is unable to spare the resources for learning the material.

Some of the original work on Cognitive Load Theory [Sweller 1988] focused on the learning differences between domain novices trying to solve a problem with a specific goal ('How far will the ball roll off the ramp, given friction, gravity, etc.') versus

those with a non-specific goal ('Solve for as many variables in this situation as you can'). Sweller observed that the non-specific goal group actually learned more, and so he hypothesized that the search process used by the specific goal group to reach their goal was actually imposing cognitive load sufficient to hinder the learning experience. A comparison of domain novices and experts indicated to Sweller that, while novices trying to reach a particular goal would try to search backwards from the goal (in a process similar to state-space-search), experts used the more direct method of going forward, choosing the next operation based on schema developed through previous experience.

One implication of these initial thoughts was that it may be better for novices to learn material through a series of worked problems rather than simply being given problems. The idea is that, for a novice learner who lacks schemas for the domain, simply giving them problems can be cognitively overwhelming; worked-problems provide sufficient scaffolding, allowing for more cognitive resources to be devoted to germane load.

A more recent finding in CLT research is the notion that learning techniques that work well for expert learners, such as being asked to mentally explore the material, may not work well for novices. Likewise, techniques that work well for novices, such as heavy scaffolding of the problem, may not work well for experts; indeed, some work has shown that experts may actually be hindered by such scaffolding. Researchers have termed this idea 'expertise reversal,' and have explored it in the context of CLT [Kalyuga, Ayres et al 2003].

For the novice attempting a new problem without much guidance, the great number of available options can cause cognitive overload, whereas the expert already has a suite of schemas built up to better handle it. When the novice attempts a scaffolded problem, the scaffolding can remove much of the cognitive burden, but for the expert attempting the same problem the large amount of redundant information in the scaffolding can actually cause extraneous cognitive load, hindering the reasoning process; as counter-intuitive as this is, researchers have found some experimental evidence to back it up.

2.6. The Cognitive Dimensions Framework

In order to help evaluate the design choices made for my addition to CoSolve I will be using the Cognitive Dimensions framework [Green and Blackwell 1998]. The Cognitive Dimensions framework provides a way to evaluate the usability of 'information-based artefacts,' such as word processors, programming languages, graphics packages, and interfaces for devices such as telephones and DVD-players. The framework provides a 'broad-brush' treatment rather than a detailed evaluation, and so gives fairly qualitative and high level suggestions. The framework was selected due to its broad nature, and due to the fact that it is designed to be both easy to use and understand.

The framework is intended for use by people who are not usability experts, and consists of roughly 13 dimensions that can be used to examine aspects of usability of the system (I say roughly, as the details vary from version to version of the framework). In addition to the dimensions, the framework describes the relationship between dimensions, how, for instance, increasing the dimension of 'abstraction usage' can increase the 'viscosity' of the system. The framework also describes design maneuvers through which users can improve the systems regarding one of the dimensions, though potentially at the cost of harming it with regards to other dimensions. These maneuvers are less relevant here, as the framework is used here for summative evaluation, and so will not be presented.

In the interests of space, only a subset of the framework is presented here; of the 13 dimensions, I will consider only those most relevant to this discussion of adding openness to CoSolve. The framework, being usable for a wide variety of systems, contains a number of dimensions that are less relevant to this work; some examples of these include dimensions looking at how close the interface matches the domain it represents ('closeness of mapping'), and the degree to which the interface causes one's work to become resist to major changes ('viscosity').

Some other dimensions, such as that of 'Secondary Notation' are relevant to CoSolve, but are left unchanged by the addition of open features, and so will not be covered, as we are interested here in the contrast between CoSolve with and without

the Consultant. The Cognitive Dimensions framework is covered in much greater detail in literature [Green and Blackwell 1998].

The four dimensions most relevant to this work are presented below:

Visibility and Juxtaposability: These seem like two strongly related but distinct concepts, yet the referenced version of the Cognitive Dimensions framework lists them together as a single dimension, and so they will be treated here as one.

Visibility is being able to 'view components easily,' but also extends to being able to search for information. A physical telephone directory makes it easy to search by name to get a number, but not to search by number to get a name, resulting in lower visibility.

Juxtaposability is the readiness with which you can compare different components in the system; it is somewhat awkward to conduct a side-by-side comparison of two charts on different pages of the same book for instance, but the same comparison may be easy if we could open each chart in a separate browser window on a computer and set those windows side-by-side. In the example of charts on different pages of a book, one may be forced to flip between the two charts, temporarily holding the other in memory, resulting in higher cognitive load.

Abstraction: Abstractions are, like the name implies, macros, aggregates or short-hand versions of other operators or symbols bound together. Once learned, abstractions can result in more concise notation and can represent important combinations of ideas. Some systems allow user created abstractions, such as defining a function in a programming language; these can be helpful to the creator, but may present the danger of confusing others using the system (and possibly the creator herself), especially if poorly designed. For experts, abstractions can improve efficiency, but for novices, however, abstractions can sometimes pose a barrier to use the system and can easily be misunderstood and misused. On the other hand, sometimes thinking in terms of meaningful abstractions can make it easier for a novice to understand a complex system; it depends on the system and on the abstractions used.

Another aspect of abstractions is bringing the representation of the system more in line with the user's understanding of it. That is, abstractions can group together confusing details into higher level concepts that are easier for the user to understand. One example of this is representing a student in a programming language. By grouping together numerous variables and functions into a common class called 'StudentProfile,' the interface for the programmer can be made somewhat cleaner than it would be otherwise.

Hard Mental Operations: Of particular interest here are mental operations that are hard because they require mental navigation of some complex space, such as a maze, where one has to keep track of where one has been and where one is going in order to make progress. Navigating menu systems and directory hierarchies also fall into this category. Such operations are undesirable because of the extraneous cognitive load they carry.

Progressive Evaluation: This dimension describes the kind and degree of feedback users can get from their partially completed works. A system in which it is easy to check partial solutions has a high level of progressive evaluation, and those in which a complete solution must be attained before feedback can be given have lower progressive evaluation. Consider an extremely modular program consisting of many independent pieces that can easily be tested alone; this would have a fairly high progressive evaluation. Now consider one program written as a single giant function; in this case it is much harder to verify that different conceptual parts work until the entire program is finished.

It should be noted that the individual dimensions are not necessarily all positive or all negative. Systems with high abstraction or low abstraction, for instance, have both advantages and disadvantages.

In chapter 4 these aspects of the Cognitive Dimensions framework will be used to evaluate the design of the CoSolve Consultant.

This chapter has described various aspects of work relevant to the openness in collaborative problem-solving systems. Chapter 3 will go into my earlier work with the theme of openness. Many of the ideas and lessons from this earlier work guided my recent work in adding openness to the CoSolve system.

Chapter 3: My Earlier Work in the Area of Openness

Openness has been a consistent theme throughout my work, and many of the lessons learned from my earlier work have guided my work in adding openness to CoSolve. This chapter describes the portions of my earlier work that led to my contributions to CoSolve.

3.1. The Data Factory, an Open Visual Programming Language

My work in openness began with my extension of an open visual programming language called the Data Factory [Tanimoto 2003, Robison 2005]. One key idea of the data factory is that the program is shown in great detail to the user, both in its static and dynamic (run-time) states, and is so an example of a very open system. The Data Factory is intended for research and educational purposes, as opposed to purposes of professional software production.

3.1.1. The Original Data Factory

In the original version of the Data Factory, a program is written as a collection of functional components laid out on a 2d grid and connected by 'conveyor belts,' similar in some respects to circuit diagrams. Numbers, the data of the system, travel along these conveyors and into the functional components, which then perform operations on them such as adding two inputs and outputting the result. The components are hard-coded in the system, and include arithmetic operations as well as various control operations. The resulting language allows one to write programs for various computations, such as computing prime numbers. A screenshot of a simple program in the Data Factory can be seen in figure 3.1.

One interesting feature of the Data Factory is that the same interface is used for writing and executing the program, unlike most programming languages where the static code is written, and then separately compiled and executed. As a result, the Data Factory shows the complete state of the program as it executes in real-time; a strong example of openness.

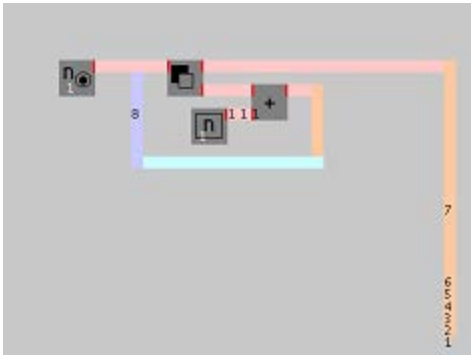


Figure 3.1: A simple program in the Data Factory that counts integers starting at 1. The grey squares are the functional components, which take inputs (if any) on the left and produce outputs (if any) on the right. The colored lines between components are the conveyor belts which transfer data from component to component (the different colors indicate the direction of conveyance). The numbers are the individual items of data undergoing computation.

One thing that the Data Factory lacked was the notion of an array storing key/value pairs. My contribution to the original Data Factory was to add such an array in a way that was consistent with the extensive openness of the system. The constraint of extensive openness in finite screen space seems to be at odds with the notion of an unbounded array, and one of the main difficulties was figuring out how to compromise and satisfy both. My solution to this problem was to include an unbounded array, but have the display show the item most recently accessed (read from or written to), as that should be the most 'relevant' state. The keys are values used by the array are integers, as are all items of data in the Data Factory.

The neighbors of the recent element are also shown, given the importance of spatial proximity in many programs, though they are drawn smaller to suggest being further away and less relevant than the indexed item. A screenshot of this updated version of the Data Factory is shown in figure 3.2. Upon receiving a different index, the display would smoothly animate to showing the new index, visually stepping through the indices in-between to indicate the transition.

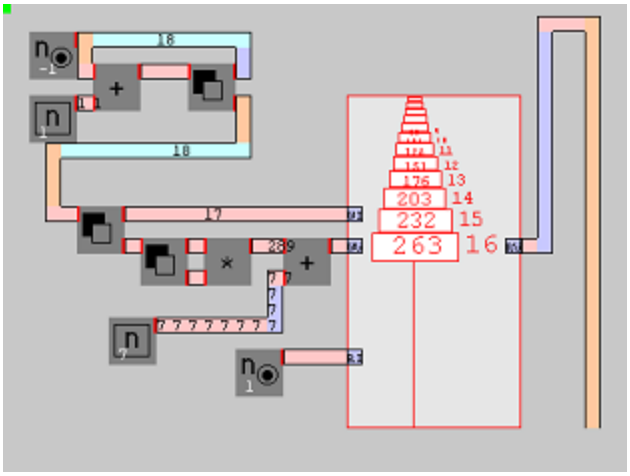


Figure 3.2: A Data Factory program that computes a function on integers and writes them into an array. The array visualization shows the value in the red box and its corresponding index to the right. When it receives another index as input, an animation transitions the view to the new index. The function being computed is n^2+7.

3.1.2. Adding Functional Abstraction to the Data Factory

The original Data Factory is an intriguing example of openness, but is somewhat limited in the complexity of what it can represent. In order to further explore this idea of extensive openness I created a new version of the Data Factory, simply called Data Factory 2, that supported a number of other features, foremost among them functional abstraction and recursion[Robison 2006]. One of the key difficulties was determining how to handle the visualization of functional abstraction and recursion in limited screen space, and maintaining the 2d grid metaphor.

My solution to this problem was to show functional components as boxes that could then be opened up so that the user could examine their internals. In order to facilitate this, the internals of the component were drawn smaller and in the same space, and the user was allowed to zoom in if they so desired. A screenshot of this work is shown in figure 3.3.

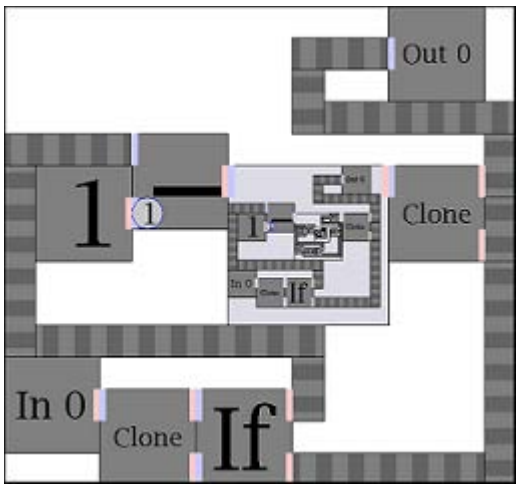


Figure 3.3: Example of the definition of a functional component in the second version of the Data Factory. 'In' components are the arguments to the function and 'Out' components are returned values of the function. The large box in the middle is a recursive call to this function; the user has clicked on it to reveal its inner workings, a miniature version of this component. There is no data being processed in the screenshot, but if there were, the different levels of the recursive call would be shown to maintain their own data. This function takes an input n and outputs 2^n copies of the value 0.

The Data Factory, in both of its versions, is a novel demonstration of extensive openness in a visual programming language. One problem with it from a practical perspective was that complete openness, while an intriguing idea, is often not desirable due to the display complexity it may require. Even Data Factory programs of moderate size could easily become complex to the point that their own creator could get lost inspecting them. Providing tools to alleviate the complexity, or reducing the complexity shown, is then desirable; this idea was explored in my work on openness in tutorial systems, for which I built a framework for running tutorial programs.

3.2. INFACT and the Interactive Window Framework

Given my experiences with openness in both versions of the Data Factory, I was interested in exploring the ideas of openness more in the context of the educational environment INFACT. INFACT is an online educational environment with a forum for class discussion and tools for assessment, including conducting tests [Tanimoto, Carlson and Husted 2002]. It also hosts educational tools, such as the image processing program PixelMath [Tanimoto 2006]. Built into INFACT is a software platform known as the Gnome system, which allows developers to write arbitrary agents that run on the server and can observe student interactions and store information in the INFACT database, in large part for assessment [Benson, 2004]. The Gnome system is a powerful platform for writing agents that can run on INFACT, monitor student actions and make use of INFACT's resources, and so it seems like an ideal platform on which to create a tutorial, but there was unfortunately no direct means for a Gnome agent to interact with a student, making it very difficult to write a tutorial using the Gnome system.

To address this I wrote the Interactive Window system that provides a client-side interface for the student to interact with INFACT. To allow complete versatility in what interactions the student could be offered, the Interactive Window system, or IWindow system, is capable of loading remote Java classes. As a result, developers can write their own Java tutorials for interaction (for instance, a windowed system for showing graphics and accepting mouse or text input) and easily plug them in. The tutorial content can then be controlled from an agent on the Gnome system, allowing it to talk to the rest of INFACT. Using the IWindow framework I created some simple tutorials, including one to teach users how to create simple programs in the Data Factory to demonstrate flexibility of the system. The details of the IWindow system are entirely technical, and so will not be presented here.

3.3. Open Tutorials

With IWindow infrastructure in place onto which tutorials could be built, I was then able to proceed in creating open tutorials for INFACT. As described earlier in chapter 2, researchers have explored various aspects of openness in intelligent tutoring systems. One area that struck me as less developed than the rest was the idea of opening up the tutorial process. That is, many researchers have looked at opening up many aspects of the learner and peer models, but solvers are usually kept in the dark regarding the inner-workings of the tutorial itself. My work shifted to looking at showing aspects of this process to learners, so that they could learn about how they are being assessed and guided in the instruction.

Towards this goal I created two tutorials using the IWindow framework and ran exploratory user studies looking at the effects of both. The first study had six learners using a computer tutorial on the topic of resolution in propositional logic, and examined the effects of showing the learners the internals of their learner models and of their progress through the tutorial. There were open and non-open versions of the tutorial, which were mostly identical, with the only difference being that in the open version learners could see details regarding their knowledge and their progress through the tutorial; the non-open version offered only high level information in the form of score, skill and progress meters. The study had a within subjects design, and so all subjects had access to both the open and non-open versions during the first two phases of questions, though the order in which they were received was randomized. For the third and final phase of questions the learners were allowed to choose either the open or non-open version of the tutor.

This first study suggested that learners do appreciate being given an understanding of the internals of the tutorial on some level, though it was not seen to affect performance of answering questions. I also saw that, given a choice, five out of six learners chose to use the open version instead of the non-open version. Yet there were questions that this study left unanswered: how would this work with views into more complex processes, and what is the impact of the open view on learner trust? To examine these I ran a second study.

For this second study I created a new tutorial in the IWindow framework. This one also taught the topic of resolution in propositional logic, but this one was more complex than a simple series of questions, as the first one had been. In this tutorial, the learner's goal was to sufficiently demonstrate mastery to the tutor; towards this end the learner was given instructional materials and feedback on his work, but was also given free rein on what problem to handle next by selecting a topic and difficulty. The learner was shown his progress in each of the three topics covered, and when he achieved mastery in each of the areas, the tutorial was complete

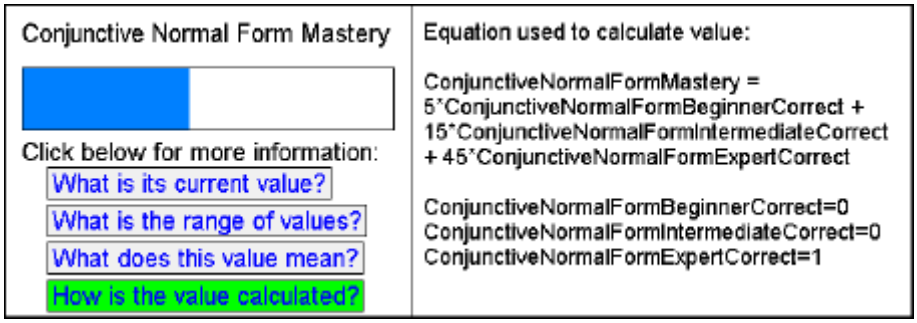


Figure 3.4: A portion of a screenshot shown to the learner in the second open tutorial I created. This shows the learner one item in the learner model and how it was calculated.

This tutorial also had open and non-open versions, though unlike the previous study, this one was run between subjects. Learners in the control condition were told that the difficulty of problems attempted would influence their progress, but not told the details of how, as is the case in many non-open systems. Learners with the open version of the client were additionally able to open windows showing details on their knowledge profile and how the tutorial computed those values. In theory, learners could use the information regarding how these values were calculated in order to more efficiently complete the tutorial.

A total of nine participants took part in the study; four in the control condition and five in the experimental condition. Data was gathered from the learners' actions during the task, and from a questionnaire and brief interview following the task. Those using the open version rated it as being a better experience, this result was not statistically significant. What was significant was that learners with the open version reported greater trust in the tutorial. Learners in both condition groups took similar amounts of time to complete the tutorial.

At the end of the experiment each learner was briefly shown the alternative version of the software (open or non-open), and were asked which they would prefer. Those seeing the open version for the first time had it described to them as "showing more information, but is somewhat more cluttered and complex," whereas those seeing the non-open version for the first time had it described as "a more stream-lined one with a less cluttered view." This wording was chosen to given verbal preference to the non-open version. Nonetheless, six out of the nine chose the open version, while two were neutral or preferred a hybrid version (one from the control and one from the experimental) and one from the control condition preferred the non-open version.

This second study showed some evidence that openness can improve trust in the tutorial, and that learners tend to favor it. In retrospect, the study focused too much on quantitative metrics, which are less meaningful for a small pool of participants. In my more recent study, described in chapter 4, more attention was given to qualitative data found seen in solver behavior and interviews.

To further examine this second tutorial, I will now evaluate it according to the Cognitive Dimensions framework. The evaluation will focus on the differences in cognitive dimensions between the open and non-open versions of the tutorial.

Visibility and Juxtaposability: One of the primary gains offered by the open version is that of enhanced visibility, as the open version freely offers up information on the internals of the tutorial that would be difficult for the learner to figure out otherwise; the same information could be gained by a trial, error and observation of the learner model, but this would not be simple to do. The open view provides this information

directly, however. The juxtaposability is left largely unchanged, as the interface does not include additional functionality for comparing information regarding the learner's model or the tutorial's structure. Given the simplicity of the system, this limited juxtaposability does not seem to be an issue. One could imagine it being more important if the learner model became much larger and more complex, as the comparison of different components could be useful in trying to understand how it all works.

Abstraction: The level of abstractions in the tutorial is left largely unchanged by the addition of openness. Some minor abstractions existed without the openness, primarily the learner model consisting of the learner's mastery defined in terms of mastery of three areas of problems. The open view simply reveals this structure, showing how the abstraction is defined, but does not increase or reduce the abstractions.

Hard Mental Operations: The level of hard mental operations is increased due to the addition of openness, as the optional task of attempting to make sense of the open information requires navigating a fairly shallow menu-system, examining descriptions and equations, and mentally mapping those to one's performance in the tutorial. The increase is not great, due to the simplicity of the system. The hard mental operations introduced are then not mandatory for using the tutorial, but are necessary to actually make use of the openness and corresponding benefits to visibility and progressive evaluation.

Progressive Evaluation: This dimension improves with the addition of openness, as the basic information shown to the learner in the non-open version lacked context for evaluation. By giving the learner a view into the assessment process, it becomes easier for the learner to gauge his progress. A relevant analogy would be seeing how far one had traveled, but also seeing a map of the route taken so that that distance can be seen in relevant context.

The openness included in the second tutorial offers gains in visibility and progressive evaluation, with a likely minor increase in hard mental operations. One way to improve the progressive evaluation further, providing more context with which the learner can understand his progress, would be the addition of peer views, perhaps similar to those described in chapter 2. While seeing one's standing in the tutor's eyes is helpful, seeing social comparisons can provide a different, and potentially useful, perspective. This avenue is one of those explored in the next chapter describing the CoSolve Consultant, where information for social comparisons is made available to solvers in CoSolve.

3.4. An Engine for the PRIME Designer, the Predecessor to CoSolve

My introduction to CoSolve came about as a result of working with the PRIME Designer, a collaborative design tool whose ideas later led to CoSolve [Tanimoto, Robison and Fan 2009]. PRIME stands for "Puzzle Rooms with Image and Music Experiences." The PRIME Designer allows teams of four designers to collaborate in the creation of 'PRIME games.' PRIME games are described as follows:

> "Games in this format are limited to 9 square (or cubical) rooms in a 3 by 3 layout. Any adjacent pair of rooms may have a door between them. Each room may have background music that starts playing when the player enters the room. Each wall may have wallpaper (given as an image file) as well as an image puzzle or a music puzzle. When a player solves a puzzle, credit is awarded in the form of a "magic phrase" which can be uttered to perform actions such as opening a door or gaining points."
>
> [Tanimoto, Robison and Fan 2009]

The four team-members each take on a different role in the creation of the game: the architect decides the flow and aesthetics of the rooms and places puzzles; the image puzzle designer creates puzzles that are based around restoring fragmented

images; the music puzzle designer creates puzzles that are based around unscrambling a scrambled series of notes; and the logic designer decides rules of the game, such as how the solving of puzzles links to progress in the game through magic phrases and awarding points.

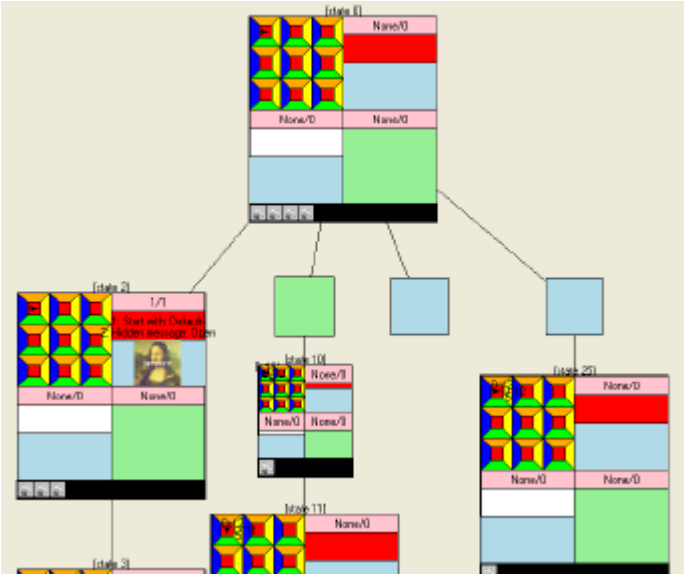


Figure 3.5: A screenshot of the PRIME Designer, an environment for teams of four designers to collaborate in creating a PRIME game. The design history is shown to designers as a state-space search tree, with the root of the tree, which indicates the initial state, being shown on top. In this screenshot, each state shows a view for each of the four designers in its four quadrants.

Of greater importance here is the manner in which the team of designers worked together to create the PRIME game, as these ideas later led to the creation of CoSolve. In the PRIME Designer, the team starts with a functional, but extremely simple, game: it is simply the three by three grid of rooms, but without decoration, puzzles or connecting doors. In short, it is a valid, but trivial, game. Designers can modify this by applying operators to this initial state; instead of overwriting this initial state, these changes result in a new state being created and shown below the old one. Designers can, at any point, ‘back up’ to earlier work or jump over to work performed by their team-mates.

The design history of the team is shown to them as a tree, with the root of the tree being the initial trivial state and being shown at the top of the display. Designers can work on any part of the tree they want, allowing them to move around the design space. This is essentially an open view of the design history. A screenshot of the PRIME Designer is shown in figure 3.4.

My contribution to the PRIME Designer project, apart from having users collaborate using the system and making observations of their behavior, was to create a client through which designers can play the PRIME games they created [Tanimoto, Robison and Fan 2009]. Two game engines already existed: one was text-based and the other using an overhead view in 2d graphics. I created a 3d graphics engine through which the player could explore the game space from a first person perspective; the client was written in C++ using OpenGL. A screenshot of the 3d engine is shown in figure 3.6.

The 3d engine accepted mouse and keyboard input, although the primary mode of interaction was using the keyboard in order to be more accessible to players without experience using mouse-driven first-person engines. The engine was designed to run existing PRIME games, using the xml files produced by the PRIME Designer and used by the other two engines.

My interactions with the PRIME Designer were mostly as a user of the tool and as the creator of the 3d game engine, but my experiences with it then led to my involvement with CoSolve, described in chapter 4. At this point I had worked with the extensive openness of the Data Factory and the possibilities of open processes in tutoring systems, and had been exposed to open problem-solving through the PRIME Designer. My next step was to apply these ideas of openness to CoSolve, the system that grew out of the ideas of state-space search design history of the PRIME Designer. Chapter 4 will describe CoSolve and my contributions to it in more detail.

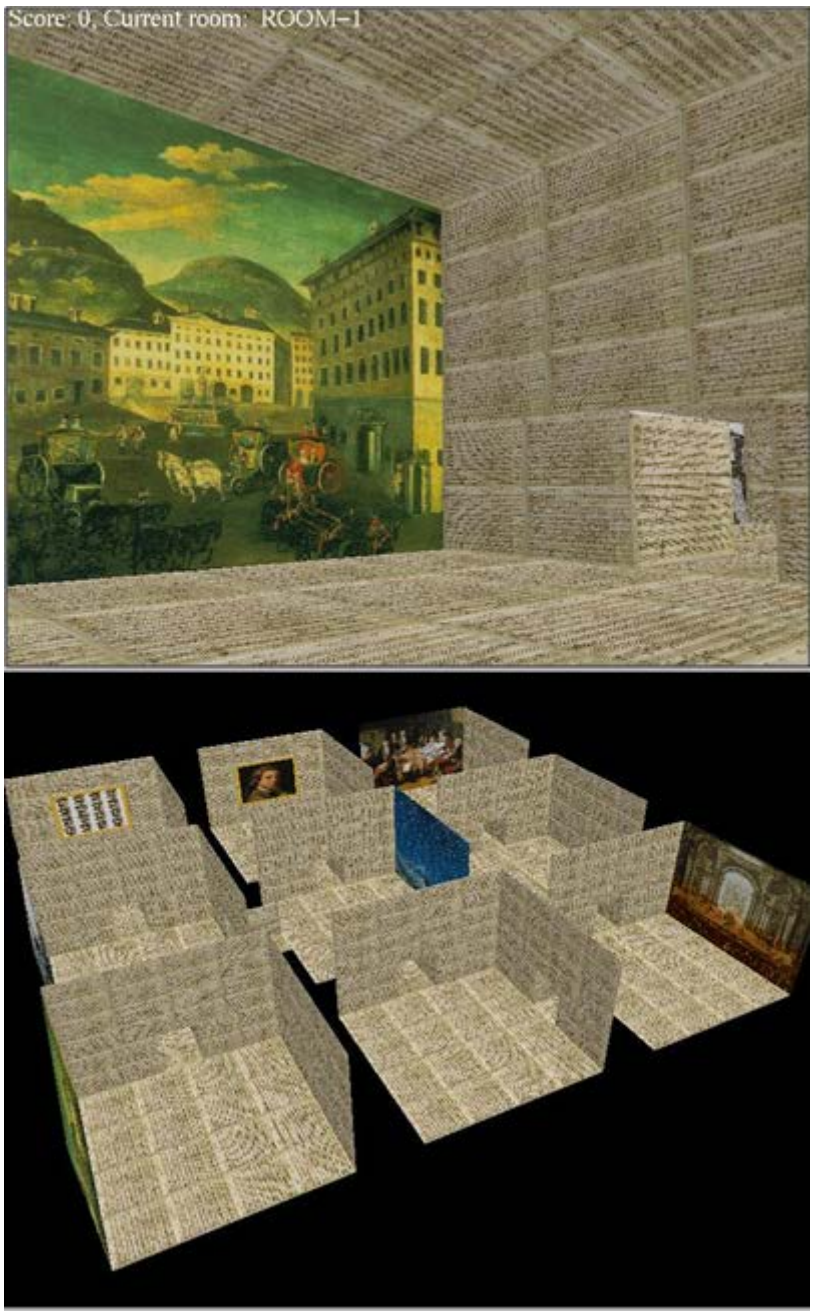


Figure 3.6: Two screenshots of the 3d engine written for games created by the PRIME Designer. The top screenshot shows a player's perspective from inside a room. The bottom screenshot was taken from a perspective floating outside and looking down on the three by three grid of rooms.

Chapter 4: CoSolve, the CoSolve Consultant and a Study to Investigate Them

In my work I applied openness to CoSolve, an online tool for supporting collaborative design and problem-solving. This chapter will first describe aspects of CoSolve in order to pave the way for a discussion of the additions made to it. A complete exploration of CoSolve would be infeasible and unnecessary in this work. However readers can find more information in the literature [Tanimoto, Robison and Fan 2009]. Openness was added in the form of the CoSolve Consultant, my work, which is also described in this chapter. Second, this chapter will describe a suite of tools I developed to analyze solvers' use of CoSolve and the CoSolve Consultant. Third, this chapter will describe the design of the study created to evaluate the CoSolve Consultant and to learn about solvers using CoSolve.

4.1. CoSolve

CoSolve allows users to come together online to collaboratively solve problems within its framework. This collaboration may be synchronous or asynchronous, and can be conducted over an arbitrary period of time, with any number of users working together on a problem. Users in the system that solve problems are referred to as 'solvers'.

In order for solvers to solve, the problem to be solved must be precisely described to the system. This is the task for 'posers,' users who create formulations of problems to solve, which may then be worked on by solvers. The roles of posers and solvers are not exclusive; users can hold both simultaneously, and indeed, most posers are solvers.

The details of posing a problem aren't entirely relevant here, but a high level view is provided to give some insight into the system. The poser constructs a 'problem template', which describes the problem to be solved completely and precisely; once such a template exists, it can be instantiated, and solvers may work on that instantiation, called a 'solving session.' Templates are written primarily in the Python programming language, and in them posers describe several things: 1) the initial state

of the problem, 2) how a state is shown visually to a solver, and 3) a list of operators that, when applied to a state, create a new state.

The role of the solver is then to join a solving session and apply operators to existing states; this will then generate new states, to which the solver can then apply operators, building a solution step by step. One novel aspect of CoSolve is that it displays the solving session as not just the most recent state, but all states, shown as a tree with the initial state as the root; this is similar to (and was inspired by) the PRIME Designer. 'Children' of a node (states created by applying an operator to that 'parent') are shown one level down from the parent in the view. In CoSolve the entire design history is visible; this makes it easy for solvers to view and build on any of their past work.

Let's consider an example to illustrate this. One puzzle readers may be familiar with is the 'Eight Queens' problem. In this problem we start with a blank chess board– an 8x8 grid. The goal of this puzzle is to place eight 'queen' chess pieces such that no queen threatens another, according to the rules of chess. That is, a goal state will have exactly eight queen pieces so that no queen can be reached by another via moving in a horizontal, vertical or diagonal line. Solving this may initially involve a fair amount of trial and error, and so is ideal for solving in CoSolve, given CoSolve's affordances to help solvers keep track of their solving process. The CoSolve template for the Eight Queens problem was originally created by Mike Duong, an undergraduate in the Computer Science department at the University of Washington.

In the context of the Eight Queens problem, an empty chess board is the initial state, and each operator lets the solver place a queen on the board. In order to be consistent with the rules of the Eight Queens problem, a move to place a queen is only allowed if it does not cause a queen to threaten another; the operator will fail if you try to place it somewhere illegal. If a solver performs such an illegal move, the resulting state will be an error state which allows no further operations; effectively a dead-end. However, since CoSolve keeps track of all previous states, it is easy to recover from an error state by simply backing up and trying again elsewhere.

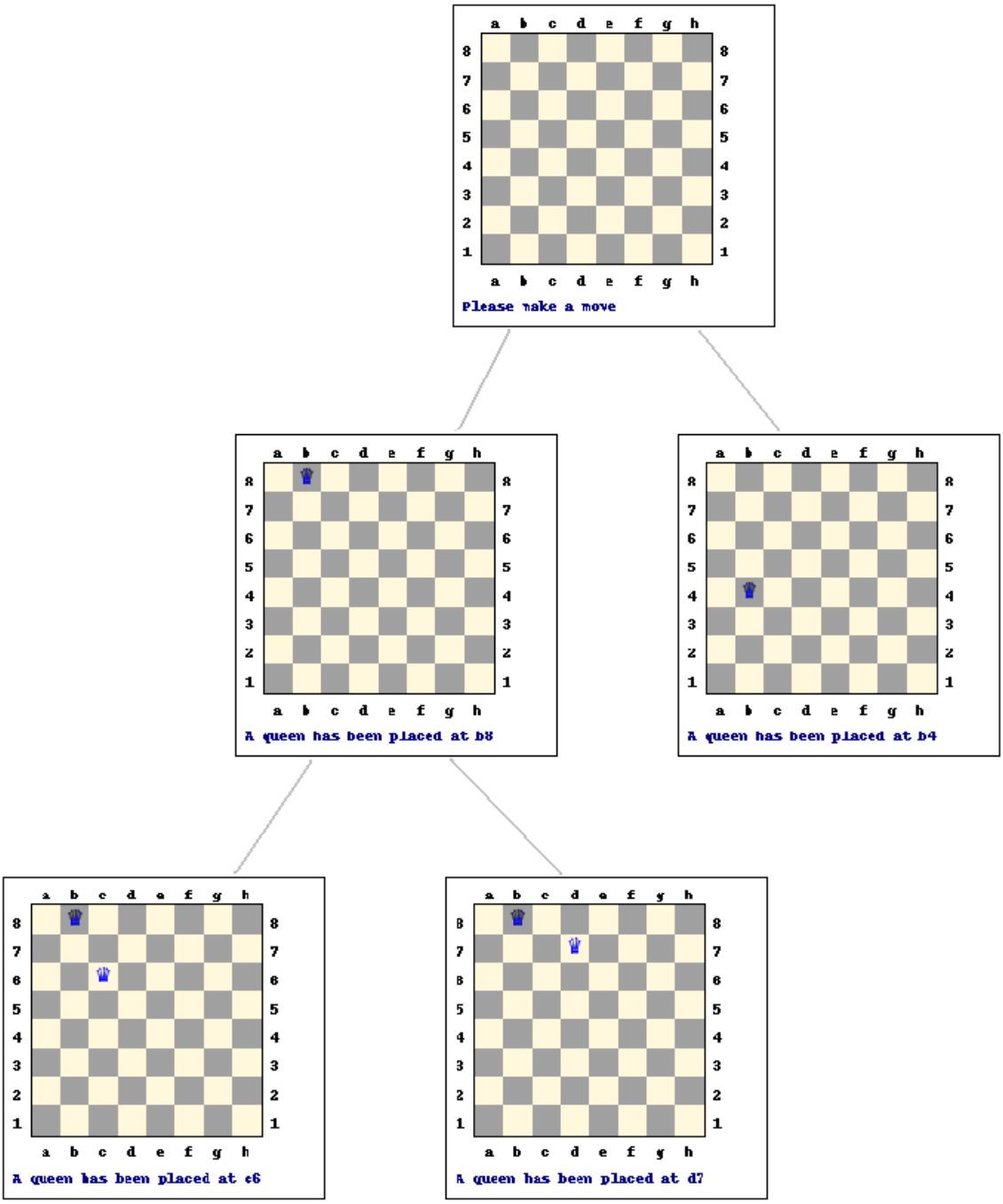


Figure 4.1: This screenshot of the CoSolve Flash client shows a solving session for the Eight Queens problem. The top (root) node contains the initial state—an empty chess board—and its two children show the effects of applying operators. The left child of the root itself has two children. The deeper a node is in the tree, the more operators have been applied. This Eight Queens template for CoSolve was created by Michael Duong.

Figure 4.1 is a screenshot of a solving session in the CoSolve Flash client, where a solver is trying to solve the Eight Queens problem. Each node in the tree is thus a state of the problem, beginning with the root—the initial state, an empty board—which is shown as the topmost node in the tree. In the figure is a tree with several branches, each representing another possibility, and so the process is not simply a linear one. For the next step in the design process the solver can apply an operator to any of the existing states; for instance, a solver can apply an operator to the node just created and continue his work, back up to 'redo' a previous turn, or even jump somewhere entirely different in the tree. CoSolve does not allow for the deletion of nodes, so the entire solving history will always be visible and capable of being built upon.

CoSolve's tree view of the solving session is one of its biggest advantages. First, no work is ever 'lost': every possibility tried is remembered and can seen and potentially be built upon later. This preserves the entire solving history and allows solvers to jump back to any state at which they had previously been. Second, it makes it easier to collaborate on the same problem, as multiple solvers can all work in the same space without stepping on each others' toes.

In CoSolve, any number of solvers can come together to solve a problem in a solving session. All solvers can see the work of all other solvers, and all are free to build on any existing node. The resulting tree can then have nodes authored by any solver scattered about. Because there is no deletion of nodes in CoSolve, other solvers cannot 'interfere' with your work: they can only add to it.

Trees can grow quite large, and so navigation tools are needed. While it is often desirable to have a zoomed-in view of a node when applying an operator to it (to see the fine details of a node, for instance), to understand the process as a whole it is more useful to zoom out and view the entire tree. Solvers using the CoSolve Flash client can pan and zoom around at will, making it easy to zoom in and focus on a particular node, or to zoom out and view the entire tree; these are standard techniques for navigating graphs [Herman 2000]. Each solver controls her own view of the solving session; the location and scale factor are stored in the client, and are not characteristics of the solving session itself. That is, a solver changing her view does not interfere with the views of other solvers.

This common workspace, in which the ideas and products of all solvers are shown to and capable of being continued by all, meets the conditions of a feature of coordination in the taxonomy described by Penichet [Penichet 2007]. Inherent to the idea of coordination are the ideas of conveying to team-mates the status of the problem and of the team, and of managing future work. The open view of the team's work so far helps to convey the team's history and inform regarding what has been done. CoSolve does not attempt to explicitly dictate to solvers what to do—that is, it does not play the role of a manager—but it gives solvers functionality with which they can better coordinate themselves. Similarly, this shared view matches the notion of information sharing in the taxonomy, as the 'document' being created (the problem solution) is accessible to all, as are the intermediate results leading up to it and various alternative explorations. Because solvers in CoSolve work directly on a shared document (as opposed to working on their own copies and passing them around), CoSolve demonstrates a very direct type of information sharing, according to the taxonomy.

In addition to being able to apply operators, and thus being able to create new states, solvers can also annotate existing states, adding text associated with that node which can be read by other solvers as well as themselves. These annotations can be used for a variety of purposes, such as to comment on the state or note future plans. Multiple annotations can be created for a node; multiple solvers can hold a discussion about the node in this manner. The author of the annotation has the option of marking the annotation as 'positive,' 'neutral' or 'negative,' depending on whether the author believes the node to be leading to a good, so-so or bad solution.

The annotation system provided by CoSolve directly meets the notion of a communication feature as understood in the taxonomy described by Penichet [Penichet 2007], as it provides functionality needed to pass messages back and forth between solvers. This ability to 'talk' to other solvers serves a number of roles in the collaborative problem-solving process, such as documenting internal thoughts and interactions, and complementing the coordination process. Due to the way annotations in CoSolve are implemented, the communication can be either synchronous or asynchronous, permitting some degree of flexibility.

In the study to be described, solvers were told to mark a node as ‘positive’ if they believed it to be a promising node on the path to a good solution, and as ‘negative’ if they believed it to be a step towards a poor solution. This good/bad evaluation relied entirely on the judgment of solvers.

In the case of the Eight Queens, the problem has an explicit goal: a state in which there are eight queens on the board simultaneously, placed using legal moves. Such goals are not necessary for CoSolve templates; a poser (a user who creates a problem template in CoSolve) can make problem templates that allow solvers to solve without end, going as deep in the tree as they like. Even if the problem has a well defined goal, solvers can always keep adding nodes elsewhere in the tree; they can always make it broader, if not deeper. In its current form, CoSolve trees can be built upon indefinitely.

The interface with which solvers interact with the CoSolve system is a Flash client that runs in a web browser. There are other clients for CoSolve, including HTML and Java clients, but they are not relevant for this work and will not be discussed here. The code for the Flash client is written in Adobe’s Actionscript language, and was built using the Flare toolkit for information visualization, which is based on the prefuse toolkit [Heer 2005]. The client is the product of several programmers: Christopher Brenan (who designed much of the original system), Robert Thompson, Sandra Fan and myself. My contributions have primarily been adding the CoSolve Consultant and doing general debugging. The CoSolve website itself consists primarily of PHP, and was created by Sandra Fan using Drupal, an open-source content management system. The CoSolve Flash client communicates with the CoSolve website through a series of Drupal web services, through which it provides and receives information.

As mentioned, any number of solvers can work together in a solving session, and solving sessions can grow arbitrarily in terms of the number of nodes and annotations contained. In the CoSolve Flash client, when a solver loads the solving session, all nodes are loaded and displayed. If another solver adds nodes, the views of other solvers of the same session will not automatically update; only the author of the node will see the new node appear. In order to see these new nodes, other solvers must manually click a button to update the view of the tree. An element of the display keeps

solvers informed of how many more nodes they will see on an update, saying, for instance, “You can update your view to see 4 new nodes.”

In order to better facilitate communication between solvers, the CoSolve Flash client has also been outfitted with a window displaying all annotations in the session. The list is ordered chronologically, and includes information on who placed the annotation and at what time. In addition, solvers can click on items in the annotation list to cause the view to move to and zoom in on the annotated node.

Solvers in a CoSolve solving session are all ‘equal’ in the sense that they have the same capabilities. The work of Sandra Fan, a member of the CoSolve research group, has looked at assigning roles to solvers, allowing for explicit division of labor. However, the work presented in this thesis is independent of the roles system, and so the roles system will not be discussed.

One observation regarding CoSolve is that it provides an open view into the problem-solving process that is there regardless of the tool used; consider a solver trying to solve the Eight Queens problem with a physical board and pieces. The solver may ponder, make moves, take notes, start from scratch and perhaps ‘undo’ moves; CoSolve simply provides a clearer view into that process.

Prior to my involvement, CoSolve was a powerful tool for problem-solving, and was open in the sense that it provided a complete view of the problem-solving process, and not just the current state, as is the norm in problem-solving and design environments. That said, solvers in CoSolve were somewhat insulated from each other: while they worked together in the same space, and could communicate via annotations, collaboration was often difficult due to the inaccessibility of some information. Understanding who has done what for a couple of solvers is easy in a tree of a few nodes, but what about a tree consisting of hundreds of nodes? How can solvers evaluate their progress as a team, and their contributions, as well as the contributions of their team-members? CoSolve does an excellent job of opening up the problem-solving process, but also introduces complexities of its own, and can leave solvers in the dark regarding their solving process and team interactions.

The next section will describe the CoSolve Consultant, which I created to address these issues of collaboration and self/team awareness. Through additional

interactions, the CoSolve Consultant tries to help make solvers more aware of the solving process, as well as the processes of their team-members and of themselves.

4.2. The CoSolve Consultant: Opening up CoSolve

The CoSolve Consultant was added to CoSolve in order to better establish the openness in CoSolve. The CoSolve Consultant essentially consists of a series of tools added to the usual CoSolve interface, each of which exposes the solvers to additional information and new ideas that can help them in the problem solving process.

The very idea of openness fits the concept of reformulation, as described by Fischer [Fischer 1990]. In reformulation, the user is given additional insight into how the computer system 'understands' the problem, and so allows users (solvers, in this case) to build up their own understanding and bridge the gap between the human and computer representations of the problem. The openness present in the CoSolve Consultant serves this purpose by showing how CoSolve 'understands' the problem-solving history and team interaction.

Figure 4.2 shows an annotated screenshot of the CoSolve Flash client, including its full interface. The CoSolve Consultant is shown in the screenshot, and will be described shortly.

I will now describe the features included in the CoSolve Consultant, which was used by the experimental group in the study related later in this chapter. The Consultant was designed to further open up the solving process, giving solvers additional insight into the efforts of their teams, and into the CoSolve problem-solving process.

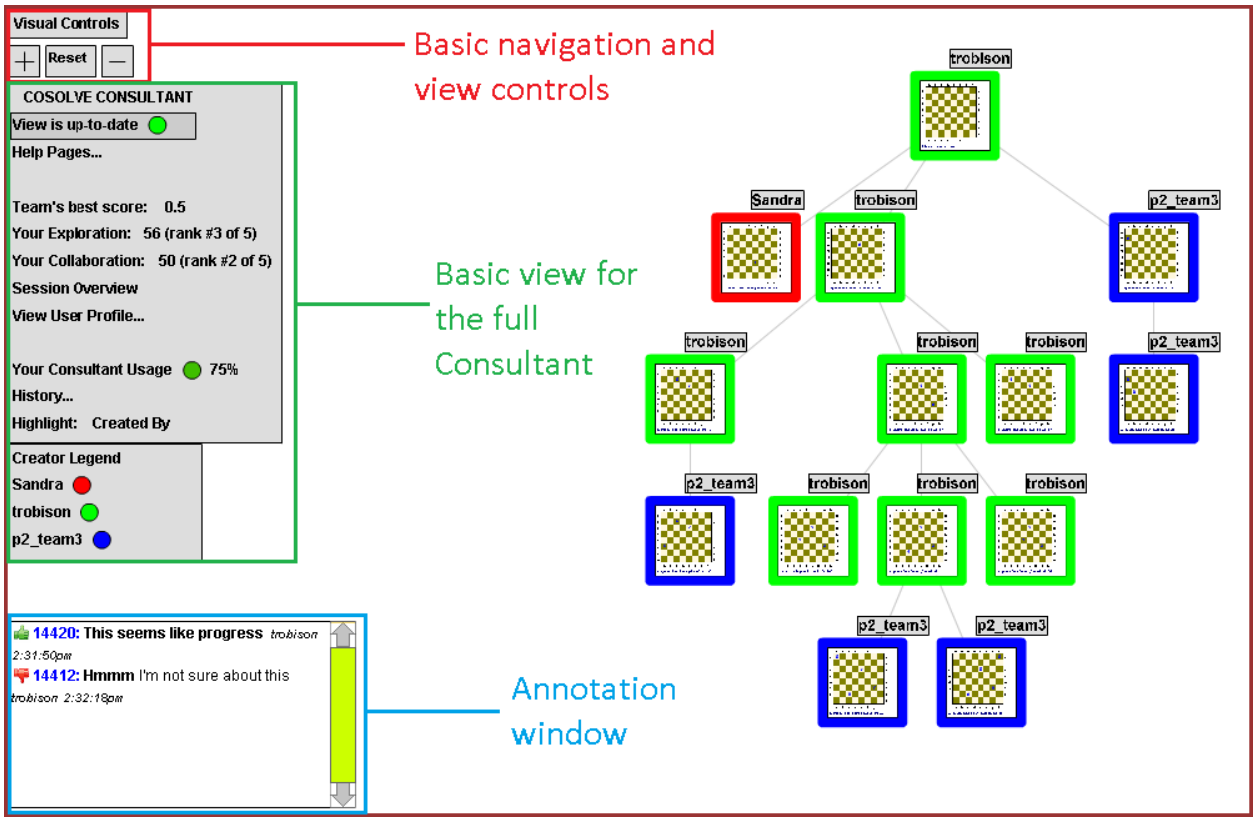


Figure 4.2: An annotated screenshot of the CoSolve Flash client. This screenshot shows nodes highlighted by creator, a feature of the CoSolve Consultant.

One of the key features of the CoSolve Consultant is that of highlighting, which shows the background of different nodes in different colors to convey additional information about that node. One of the most important aspects of the CoSolve system is the fact that solvers can zoom out and see the entire tree (representing the entire design process) all at once. Yet when zoomed out, while it is easy to see the whole tree structure, it becomes very difficult to see details of individual nodes, including who created them and what states they contain. To combat this, highlighting was added to allow solvers to zoom out and see the entire tree while still being able to get some information on a per node basis.

Each form of highlighting maps a color and label to each node; the color is shown behind the node and the label is displayed above it (the view of the node itself is not blocked, however). The label text is large enough to be legible from a partially zoomed out view; creating labels large enough to be visible from a fully zoomed out view would

difficult, given the need to draw several hundred nodes and their labels simultaneously while maintaining the tree structure. Being able to compare various nodes from a zoomed out view, using highlighting, can help reduce the cognitive load experienced when comparing nodes.

One lesson I learned from my work with the Data Factory, in both its versions, was that extensive openness, while interesting, is not always practical. That is, showing too much information can be problematic, as described by CLT. As a result, the highlighting system for the CoSolve Consultant is designed to show only a single highlight at a time. This is also in line with the CLT idea of expertise reversal, in that different views of the nodes may be simultaneously useful to experts and confusing to novices, or vice versa; by letting solvers choose for themselves, we can potentially avoid confusion. The highlighting view also shows the information directly on the node, preventing cognitive load that may occur if the two were shown separately and had to be mentally connected by the viewer.

Several forms of highlighting are included in the CoSolve Consultant. All were created to address problems I had directly encountered while using CoSolve, or had observed others encounter. Solvers can change the form of highlighting at any time via a menu in the Consultant. An initial possibility was to allow solvers to turn on multiple forms of highlighting at once, though this could have resulted in quite a bit more extraneous cognitive load at any given point, as there would be much more information being displayed, much of it likely uninteresting to the solver. It also would have complicated the interface for choosing highlighting, and so the current interface allows only one form of highlighting at a time. The forms of highlighting are as follows:

Score: This form of highlighting gives each node a color based on how its score compares to the highest scoring node in the solving session. The colors range from a dark red for lower scores to a bright green for higher scores. The text label indicates what percent of the high score this node represents. This highlighting scheme was designed to address problems of finding ‘good’ nodes in large trees.

Solution Path: This form of highlighting computes the path from the root of the tree to the highest scoring node, and displays all nodes along that path in bright green, and displays all other nodes as a dull red. The text label simply indicates whether the node is or isn't on the solution path. This highlighting scheme sounds similar to the 'Score' highlighting, but addresses the problem of finding the highest scoring node. In my experience building large trees in CoSolve, the 'Score' highlighting is helpful for approximating the value of a node, but small differences in color are difficult to pick out, and so a new scheme was needed to target the solution path.

This form of highlighting serves the same purpose as the process openness in the open tutorials I designed, described in section 3.3; it shows the solver how the system regards her progress. And, as suggested in 2.1 and observed in 3.3, this can improve the solver's trust in the system. One difference between the two is that in CoSolve, a solver could, in theory, determine the same information manually without the Consultant by looking at each node, though it would be a slow and tedious process.

Annotation Count: This form of highlighting shows many annotations each node has. Nodes without annotations are left without a highlight, while nodes with a small number are shown with a red border. The more annotations a node has, the brighter and greener its border becomes. This highlighting scheme was designed to help with finding out where solvers were annotating, such as important nodes that had received a lot of attention from solvers. It provides a different perspective from that of the annotation window, in that the highlighting gives solvers a spatial perspective of where the tree is being annotated.

Recently Created: This form of highlighting displays the relative 'age' of nodes. The oldest nodes (those with the oldest timestamps) are shown as dull red, whereas the newest nodes are shown as bright green. The computation is weighted to emphasize new nodes; the oldest 2/3 of the nodes are shown in dark red, to make comparing the ages of new nodes easier, as more recent nodes are generally of more interest.

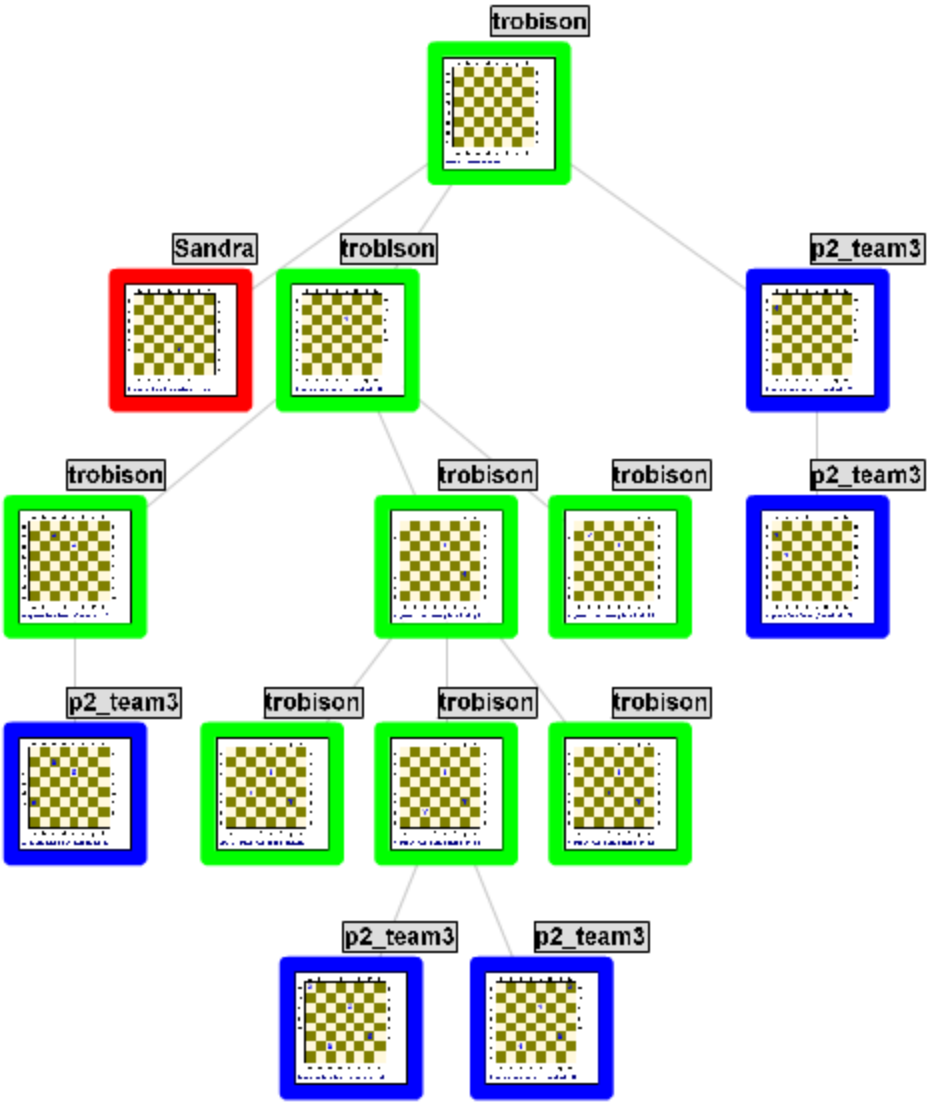


Figure 4.3: A screenshot of an Eight Queens solving session where three solvers are working together. Each node here lists its author above the node, and has its highlighting color dependent on the author.

This highlighting scheme was created in order to, first, help solvers understand the layout of the tree: for instance, “This branch was the first created, then this one, and this subtree is the newest.” Second, it serves to point out where recent work has taken place, allowing solvers to understand where growth is occurring. Anecdotally, in my experiences with solvers and with solving during the construction of the Consultant

some of the barriers to collaboration stemmed from solvers being unable to separate the old from new portions of the tree.

Created By: This form of highlighting displays a color indicating the author of the node. Each author has a unique color associated with her; this mapping can be seen in a legend shown to solvers. The text label associated with each node displays the solver name. This highlighting scheme was created to help solvers understand the collaborations of their team-members. For small trees, say of a dozen nodes, it's fairly easy to keep track of who has done what, but when the number of nodes in a tree grows into the hundreds, it can be confusing to figure out who has done what. In my own experience, this view also seems to also be helpful in navigating the tree, since as a solver I tend to remember my own work, and so can use the shape and color of portions of the tree as landmarks in navigation.

Annotation Balance: This form of highlighting indicates how well a node has been rated, in terms of positive, neutral and negative annotations. Like other forms of highlighting, this form uses color interpolation from a dull red to a bright green. A red indicates that the node has received mostly negative annotations, whereas green indicates that it has received mostly positive annotations. A muddy yellow in-between indicates that it has received mostly neutral annotations, or that the positive and negative annotations have balanced out. This highlighting scheme was created to help solvers find good and bad nodes, according to their own usage of positive and negative annotations.

In addition to highlighting features, intended to convey information more usefully when the view is zoomed out, the CoSolve Consultant also provides a menu-based window system for conveying arbitrary text and graphic information. By clicking Consultant menu options, solvers can open a variety of windows displaying additional information about their work, the work of their team-mates and about the solving session itself. An example of this can be seen in Figure 4.4 below.

In Figure 4.4, several of the metrics are listed with an associated rank. In order to provide context for some of the information, specifically the statistics kept for each solver, some information is shown as a ranking among team-mates. Two hypothetical solvers (a high and a low solver; an expert and a novice) are added to these rankings to emphasize what high and low values are for a given metric. Clicking on a ranked item opens another window showing a list with values and their respective solvers.

This ranking is intended, first, to help solvers understand the values they are shown. Being told that one's 'exploration score' is 57 isn't very meaningful in itself, but being able to see it next to peers' scores, and those of an expert and novice can lend perspective and help solvers understand the value. Second, showing peer models can motivate solvers, as described in 2.1.

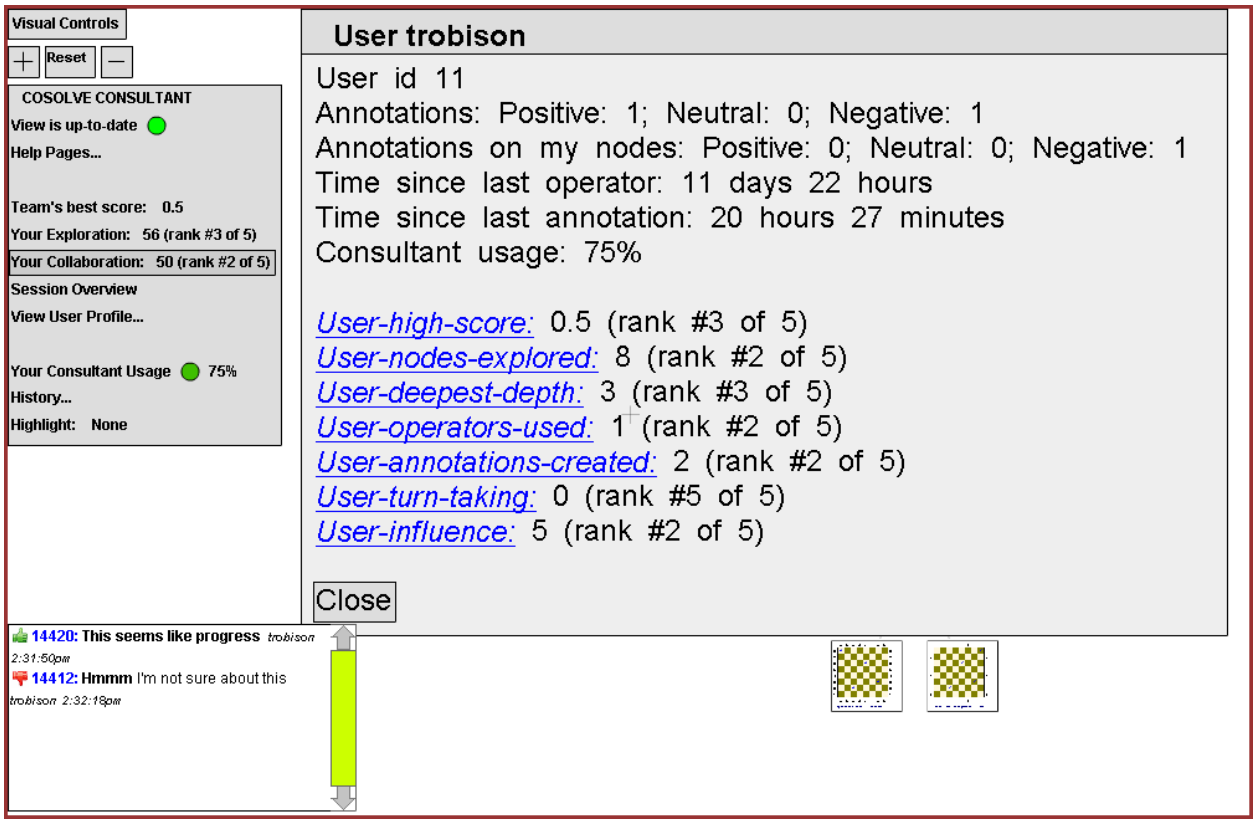


Figure 4.4: A screenshot of the Eight Queens solving session showing a Consultant window. This window shows information regarding the solver 'trobison,' such as timing information. Underlined text in blue represents a link: this text can be clicked to open up detailed information on that topic in a new window. The use of 'rank' is described in the text preceding the figure.

The information provided in the Consultant window system serves two purposes:

- Informs solvers of their contributions and statistics of usage. These have the aim of increasing knowledge awareness and team awareness, helping solvers understand how they work as individuals and as part of a team. Providing information on the actions of one's team-mates could also make it easier to collaborate, as they can more easily see the contributions of others. It can also help a new solver get up-to-date when joining a solving session in which a good deal of work has already been done.

- Exposure to CoSolve specific ideas. Because the tree-based design will be initially foreign to most solvers, numerous tree characteristics and their meanings may escape the notice. For instance, the notion that turn-taking is one way to view collaboration in CoSolve. The information provided in the window system can help solvers learn more about solving in CoSolve, and so perhaps understand and solve more effectively.

Any number of windows can be held open at one time. Windows can be dragged around, and can be closed by clicking a 'close' button. Solvers can interact normally with the CoSolve interface while windows are open: that is, nodes and annotations can still be created while windows are open. There are around 40 different windows that can be opened, in addition to one window for each solver describing that solver, as seen in Figure 4.4.

The content presented in the windows varies from presenting statistics about solvers and about the solving session to providing static text, such as descriptions of parts of the Consultant. The information presented was written to help solvers use CoSolve: it is not specific to any template (there is no advice, for instance, on how to solve the Eight Queen's problem).

To illustrate the range of details offered by the Consultant, a partial list of its contents is included:

- Session Overview: A summary of the work done in this session, including the total number of nodes, annotations and several other metrics such as average branching factor, narrowness (which is large for narrow trees and small for broad trees) and top-heaviness (which is large for trees with most work near the top and small for trees with most of the work near the bottom). These metrics quantify the shape of the tree. Narrowness and top-heaviness are defined more formally in 4.4.2.
- Solver Profiles: A summary of the contributions and statistics of usage for each solver.

- Solver Exploration Score: An aggregate metric representing how thoroughly the solver has explored the problem space, in relation to her team-members. The equation used to calculate this aggregate is included in the solver's view. This aggregate is based on the number of nodes created, the deepest depth of nodes created and the number of operators used.
- Solver Collaboration Score: An aggregate metric representing how thoroughly the solver has collaborated, in relation to his team-members. The equation used to calculate this aggregate is included in the solver's view. This aggregate is based on the number of annotations created, turn-taking (defined shortly) and a related characteristic called influence.
- Nodes on refresh: A display of how many new nodes (created by other solvers) the solver will see if they refresh their view. The ability to refresh was present in CoSolve before the Consultant was created, though in the Consultant it was made more prominent and included a count of how many nodes would be added.
- Help pages: Links to information on the CoSolve interface and the rules for CitySim, the problem template used for the study (CitySim is described in more detail later in the chapter).
- Team High Score: The highest score achieved in this solving session.
- Consultant Usage: This value was an indication of how often the solver had made use of the Consultant. The sole purpose of this feature was to incentivize usage of the Consultant in the study; it would be excluded in actual usage of CoSolve to solve problems.

The metric called turn-taking, computed and shown to solvers in the Consultant through the 'Solver Collaboration Score' window, is an interesting metric in that it measures how closely team members work together. When a node is created, if the author of that node is not the author of the parent as well, then its creation is counted as an instance of turn-taking. Turn-taking is illustrated below in figure 4.5.

Figure 4.5: An illustration of the turn-taking metric: on the left, all three nodes have different authors (and so are displayed using different colors), and so results in maximum turn-taking for three nodes (a value of 2), and on the right all three nodes have the same author, which results in the minimum turn-taking (a value of 0).

Turn-taking is then a direct metric of how much one has built off the work of one's team-mates. It is a fairly strict metric in the sense that nodes must have parent/children relationships to meet the definition; if, in figure 4.5, the blue solver added a large number of child nodes to his previous node on the left, it would not change the turn-taking value, despite there being substantially more work in the subtree started by the yellow and red solvers. Nonetheless, turn-taking is a useful metric, and one is conceptually related to Barron's metric of accepting or discussing correct proposals [Barron 2003]. At the moment, turn-taking only indicates when a solver extends the work of another; it does not distinguish whether the addition was desired or appreciated by the solver whose work was extended. It is conceivable that future versions of this metric could take into account the quality of the added node as well; this would further illuminate how solvers interacted.

Much of the information in the CoSolve Consultant's windows is organized so that the solver can click items for additional details or explanations; very little information is thrust upon solvers unless they request it. The resulting setup spreads information across numerous pages, though this has the nice benefit that a solver will only be confronted by information she requested. Apart from avoiding unnecessary extraneous cognitive load, this also makes it less likely to fall prey to the Cognitive Load Theory problem of 'expertise reversal,' namely that forcing the same interface on novices and experts has the potential of getting in the way for both. By letting solvers

explore as they will, it is less likely that they'll stumble upon unwanted information. This setup does have the downside of requiring more clicks, however.

The metrics computed and shown to solvers in these windows is a form of restructuring, as described by Fischer [Fischer 1990]. In restructuring, the designers of a system seek to make it easier for users to use by moving away from low-level language to higher-level language more easily understood by the humans. In this case, the low-level language consists of raw counts, parent pointers, creator ids and timestamps; to make this more meaningful and useful to solvers it is conveyed to them as higher level metrics describing the shape of the tree and degrees of collaboration.

4.3. Evaluating the CoSolve Consultant Using the Cognitive Dimensions Framework

The Cognitive Dimensions framework, described in chapter 2, will now be used to evaluate the CoSolve Consultant. To begin, we will consider CoSolve without any Consultant features. Recall that CoSolve without any Consultant features means that there is no highlighting and no information on the high score, collaboration, etc. Other features, such as the ability to create nodes, annotations, refresh the view and navigate the tree are present and unchanged. This evaluation focuses on the solving aspect of CoSolve (as opposed to the posing aspect), and is independent of the details of the problem being solved.

Visibility and Juxtaposability: The visibility in CoSolve is somewhat limited, as though the tree view gives a complete view of the problem-solving process, it also means that there is a great deal of information being shown, making it harder to sift through looking for something in specific. Consider the task of finding out which are the most recent nodes created: this would normally be a slow process of zooming in on each node and checking the timestamp, node after node. Yet finding recent nodes could prove useful for solvers trying to get up-to-date on recent progress.

Juxtaposability has similar limitations. Comparing nearby nodes, say a parent and child, is easy and often helpful, but there is no easy way to compare nodes

physically distant in the tree. Using the zoomed out view, showing the entire tree, means it difficult to see the details of the nodes. Currently, one way to compare two distant nodes is to pan from one to the other using a zoomed in view. This is a slow process that requires retaining the details of one in memory for comparison with the other. Juxtaposability is better for annotations, as a solver can more easily compare those using the annotation window.

Annotations and the annotation window can be used by solvers to some extent to improve both visibility and juxtaposability: nodes can be annotated to make them easier to find later, and a solver could in theory annotate the two nodes being compared, and use the annotation window's feature of clicking an annotation to jump to its node to quickly jump between the two nodes for comparison. This too would be a slow process, and would create numerous wasteful annotations that do not represent any meaningful communication, as annotations are meant to.

Abstraction: For the most part, solving in CoSolve requires few abstractions. Operators are applied one at a time, without any sort of macro system and CoSolve does not let solvers define abstractions. The result is that the system is easier to learn, but lacks the ability to define a series of operators as a single macro. That said, abstractions could be introduced by the template author, but template creation is beyond the scope of this paper.

Hard Mental Operations: CoSolve can be said to have some minor problems with hard mental operations in that problems with visibility and juxtaposability may require keeping several bits of information in mind, such as the details of the state I want to compare, or where in the tree I saw a promising node. The menu system found in the CoSolve system has only a few options, and is therefore relatively simple to navigate, and so doesn't seem to contribute to the hard mental operations of CoSolve. There can certainly be hard mental operations imposed by the problem being solved, but CoSolve itself seems largely free from them.

Progressive Evaluation: Progressive evaluation is another dimension in which CoSolve is limited, as it can be unclear even to an experienced solver whether a given

tree is ‘good’ or not, or how much progress is taking place. A solver can perhaps evaluate whether a good solution exists by looking through all terminal nodes (assuming criteria for success are well defined), but in terms of gauging collaborative activity and individual contributions, it can be very difficult to evaluate. Without any sort of highlighting it can be very difficult to tell who has done what.

We will now examine how the open features of the Consultant change the position of CoSolve in the Cognitive Dimensions framework.

Visibility and Juxtaposability: One of the primary contributions of the CoSolve Consultant, in terms of the Cognitive Dimensions framework, is the addition of tools that improve visibility and juxtaposability. The highlighting features make it easier to find regions of the tree based on desired criteria: recent additions, areas by different solvers, high scoring areas, etc. One particular use we saw in the study was using the ‘Highlight by Solution Path’ feature to show where the highest scoring node was. Currently, more involved searches, such as finding the highest scoring node created by a given solver, are currently not possible; combinations of highlighting schemes and additional ones can be considered for future work.

The Consultant improves the juxtaposability of CoSolve, but does not completely eliminate the problems. The highlighting features make it possible to do comparisons from a zoomed out view: eyeballing the age of nodes on different sides of the tree, for instance. There are still some problems, such as trying to compare nodes with similar values from a zoomed out view, as the color differences between them may be difficult to pick out with the naked eye, but overall highlighting provides a great improvement for juxtaposability. The large text labels that accompany the highlighting can be seen from a partially zoomed out view, making such an intermediate level feasible for comparing nodes.

Abstraction: The CoSolve Consultant introduces new abstractions in the form of computed metrics. These have the advantage of encapsulating high-level ideas, such as the ‘narrowness’ of the tree (one of several computed metrics shown to solvers, and described in 4.4.2), making it easier for solvers to understand the tree. On the

other hand, they have the disadvantage of increased complexity, as they must be understood in order to be used. The added complexity is light, however, as the metrics are mostly simple and designed to be easy to understand, and their use is completely optional.

Hard Mental Operations: The open features of the Consultant alleviate hard mental operations in the sense that juxtaposition and searching become easier. The Consultant does, however, potentially increase hard mental operations in another way. The menu and window systems used by the Consultant are more complex than the menu system built into the CoSolve client, in large part because they hold more content. As a result, navigating the menu and window systems is somewhat more difficult. That said, while they are more complex, they are still not all that complex, and so the increase is fairly minor.

Progressive Evaluation: One large improvement provided by the Consultant is that it makes it easier for solvers to evaluate the progress of their team and the team-members individually. Most of this is due to the metrics provided to solvers describing the state of the tree itself and the contribution of different team members. One feature it is lacking is some sort of progress evaluation for specific states in the problem being solved; say, deciding whether or not a given CitySim state is good or not. While no doubt helpful, this sort of functionality would be heavily dependent on the problem, whereas the Consultant is designed to be usable for any type of problem in CoSolve.

From the perspective of the Cognitive Dimensions framework, the addition of the Consultant appears to provide a gain in visibility/juxtaposability and progressive evaluation, and has some benefits for increased abstraction. On the other hand, it may contribute to hard mental operations, and there are some minor downsides to the increased abstraction. The net gain, however, appears to be positive, and the Consultant appears to have shifted CoSolve towards a more useful state.

4.4. Consultant Analysis Tools

Another contribution of my work is a suite of tools with which solvers and solving sessions can be analyzed. Some of these are analyzes shown to solvers through the Consultant interface; these are described in the section on the Consultant and need not be described again here. Other tools exist unknown to solvers and are meant to be used by researchers after the session has ended. This section will briefly describe some of these, and the analyses of the data from the study performed will be described in chapter 5.

As mentioned, CoSolve is in many ways a unique system: it is, to the best of my knowledge, the only collaborative problem-solving system that visualizes the session history as a search tree. One downside of this novelty is that we lack established ways of evaluating the performance of solvers in CoSolve. Certainly, simple metrics like number of nodes and annotations can be used (and are used), but in light of the unique tree structure that emerges from a solving session, it seems wasteful to ignore that complex visual representation in favor of simple metrics.

The analysis tools can be roughly grouped into three categories:

- Session History
- Metrics
- Solver Action Profiles

Each of these will be described in subsections below.

4.4.1. Session History Tools

As with many processes, seeing a static snapshot of one point in time of a CoSolve solving session does not effectively convey what happened to reach that point. A small tree, such as some of the trees shown earlier in this chapter, may have been created branch by branch, or level by level; solvers may have worked together at the same time or at different times; work may have been done in bursts or done at a steady

pace. In short, much of the information on the solving and collaboration is lost if you view a static image.

For this reason I created the CoSolve Consultant History Tool, which allows one to 'replay' the solving session, showing the ordering of nodes and annotations and helping to tell a more complete story. While the tool is a part of the CoSolve Consultant interface and can be run at any time during or after a solving session, it was not accessible to solvers during our study. The tool has the functionality one would expect from a replay system: ability to jump to the start, fast forward, rewind, pause and slowdown/speedup the replay. The time from the start of the solving session, in minutes, is shown during the replay. The History Tool does not modify the tree in any way; it is only an altered view of a solving session.

4.4.2. CoSolve Metrics

I created a number of metrics to help characterize CoSolve solving sessions; while watching a history replay of the creation of a session, it is sometimes desirable to be able to describe the session numerically. For instance, it is convenient to express solver collaboration as a few different values so that the values from different solving sessions can be compared. Some of the more important metrics, many of them used in the analysis of the results of the study in chapter 5, are described here. Simple metrics, such as node or annotation counts, are left out here. Also, this list contains only automatically computed metrics, and not those that require human tagging or judgment.

Turn-taking: Described earlier in this chapter, *turn-taking* is a metric of how closely solvers work together, and can be computed per individual or per team. To summarize, a node creation event is considered an act of turn-taking if the author of the node is different from the parent node's author. An area of the tree constructed entirely by the same author will have a low turn-taking value, and an area where solvers built nodes off of each other will have a higher turn-taking. Turn-taking is useful as a metric in that it fairly closely matches my intuition of 'working together,' in a fairly direct sense.

Peer Annotations: An annotation is considered a *peer annotation* if the author of the annotation is different from the author of the node on which it is placed. The presence of a peer annotation, as opposed to a 'self annotation' (an annotation on one's own node) can be viewed as evidence that the author was to some degree aware of the operations of her team-mates. Currently peer annotation is computed for each solver but does not distinguish which peer's nodes were annotated; the tools could conceivably be modified to shed light on which solvers tended to work together.

Top-heaviness: This metric is computed for the entire team and indicates where the bulk of the tree's nodes are. A high value (with a maximum of 1) indicates that the majority of the nodes were created toward the top of the tree near the root; that the tree is 'top-heavy,' which suggests that the team was more concerned with exploring more in the early stages of the problem. A low value (with a minimum near 0) indicates that the majority of the nodes were created lower in the tree, which suggests that the team was mostly concerned with exploring the end stages of the problem. The top heaviness of a session is a function of the average node depth, and is computed as follows, where N is the set of all nodes in the solving session:

$$\text{TopHeaviness} = 1 - \frac{\sum_{n \in N} depth_N}{|N|\, TreeHeight}$$

This value is undefined for trees with one or zero nodes. A long branch with many children of the root has a top heaviness near 1, whereas a long branch that ends in a node with a large number of children will approach a top heaviness of 0.

Narrowness: The *narrowness* metric is computed for the entire team and indicates whether the tree is broad (a value near 0) or narrow (a value near 1). A broad tree suggests the exploration of many possibilities at each level, whereas a narrow tree suggests a more focused effort on fewer branches. The narrowness metric of a session is a function of the average node depth, and is computed as follows, where N is the set of all nodes in the solving session:

$$\text{Narrowness} = \frac{2 \sum_{n \in N} depth_N}{|N|(|N|-1)}$$

Narrowness is undefined for a tree with one or fewer nodes. A single long branch will have a narrowness of 1, whereas a root node with a large number of children will approach a narrowness of 0.

Number of Same-Author Connected Regions: A *same-author connected region is a connected region* of the tree where all the nodes have the same author. A small number of such connected regions would indicate that solvers tended to work mostly in their own areas, whereas a large number would indicate that solvers interacted more in node creation, and did not maintain much 'private' space in the tree. A derivative of this is the average size of a solver's connected regions.

Standard Graph Theory Metrics: This is a broad category I use to cover graph theory metrics that can be meaningfully applied to CoSolve trees. This category includes things such as: leaf node count, tree height, average branching factor, tree width, average node depth and single child count. While not contributions of mine, these were used in the analysis of the solving sessions.

In addition, the tools were capable of breaking down the metric calculations into smaller time units than simply 'the entire session.' Teams were given two five minute breaks, cutting their solving time into three subsessions of 25 minutes each (these and other details are described in section 4.5); the tools were able to generate statistics per subsession as well as for the entire solving session.

4.4.3. Creating Profiles for Solvers

In order to better understand how solvers use CoSolve, I've created a tool that takes a solver's actions as input and generates a transition table as output. This can help us see patterns in solving behavior, such as tendencies to perform certain actions one after the other. The table gives $P(\beta \mid \alpha)$, the probability that the solver will now perform action β given that she has just performed action α. The tool also generates a priori probabilities of each of the actions. From this data one can get a better sense of what actions solvers perform, in the context of other actions. While interesting on its

own, this data is more useful when comparing and contrasting the behavior of different solvers, as is done between solvers in the control and experimental group in chapter 5.

One choice that needs to be made in generating these profiles is determining the granularity of the actions: for instance, should all node creation events be considered the same class of events, or are there interesting subdivisions (node created off of root, off of peer node, off my own node, etc.)? Perhaps any usage of the Consultant should be labeled 'Consultant usage,' or perhaps each different feature of the Consultant should get its own separate action.

This categorization could be done a number of different ways. After some experimentation, starting with fewer categories and expanding the number, I came up with the following categorization:

All solver actions were classified into one of the following ten categories:

- Three types of annotation events: On the root node, on peer node (a node created by a team-member) or on the annotation author's own node
- The categorization for node creation is similar to the categorization for annotation, except that 'created on own node' is broken up into two categories, depending on whether the node was created as the child of the node author's previous node (referred to as 'consecutive') or not. Hence there are four categories of node creation events: Child of the root node, child of a peer node, a consecutive self node or a non-consecutive self node
- Refresh event: This occurs when the solver requests to see nodes created by others since the last refresh
- Use of some feature of the Consultant
- Changing the view of the states; in CitySim, the problem used for the study, solvers could change their view at will to see different information presented in each state. CitySim is described later in this chapter and in the appendix 1.

The division of annotation and node creation events into 'root, peer or self' is important because of the implications of actions on peer nodes, namely that it suggests

awareness of the work of the team-member and willingness to comment on or further develop the work. The distinction of 'consecutive self nodes' versus 'non-consecutive self nodes' is interesting because the former suggests digging deeper into an idea, whereas the latter suggests 'jumping around.'

4.5. The CoSolve Consultant User Study

In order to better understand the CoSolve Consultant, and to determine whether it met its design aims, a single blind study was designed and conducted to examine its effects on participants. The study was run jointly with Sandra Fan, who had research questions separate from my own to answer. Our studies shared the control group, but each had its own experimental group; hers will not be discussed here. My research goals for the study were as follows:

> **G1. Effects on Collaboration:** Investigate whether solvers with the Consultant demonstrate more collaborative behavior
>
> **G2. Effects on Problem-Solving:** Investigate whether solvers with the Consultant use different patterns of design, resulting in different shapes of trees
>
> **G3. Effects on Vocabulary:** Investigate whether solvers with the Consultant show higher learning gains of relevant vocabulary words (tree terminology adapted for CoSolve), and whether they annotate more regarding the structure of the tree
>
> **G4. Effects on Metacognition and Team Awareness:** Investigate whether solvers with the Consultant experience improved metacognition and team awareness
>
> **G5. Understanding Solvers:** In general, develop an understanding of how solvers make use of CoSolve, with and without the Consultant

In the study the experimental group used CoSolve and the CoSolve Consultant, and the control group used CoSolve and a small subset of the Consultant's features. The CoSolve Consultant described earlier in this chapter and used by the experimental group will be referred to as the full Consultant, and the subset of the Consultant's features used by the control group will be referred to as the mock-Consultant. Initially, the study was planned to test the effects of the Consultant versus no Consultant interface, but early in the design process it was decided that solving collaboratively in a large tree could become very difficult without certain information and navigation features, and so the control group was given certain basic functionality through the mock-Consultant.

The mock-Consultant is a stripped down version of the full CoSolve Consultant. It was created solely for use in a control group in the study. A small subset of features from the full Consultant were included in the mock-Consultant:

- Nodes on refresh
- Help pages
- Team's Best Score
- Consultant Usage (whose sole purpose was to incentivize usage of the Consultant in the study)
- Highlighting by creator

In my experiences using CoSolve before the study, two of the most common hindrances were figuring out who created what, and knowing what the high score of the session is. The features in the mock-Consultant primarily address these issues.

What the mock-Consultant lacks, when compared to the full Consultant, are additional highlights for viewing the tree zoomed out and numerous windows for conveying models of solvers, progress of teams, and additional metrics through which to view the session.

Figure 4.6 below shows a screenshot of the mock-Consultant. Note that this appears very similar to the previous screenshots; the main difference is that the mock-Consultant simply lists fewer features.

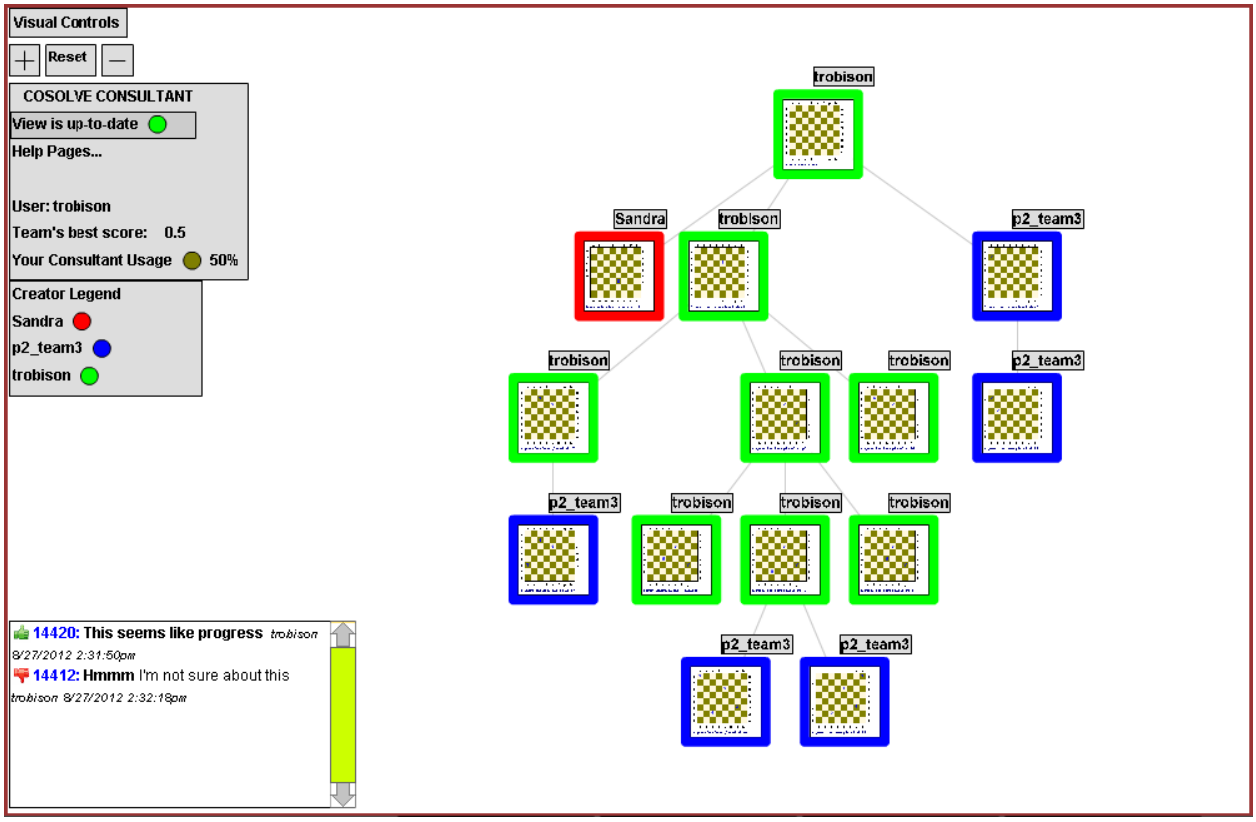


Figure 4.6: A screenshot of the CoSolve Flash client with the mock-Consultant, which consists of a subset of the features of the full Consultant.

Overall, the mock-Consultant provided solvers with functionality to show which node was created by which solver using the highlight and to show the high score held by the team at any given moment. Information in the full Consultant but not the mock-Consultant includes solver and team profiles and exposure to tree and collaboration metrics.

4.5.1. Finding a Problem to Solve

To understand and evaluate solver behavior, it was important to have solvers grouped into teams with the objective of solving a problem in CoSolve; the next question was, 'what problem should they be given to solve?' In CoSolve this amounts to the question of what template to use, as the template dictates everything about the problem in CoSolve: the starting state, possible operators, and the consequences of applying an operator to an existing state.

There were (and are) numerous templates in CoSolve that could have been used. However, there were some criteria the template had to meet:

1) Simplicity in Description: The problem needed to be fairly simple in terms of its rules, so that it could be given and explained fairly quickly, and so that difficulties understanding the rules would not interfere with the solving.
2) Complexity in Game-play: The problem had to be difficult enough that it could not be easily solved: having the problem solved perfectly in a couple of minutes would result in a very small and uninteresting tree. The participants were to try to solve the activity over roughly an hour and a half, so the problem needed to be solvable in that time.
3) Depth: The solution to the problem needed to be attainable only after numerous moves. If the tree were shallow, we wouldn't be able to gather much data on how solvers build off of each others' work.

In addition, the following characteristics, while not essential, were seen as desirable:

1) Had a well-defined 'score' metric that users are to strive to maximize. This both gives the users immediate feedback on their success, and allows easier comparison of the solutions of different teams.
2) Can be framed as a game in order to help motivate users.
3) Problem's complexity rewards exploration, trial and error. This could result in more time spent exploring the space, and greater collaboration as users share their strategies with one another.

In order to have a problem in CoSolve that met all of the essential conditions, and met as many of the optional conditions as possible, a new problem template was created from scratch. Existing templates met some of these conditions, such as being game-like and being able to be reasoned about by users, but nothing existing was the right level of complexity.

4.5.2. The CitySim Problem Template

The problem template created for the study was called 'CitySim,' and will be described in this section in order to help the reader better understand the context of the study. I designed and programmed CitySim, basing it loosely on concepts of the popular series of 'SimCity' computer games. The solver plays the role of the builder of a city, initially an empty plot of land. Through construction of buildings and the passage of time tax income is earned.

The solver is in control of an initially empty city, represented by a 2-dimensional 10 by 10 cell grid. A solver can apply an operator to place a building in a cell, which may cost some money, and may have effects on nearby cells. Unlike SimCity, this game is turn-based; each placement of a building takes one turn.

The initial state is, as was stated, an empty grid, and has a small amount of money that can be used to purchase buildings. There are 6 different types of buildings which can be placed in a grid cell, each costing varying amounts and with varying effects. Most such effects take place in a 3x3 rectangle centered on the building: a residential building, for instance, increases the population by 1 for all cells in that area. An industrial building increases jobs in that area by 2, but reduces happiness in that area by 1. Other types of buildings also modify jobs or happiness in the area.

When a cell has both jobs and population, 'employment' is generated in that cell, which results in tax income every turn. If, however, happiness falls below a certain threshold in a given cell, no tax income is generated from that cell. Two special buildings affect the size of the effect rectangle and tax rates.

The goal of the game is to get as much money as possible in the 30 turns given (one building can be placed per turn). This money must come from tax income, which in turn comes from employment, and so the game is ultimately about juggling the jobs, population and happiness values in the grid. Note that '30 turns' means that after 30 buildings have been placed, that state displays the final score and will not allow any operators to be applied to it; solvers can, of course, backtrack in the tree and apply operators elsewhere.

While the template itself was being designed, several iterations took place to ensure that the game is not too difficult to learn, but was sufficiently complex that obtaining an optimal score was non-trivial. This version of the game allows for a great deal of exploration: the branching factor is roughly 600, as the solver can place any of 6 buildings (if the city has sufficient money) in any of 100 cells. The game ends in 30 turns, so the number of possible states is roughly 600^{30}.

The view shown to solvers at each state shows the 10x10 grid on top, and some textual information below (money held that turn, income, cost of buildings, etc.). The items shown in the grid are initially the buildings placed, though the solver has the option to change the view to see other information there instead: population, jobs, employment and happiness. The solver can freely switch between these views as desired; changing the view is not an operator and does not result in the creation of a new state.

The details of the rules of the game aren't important for describing the study, and so they won't be covered here. They can be found in appendix 1.

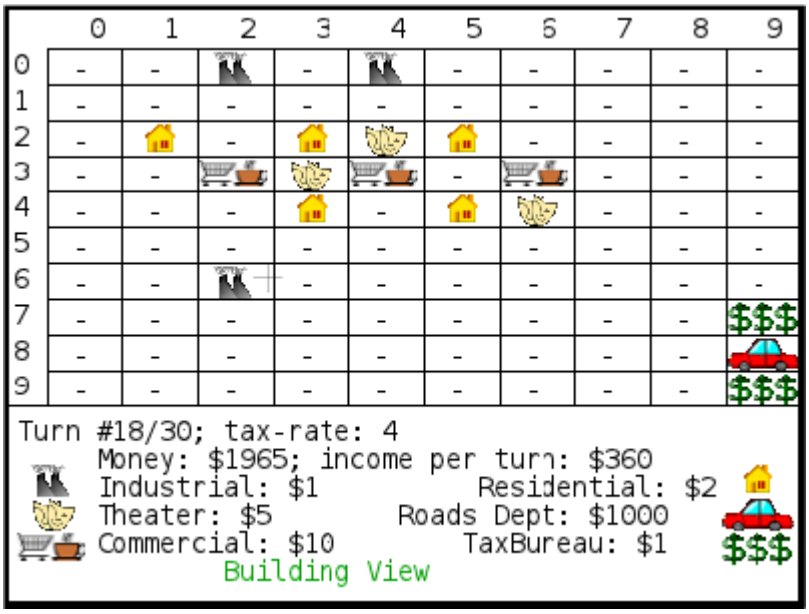


Figure 4.7: A screenshot of a CitySim game in progress. The game is on the 18th turn, and is in 'Building View' mode.

4.5.3. Study Details

Participants were formed into teams of three to meet with the researchers for a block of roughly three hours in reserved computer lab. This time included filling out a number of forms and then undertaking the activity. A wrap-up meeting was scheduled to meet within a couple of days; a separate meeting was set up for each participant. Each wrap-up meeting took around an hour, after which the participant would be completely done and would be given the payment, a $20 gift card.

Session schedule:

1. Consent forms filled out by participants
2. Background questionnaire filled out by each participant
3. Vocabulary pre-test filled out by each participant
4. An overview of the process was described to participants, and they were asked to introduce themselves and pick a team name

5. Introductions to CoSolve, CitySim and the CoSolve Consultant were given through a combination of verbal description by a researcher, PowerPoint slides and hands on tasks performed by the participants to get them accustomed to the activity
6. The solving activity began: this was started in a new, empty session (so that they could start fresh from their activities in step #5). The activity itself consisted of 3 25 minute blocks called subsessions, separated by brief breaks

For the CoSolve activities in this study participants were asked not to communicate verbally: in order to simulate solvers working together remotely we only allowed them to communicate using annotations in CoSolve. While somewhat artificial (in reality remote solvers could compute verbally via a number of technologies), it forced solvers to use the annotation functionality in CoSolve, and allowed the researchers to easily produce a complete log of communications.

To incentivize participants of the study to make a genuine effort towards solving the given problem, each member of the winning team (the team with the highest scoring node) was given another $20 gift card after all the teams had been run.

A total of six teams of three participants each took part in the study in the fall of 2011. Participants were recruited through Computer Science Department mailing lists, word of mouth and Craigslist. Participants were grouped into teams based on scheduling availability. Sessions were held in reserved computer lab space in the University of Washington Computer Science Department.

To examine the effects of the CoSolve Consultant, teams were either given the control or experimental treatment. Teams were not informed which group they were in, and were not made aware of the different conditions during the study. The teams in the control group will be referred to here as Control 1, 2, 3 and Experimental 1, 2 and 3. The order in which the teams were run is as follows: Control 1, Experimental 1, Control 2, Experimental 2, Control 3 and Experimental 3.

The introductory material, tests and forms were the same for groups in different conditions, with the exception of the introduction to the Consultant, which focused only

on the features given to that group (either the mock Consultant for the control group or the full Consultant for the experimental group). Also regarding the introductory materials, some changes were made to the presentation after the first couple of teams (one control and one experimental) had been run; the same content was presented, but the presentation was tweaked to be easier to understand and more stream-lined.

The schedule for the wrap-up session is as follows (recall that a separate wrap-up session was held for each participant):

1. Participant takes post-test
2. Researcher interviews participant; this was recorded if the participant consented
3. Participant takes wrap-up questionnaire

The post test and pre test were actual identical, though spaced out by at least a couple of hours. Some changes were made to the tests after the first 2 teams were run because of the level of difficulty experienced by participants. In the analysis of the results of the pre and post test, the first two teams (one control and one experimental) run are left out. Changes to the wrap-up interview were also made over time in order to better get at points of interest to the researchers.

4.5.4. Main Solving Session

As stated above, the main solving session (which took place after the introductory materials) consisted of three subsessions of 25 minutes each and breaks of about 5 minutes between them. The timing was enforced by researchers to ensure that different teams received the same amount of time.

When the first subsession began, the solvers were set to work on an initially empty CitySim solving session. There were no differences between subsessions: all CoSolve and CitySim features were available for all subsessions, and solvers were allowed to make as many nodes and annotations as they liked, wherever they liked in the session. All activities in CoSolve were logged by the system automatically, giving us a record of their actions in the system and associated timing information.

As was also stated above, solvers were not allowed to communicate verbally with each other; this too was enforced by the researchers. This rule extended to the breaks between subsessions, where team-members were allowed to talk to each other as long as they did not talk about CoSolve, CitySim, etc. Solvers were allowed, and encouraged, to talk to the researcher if they encountered problems with the tools. Researchers did not answer questions regarding strategy. Researchers also watched to ensure that the participants stayed focused on the activity (and did not surf the net, check email, etc.), though we never had any problems with this: participants stayed focused on the session until it ended.

Chapter 5: Study Data and Discussion

This chapter presents data obtained from the study of the CoSolve Consultant described in chapter 4, provides analysis and discusses the meaning of this data.

The primary purpose of the study was to examine the differences between the control group, which had access to the mock-Consultant, and the experimental group, which had access to the full-Consultant. Data was collected from four sources and analyzed to tease out differences between the two experiences of groups, including:

- Performance on task, both per team and per individual (this includes both nodes and annotations created, as well as various metrics gathered from the activity)
- Background and wrap-up questionnaires
- Wrap-up interviews
- Pretest and posttest

A secondary purpose of the study was to develop tools for analyzing problem-solving and collaboration in CoSolve. These metrics and techniques are described in chapter 4.

Before the study, a pilot study was conducted in which solvers used an early version of the full-Consultant. The study design originally had solvers work asynchronously, over the course of three days, and with the researchers' suggestion of working on the problem around one hour per day. The pilot participations found that this caused them to lose interest in the activity; one participant said that later on in the activity the task became a chore. It was then decided to run the solving session all on one day, which led to the current study design.

Participants made use of the Consultant during the pilot and reported it as being helpful in finding out high scoring areas and where recent work had been done. There was also the indication that metrics shown to solvers may have influenced solving behavior. Various software and the study materials were modified as a result of the pilot to better support the solvers, and to improve the data obtained from the study. The rest of this chapter deals with the results of the study.

5.1. Team Narratives

Quantitative data alone isn't sufficient to convey the collaborative process undertaken by solvers during the study; without having seen the session take place, much of the context is lost, and so, to provide context and make the discussion more meaningful, a high level summary of each team's problem solving session is provided below. Note that these descriptions are my interpretations of team interactions based on log file events, inspecting the tree and observing the solvers as they solved, and so it is possible that other observers could infer underlying causes somewhat differently.

Screenshots of the final trees are shown to display the variety of shapes and sizes possible, and to ground the coming narratives. In the trees shown, the color (red, blue or yellow) of the node indicates its author (the term in CoSolve for the solver who created the node). Solvers will be referred to by the color of their nodes shown on the screenshots. The root node (and only the root node) of each tree is green, indicating that it was created by a researcher.

The root is always shown at the top of the image, and a node's children (created by applying an operator to the parent) are always one level below it. In other words, the further down a node is, the further it is towards the end of the 30 turn game. When child nodes are created, they are created below the parent starting from left to right: that is, the first child of a node will be shown on the left, the second child just to the right of the first, and so on. Note that this ordering is only present among siblings, so observing the left/right position of nodes with different parents is not always helpful for figuring out their ordering. Relative ordering between nodes cannot necessarily be read perfectly from a static image of a tree. That said, branches on the left often tend to be older than branches on the right, which can help make sense of the tree.

If there are many sibling nodes with the same author, this often indicates that that solver was experimenting with different possibilities, while long, unbroken same-author branches often indicate that the solver was trying to see an idea through to completion. Nodes with children of all three authors often indicate that everyone found it to be an interesting node and decided to build from it.

Note the differences among the trees. They vary widely in shape and size, and these variations do not appear to be linked to the condition group. A series of metrics in the ‘problem-solving’ subsection investigates this in more detail.

The highest scoring node in each tree (and thus the one used for the team’s score at the end of the session) is marked in the screenshots below with a green box; all of the highest scoring nodes appear on the bottom row of the tree. In some cases, other nodes are emphasized by a green circle, the meaning of which is explained under the description for that tree.

Team Control 1 Profile

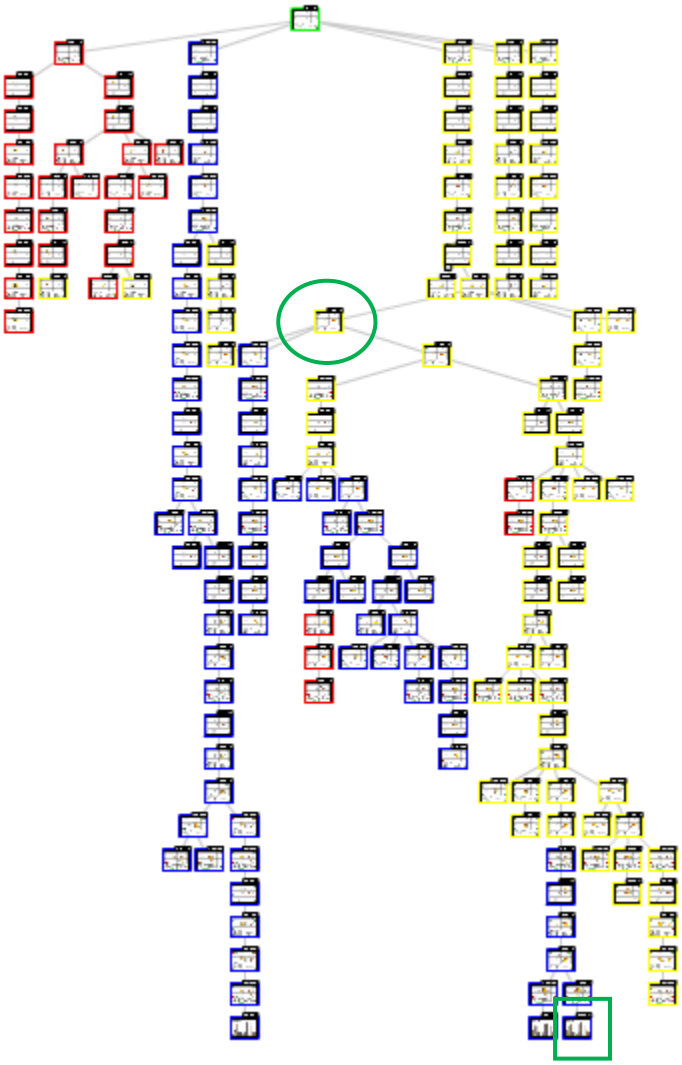


Figure 5.1: A screenshot of the solving session of Control Team 1, with each node shown in a color indicating its author. The highest scoring node is emphasized with a green box.

All three members of Control Team 1 started their own branches off the root. They worked on their own areas, for the most part, though some building off of each others' nodes did occur. Partway through the session, an annotation on a node of Yellow's appears to have rallied the team to work on that branch; most of the work after

that point took part in that subtree. The annotated node is at depth nine and appears horizontally near the center of the tree, and it is circled in green in this image.

The communication was somewhat asymmetric, as Red wrote a large number of annotations discussing strategy, and while Blue joined in for some of the conversation, Yellow was primarily quiet, though he does appear to have read the annotations of others.

The solving session was cut short due to an unexpected obligation of a team member, and so the group only performed during two of the three subsessions. Nonetheless, they achieved the highest score of any team and had among the highest number of annotations. The former is especially surprising, not only given the reduced time, but also because they had only three terminal nodes, all of them with author Blue.

Team Control 2 Profile

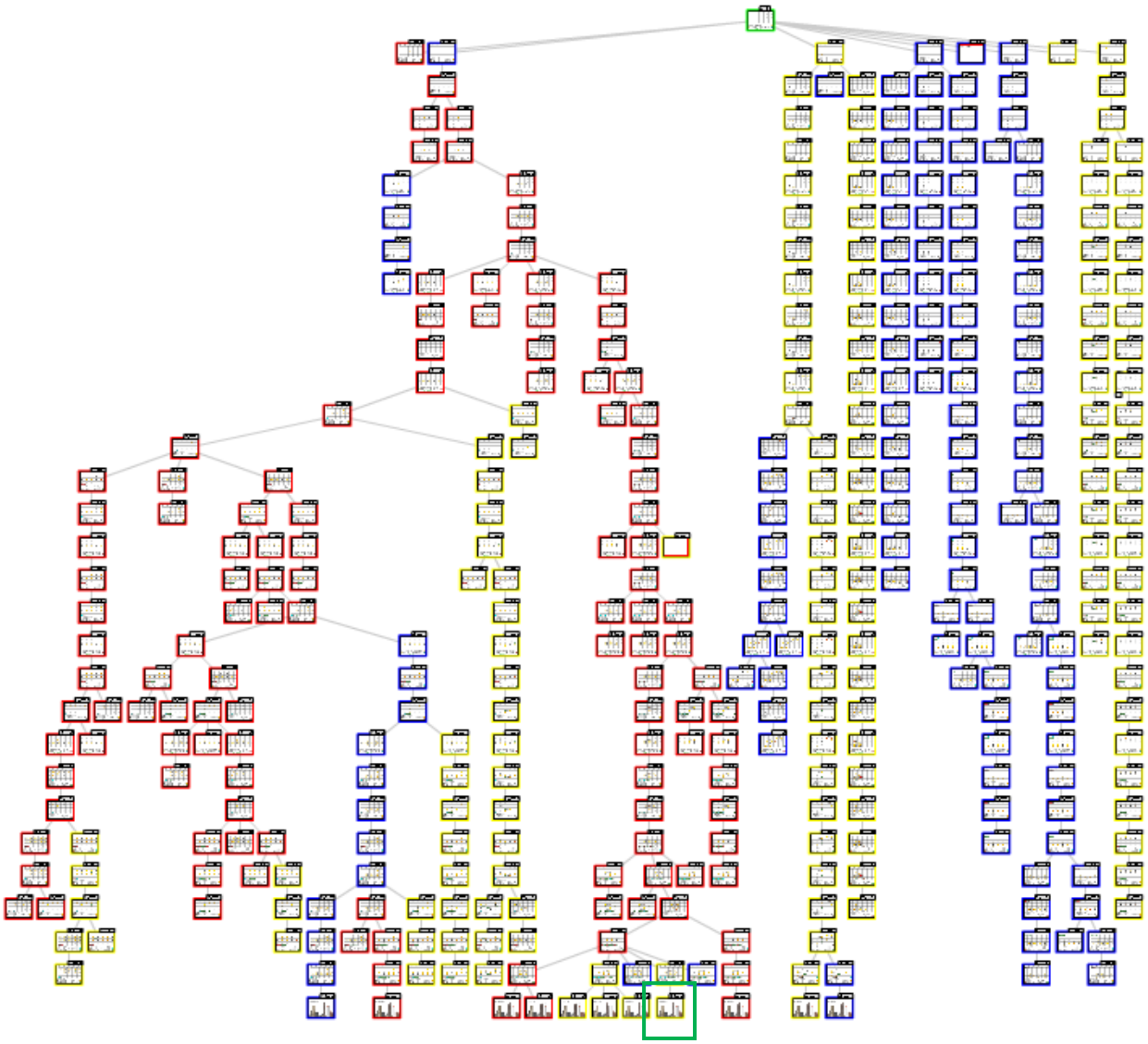


Figure 5.2: A screenshot of the solving session of Control Team 2, with each node shown in a color indicating its author. The highest scoring node is emphasized with a green box.

Control Team 2's members began working mostly on their own branches, though there was a fair amount of node creation off of each others' work. Later in the session they built more off of each others' nodes. Most of the annotations were conversational (compliments, acknowledgements, etc.) or discussing strategy, with Red and Yellow

annotating their own nodes while Blue performed a number of peer annotations (annotating the nodes of his team-members).

This team had a score near the average over all teams, and had the most nodes created among all teams. Among the control teams, this team had the most equal contribution of nodes from solvers among all three control teams.

Team Control 3 Profile

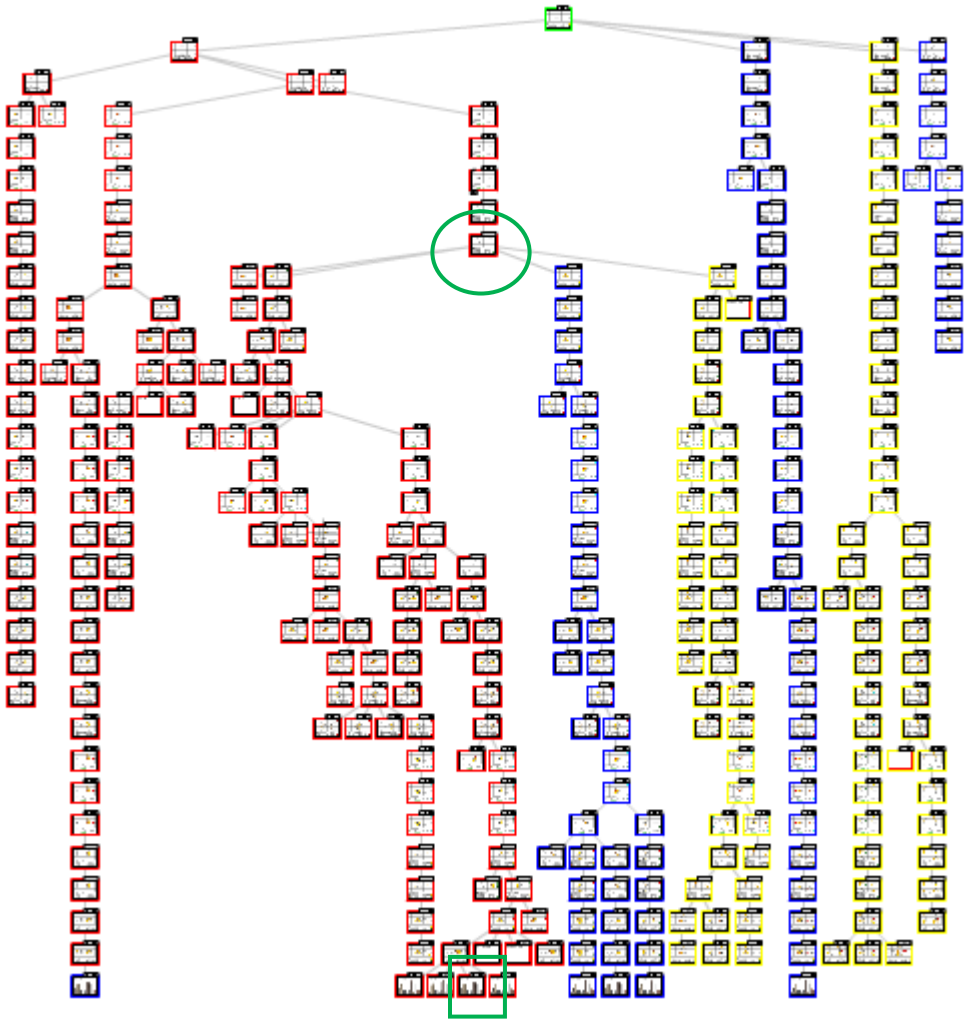


Figure 5.3: A screenshot of the solving session of Control Team 3, with each node shown in a color indicating its author. The highest scoring node is emphasized with a green box.

Control Team 3's members here started off on their own branches, and did not build off of each others' until much later in the session. An annotation by Red on one of Red's nodes drew the others to work in that subtree (which can be seen as most of the middle of the tree) for the rest of the session; this node is circled in green. Even within

that subtree, however, the solvers interacted little and focused mostly on their own branches.

This team has the distinction of having communicated the least, by far: while the average number of annotations per team is in the upper 20's, this team created only five total, with Yellow creating none at all. Blue and Red, when they communicated, discussed strategy and drew attention to nodes of importance.

Team Experimental 1 Profile

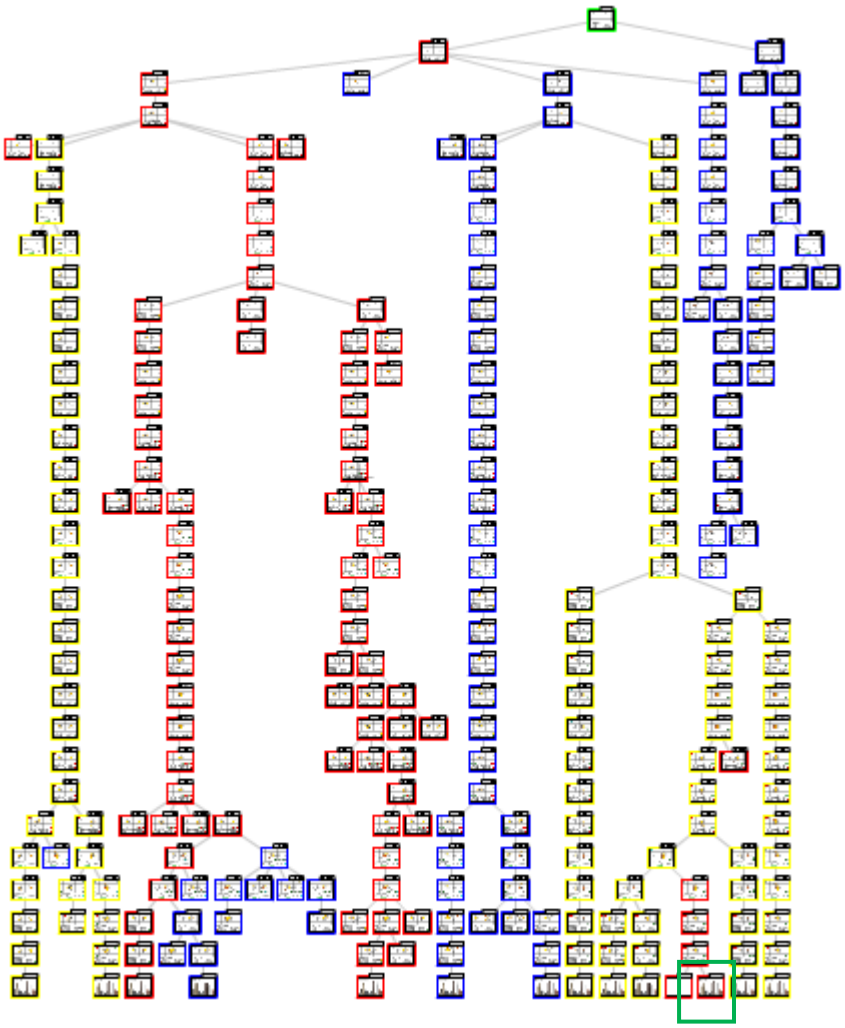


Figure 5.4: A screenshot of the solving session of Experimental Team 1, with each node shown in a color indicating its author. The highest scoring node is emphasized with a green box.

At the very beginning of Experimental Team 1's session Red began to build a branch down and the other two began to build their own branches off of it. About halfway through the session an annotation by Red at the very bottom of the tree drew Blue's attention to that area, where he proceeded to experiment. After that, the three worked mainly on their own branches, though with some occasional direct interaction.

Throughout the session, Red and Yellow were engaged in conversation with each other, mostly discussing strategy, though Blue joined in at a couple of points to comment.

This team had the second highest score, though the tree appears quite different than Control 1's (the team with the highest score), appearing larger and broader, with a great deal more work done on the lower levels of the tree (no doubt due, in part, to Control 1 only getting two of three subsessions). Also, unlike Control 1's tree, the work appears to be more evenly distributed among team members, with them having their own large areas and branches.

Team Experimental 2 Profile

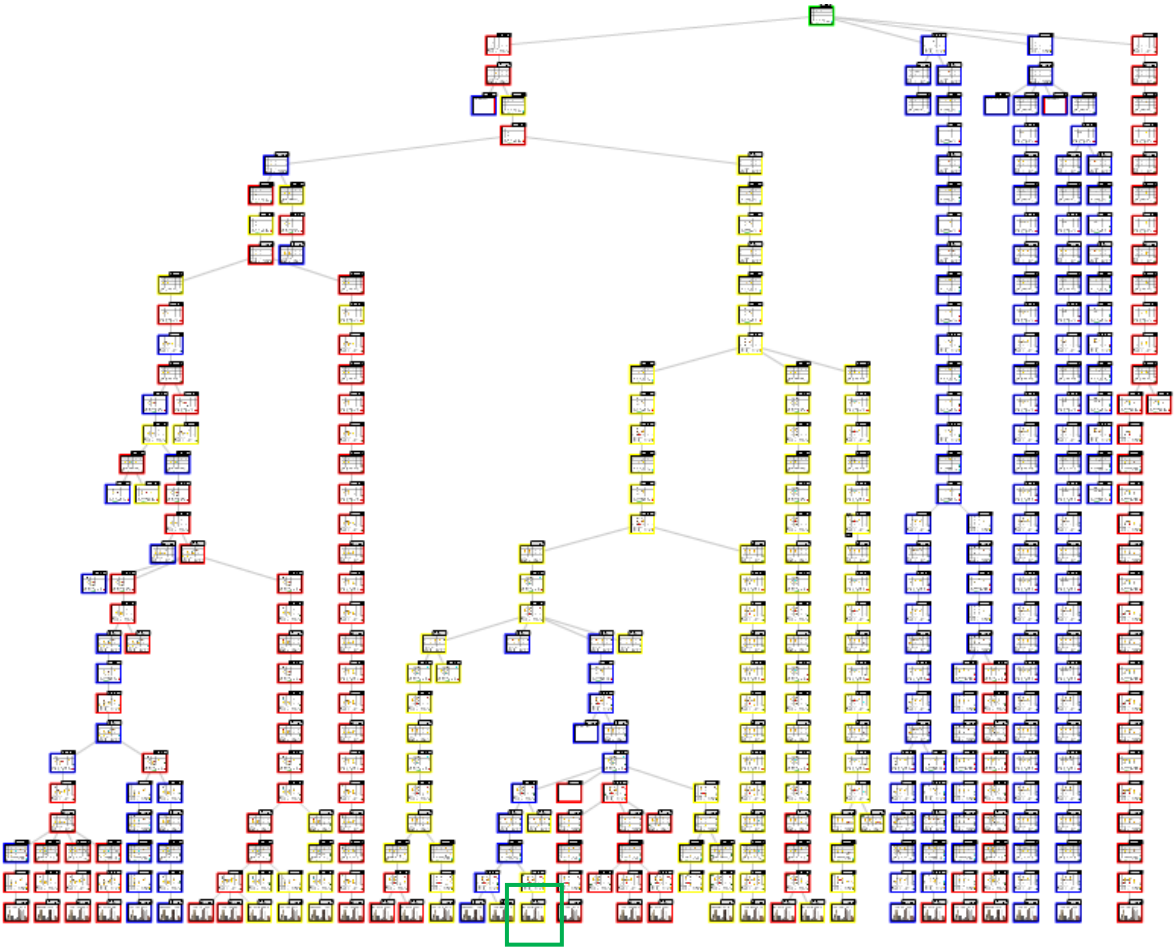


Figure 5.4: A screenshot of the solving session of Experimental Team 2, with each node shown in a color indicating its author. The highest scoring node is emphasized with a green box.

Experimental Team 2 started out working tightly together; most of their early nodes built off of the nodes of their team-members, as can be seen by the fact that the upper left of the tree shows nodes of the different colors frequently mixed together. After awhile they begin to work more on their own areas, but still came back together to work at several later points. This team also had a large number of nodes (the second most, overall) and a large number of victory nodes (nodes on the bottom level of the tree).

The team annotated fairly heavily in the beginning, when working together (both strategically and conversationally), but this tapered off when they began to work more on their own areas. Despite the large tree and reasonable number of annotations, this team had the lowest score of all the teams.

Team Experimental 3 Profile

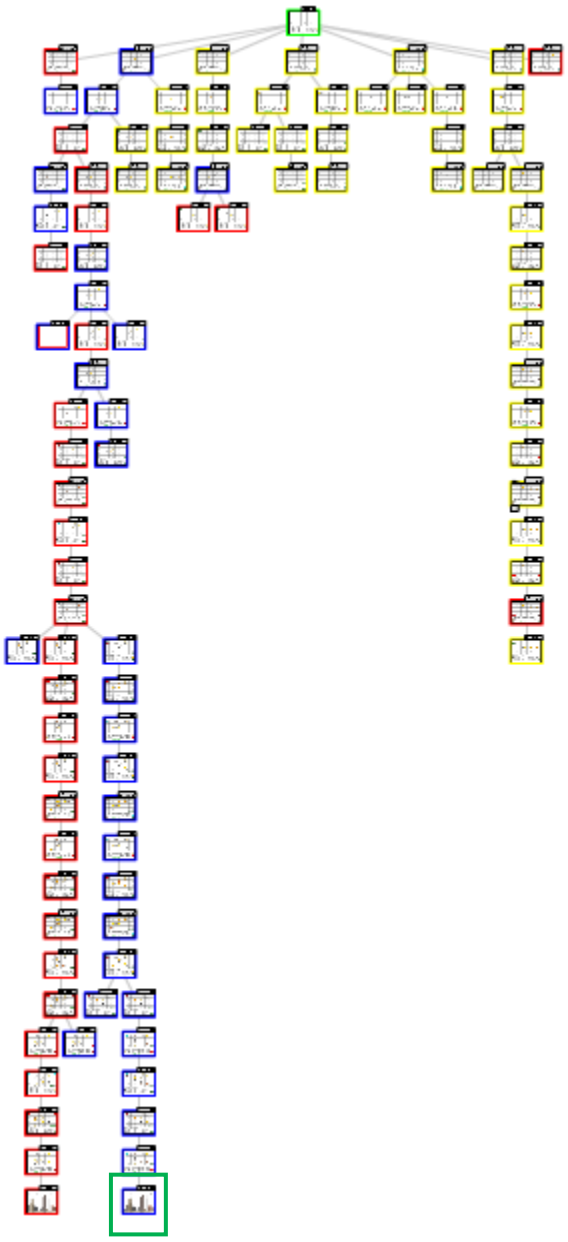


Figure 5.6: A screenshot of the solving session of Experimental Team 3, with each node shown in a color indicating its author. The highest scoring node is emphasized with a green box.

Experimental Team 3 had the fewest nodes by far—even fewer than Control 1, which had only $\frac{2}{3}$ of the total time. The session began with tight interaction and heavy

annotation; while this team had the fewest number of nodes by far, they had by far the most annotations.

The annotations discussed strategy a great deal, but also formed a conversation, with replies to questions and other statements being common. After the initial placement of nodes, team-members worked mostly on their own areas, though for Blue and Red they worked somewhat nearby. Yellow spent most of his time later in the session working alone on the top levels of the tree.

5.2. Exploring the Basic Data

The goals of this work are repeated below (having been presented in the chapter 4, which describes the study), and will be referred to in the analysis of the data:

G1. Understanding Effects on Collaboration: Investigate whether solvers with the full-Consultant demonstrate more or different collaborative behavior.

G2. Understanding Effects on Problem-Solving: Investigate whether solvers with the full-Consultant use different patterns of design, resulting in different shapes of trees.

G3. Understanding Effects on Vocabulary: Investigate whether solvers with the full-Consultant show higher or different learning gains of relevant vocabulary words (tree terminology adapted for CoSolve), and whether they annotate more regarding the structure of the tree.

G4. Understanding Effects on Metacognition and Team Awareness: Investigate whether solvers with the full-Consultant experience improved metacognition and team awareness.

G5. Understanding Solvers: In general, develop an understanding of how solvers make use of CoSolve, with and without the full-Consultant.

The differences between the control and experimental groups are explored both qualitatively and quantitatively, looking at numerous aspects of the differences between the systems. Qualitative data examined includes events and recognized patterns over time; investigating the solving trees; exploring possible explanations for observations made; scatter plots of participant/team behavior; and anecdotal evidence seen in questionnaires, interviews or in person. The quantitative values are metrics collected from solvers and teams over the course of the session. Some of these are fairly straight forward (such as counts of how many nodes were created during the solving session), while others are items of my own invention, produced in an attempt to better understand

how solvers interact with my extensions to CoSolve as with the basic CoSolve system itself.

For some of the quantitative values, t-tests have been run to analyze the statistical significance of observed differences. The alpha value $\alpha=.05$ is used throughout this analysis. All t-tests listed here are two-tailed and unpaired, and all are comparing control versus experimental condition data. A two-tailed t-test was chosen in order to account for any difference between values, not just the assumed one. Given the small study size, these tests provide one perspective for the exploration of the data rather than providing conclusive evidence that particular features lead to better solving, etc.

Before investigating the differences between the control and experimental condition groups, I will present data on the usage of the Consultant, both full and mock; if the experimental group had never made use of the full-Consultant, then differences between the condition groups would have had to have been due to something else. Recall that while the experimental groups had access to the full-Consultant, the control groups had access to a mock-Consultant, which had an interface similar to that of the full-Consultant but provided only basic features. Both groups were incentivized to use the full-Consultant or mock-Consultant features given by being offered a point bonus if they, as a team, met a certain threshold of usage, which was met by all six teams.

Average Number of Full-Consultant Window Openings: The count of window opening events was tallied and is shown below. This is the average number of windows opened by each participant in the team.

Table 5.1. Data on Consultant (both full and mock) usage in the study. These numbers represent how many times over the course of the session Consultant windows were opened by the average solver.

Avg.	Cont. 1	Cont. 2	Cont. 3	Exp. 1	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.
Number Consultant Window Openings	7.33	1.67	3.67	38.0	72.6	16.67	4.22	42.4

The first six columns in Table 5.1 list the window opening values for each of the teams, the next two give control and experimental group averages. This data suggests that while the control group made fairly light use of the Consultant, the experimental group used it extensively. The main thing to conclude from this is simply that both groups did use the Consultant. It is difficult to tell from this whether the Consultant was appealing to use, as both groups were incentivized to use it, though we do get the impression that the full-Consultant features were more appealing than the mock-Consultant features, given that it was used about ten times as much.

In Table 5.2 below, some basic statistics are shown for each of the six teams. In addition to listing the values by team and control/experimental averages, the table also gives p values from a two-tailed unpaired t-test run on those values. Statistically significant p values, using an alpha of .05, are marked with asterisks.

The values examined in the table are as follows:

High Score: The value of the highest scoring node achieved by each team.

Nodes Created: The number of nodes created in each solving session.

Annotations Created: The number of annotations created in each solving session.

Table 5.2: Data for the six teams involved in the study regarding high scores and event counts.

	Cont. 1	Cont. 2	Cont. 3	Exp. 1	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.	p values
High Score	33087	26576	26865	31314	14848	20352	28842	22171	0.275
Nodes Created	178	399	320	264	392	101	299	252	0.682
Annotations Created	38	25	5	21	22	55	22	32	0.534

The data in Table 5.2 does not appear to show strong differences between teams in the control and experimental groups in terms of scores or node/annotation creation; the full-Consultant doesn't appear to have hurt or helped solvers in this regard. All teams appeared to made use of the CoSolve system; no team appeared to have ignored the activity, for instance.

All solvers created nodes (the fewest created by a solver was 28 in team Control 1) and all solvers but one (in Control 3) created annotations. Recall that teams were given 75 minutes total in which to work on the activity, except for Control 1 which had its time cut short, as explained earlier. The control and experimental teams averaged around 4.4 and 3.4 nodes per minute per team, respectively. The control and experimental teams averaged around .38 and .43 annotations per minute per team, respectively.

To find differences between the control and experimental groups, and to address the goals stated, we need to delve more deeply. The following section addresses each goal in turn.

5.3. Exploring Data Regarding Stated Goals

The goals stated earlier deal with more specific aspects of the data, which we will now consider in the following subsections. The subsections will also elaborate on additional metrics investigated. Equations and procedures for the calculation of these metrics can be found in chapter 4.

G1. Understanding Effects on Collaboration

One of my key intentions in designing the full-Consultant was to improve collaboration between team-members; as described in chapter 2, team interactions have been shown to influence group productivity and learning outcomes. Collaboration is, as is also discussed in chapter 2, a broad term. To make dealing with it more tractable, it has been broken down into different parts for investigation:

Fraction of Nodes that are Turn-taking: Although described in chapter 4, turn-taking will explained again here for convenience. One key indicator of collaboration in CoSolve is how closely solvers work together; whether they worked in completely

separate areas of the tree or perhaps ‘took turns’ with each other creating nodes on the same branch. A node creation event is considered a turn-taking event if the author of the child node is different from the author of the parent node. Nodes built as children of the root node (created by a researcher) are excluded for purposes of this count, as it is not turn-taking between team-members. To compute the fraction, a solver’s turn-taking score is divided by the number of nodes he created; this is done so we can compare these values across different users who created different amounts of nodes.

Turn-taking on the Solution Path: This is an additional way of looking at the turn-taking metric. As mentioned in the description of CoSolve, the solution path of a CoSolve tree is the path from the root to the highest scoring node in the tree. This ‘turn-taking on the solution path’ metric counts the number of turn-taking events that occur on that path, excluding children of the root (which was created by a researcher). It is useful as a metric separate from turn-taking as it captures the amount of turn-taking on road to the team’s ‘final answer’ (the highest scoring node), as opposed to miscellaneous explorations elsewhere in the tree. It is included to highlight the team’s level of interaction together towards the final answer. This is left as the average of the number of turn-taking events on the solution path, which will always be of length 30 (for purposes of this study, at least).

Solution Path Inequality: This metric provides a different perspective on collaboration on the solution path. For this metric, the number of nodes contributed to the solution path by each solver is counted up. The further this deviates from perfect equality, in which each of the three solvers adds 10 nodes, the higher this value will be. For a tree in which all solvers created an equal number of nodes on the solution path, this value will be 0. For a CitySim tree in which one solver created all nodes on the solution path, the value will be 40.

Average Annotation Length (in characters): Apart from simply looking at counts of annotations, the length of the annotations is also of interest, as it was possible that the experimental condition led users to write longer or shorter annotations. The average length, as opposed to the length of all annotations, is considered here in order to account for the fact that different users created different numbers of annotations. The

length is given as the number of characters and includes lengths of both the title and body (if provided) of the annotation.

Fraction of Annotations on Peer Nodes: An annotation can be placed on any node; if placed on a node whose author is different from the annotation's author, then this is referred to as a 'peer annotation.' The presence of a peer annotation, as opposed to a 'self annotation' (an annotation on one's own node) can be viewed as evidence that the author was to some degree aware of the operations of his team-mates. A solving session with only 'self annotations,' for instance, may be seen as evidence that the solvers were focused primarily on their own work, and not the work of their team-members.

Fraction of 'Reply' Annotations: 'Reply' annotations can be viewed as annotations that are responses to other annotations, suggesting that the author was aware (to some extent) of the prior discussion; this is based on Barron's observation that successful groups tended to have correct proposals emerge as part of an existing discussion instead of as unrelated statements [Barron 2003]. They were determined by going through the team's annotations and manually tagging replies, based on a coding scheme described in the appendix 3. A larger fraction of 'Reply' annotations could mean that the team had a more integrated discussion, as opposed to a series of isolated comments that may or may not have been viewed by other team-members.

Fraction of 'Strategy' Annotations: 'Strategy' annotations can be viewed as annotations describing or proposing strategy, or as solvers sharing their strategies with one another (or solvers describing the strategy as a note to themselves). They were determined using the coding scheme described in appendix 3. The coding scheme used draws a distinction between 'strategy' annotations, which directly communicate strategy and 'evaluation' annotations, which evaluate a node but don't convey strategy.

Fractions of Neutral/Positive/Negative Annotations: As described in chapter 4, a solver authoring an annotation has the option of marking it as positive, negative or neutral; they were told to use this to mark whether they believed the node was likely to lead to a good solution or not. These are listed as fractions of the total number of annotations created by a solver to take into account levels of annotation creation by different teams. For instance, 'fraction of positive annotations' is computed as 'number

of positive annotations created by team x' divided by 'total number of annotations created by team x.' Annotations that are not explicitly labeled as positive, negative or neutral are considered neutral for this metric.

In Table 5.3 the averages for these values are shown for each condition group. Statistically significant p values, using an alpha of .05, are marked with asterisks.

As shown in the table, the fraction of nodes that were turn-taking was higher for the experimental group, though this difference was not statistically significant. However, when looking at turn-taking on the solution path, conceptually the 'final answer submitted by the team,' the experimental group has a larger value (almost three times that of the control group) and this is statistically significant.

Now, recall that a turn-taking event occurs when a solver creates a node whose parent has a different author. An act of turn-taking then suggests that author of the new node viewed the node of a team-member, understood it and reacted to it. Turn-taking is then a useful metric for representing how tightly team-members worked together. All of this taken together suggests that the full-Consultant encourages teams to work more closely together. That said, this form of collaboration isn't necessarily always beneficial, as it could also lead to slower progress due to team-members waiting for others, but it is interesting nonetheless.

For additional perspective on solver turn-taking throughout the tree, consider the scatter plot of the fraction of nodes that were turn-taking nodes, organized by team, in Figure 5.7. From it, there is the suggestion of higher turn-taking for those in the experimental group; teams Experimental 2 & 3 have minimum turn-taking levels higher than any participant in the control group.

Table 5.3: Values for collaborative metrics observed in the study. The statistically significant p values (in the rightmost column) are marked with asterisks.

	Cont. 1	Cont. 2	Cont. 3	Exp. 1	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.	p values
Fraction of Nodes that were Turn-taking	.0843	.0652	.0219	.0492	.148	.248	.0571	.148	.204
Turn-taking on solution path	1	2	0	3	5	6	1	4.67	.0254*
Solution Path Inequality	28	34	40	26	18	20	34	21.33	.0398*
Avg. Annotation Length	52.84	58.52	36.83	83.62	33.43	60.14	49.40	59.1	.300
Fraction of Peer Annotation	.053	.36	0	.095	.181	.491	.138	.256	.511
Fraction of ‘Reply’ Annotations	.489	.328	.111	.167	.490	.491	.309	.382	.537
Fraction of ‘Strategy’ Annotations	.5	.24	.2	.667	.545	.618	.313	.610	.042*
Fraction of Neutral Annotations	.526	.28	0	.571	.682	.269	.269	.709	.066
Fraction of Positive Annotations	.447	.72	.8	.333	.318	.091	.656	.247	.037*
Fraction of Negative Annotations	.026	0	.2	.095	0	.036	.075	.044	.669

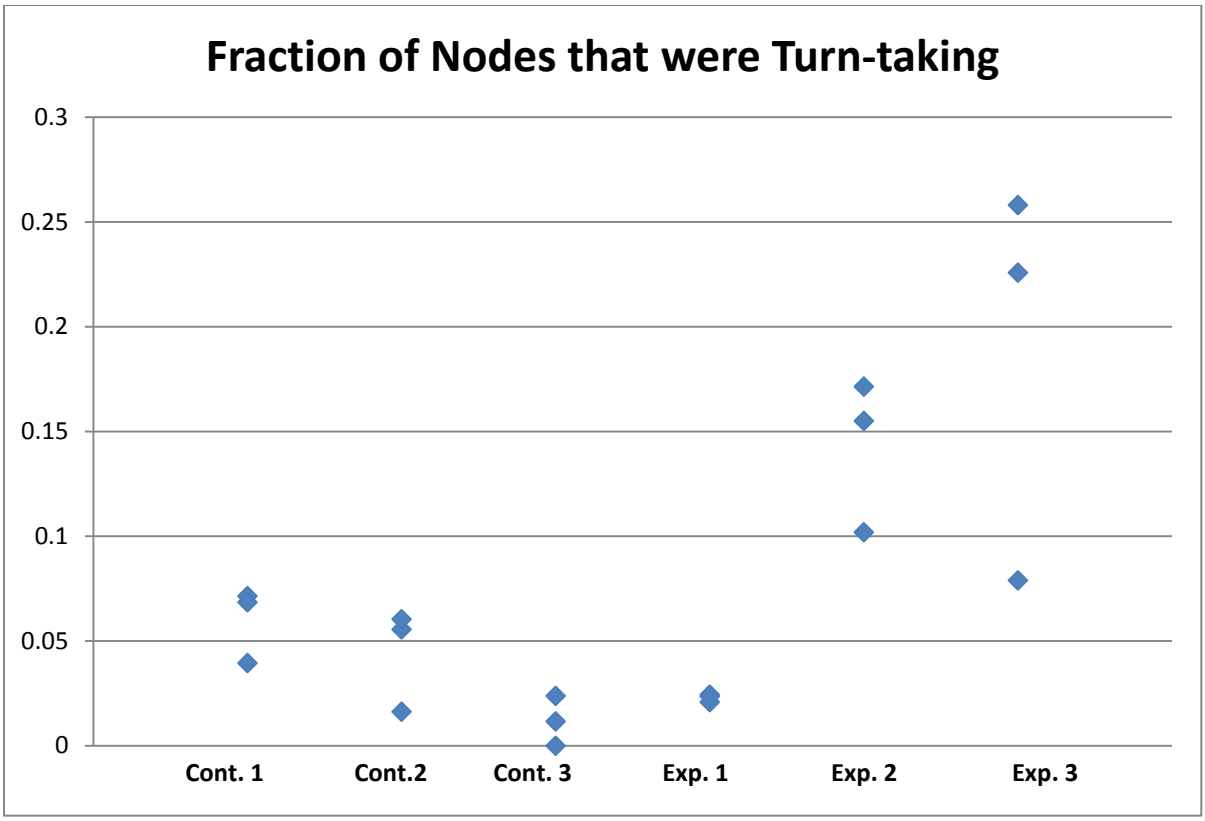


Figure 5.7: The fraction of nodes that were turn-taking shown for each participant, grouped by team.

The metric of solution path inequality provides additional evidence for the experimental group collaborating more on the solution path. The rest of the metrics shown in Table 5.3 are concerned with the placement and content of annotations. First, regarding the average length of annotations, this did not appear to differ greatly between the control and experimental groups; there is no suggestion that using the full-Consultant encourages solvers to create longer (or shorter) annotations.

Occurrence of peer annotations is an interesting metric, and the existence of an annotation on another's node provides evidence that the node's author was aware of the work of his team-mate, and that he was either sufficiently curious as to ask a question regarding it or sufficiently interested to comment on it. Both of these cases are give evidence of that author interacting with his team-mates. Regarding this metric, the experimental group has a higher portion of peer annotation, but the standard deviation

is high enough that it is unclear whether this is due to chance. This would be an interesting point to pursue in future versions of this study using a larger sample size.

Both conditions had similar portions of 'Reply' annotations; there does not appear to have been much of a difference in the length of conversations had by either group. Regarding 'Strategy' annotations, on the other hand, the experimental group had a statistically significantly larger portion of strategic annotations. This suggests that the experimental group's communications were more used for planning than for other purposes, such as evaluation of nodes and conversational aspects (saying thanks, making jokes, etc.). One possible interpretation of this is that, in addition to working more closely together in the tree (as seen in the turn-taking metric), the full-Consultant also encouraged team-members to work more closely strategically.

The groups did vary in a statistically significantly manner on how often they created different types (positive, neutral or negative) of annotations. The data shows that those in the control condition tended to create more positive annotations, and those in the experimental group created more neutral annotations, though the latter effect did not reach statistical significance. The cause of this is somewhat unclear. One interpretation is that the control teams were more optimistic about the value of the nodes they annotated. Another is that the content of the experimental group's annotations tended more towards planning and less towards evaluation; the large portion of 'Strategy' annotations (which tend to neutral in tone) in the experimental group supports this idea. Negative annotations were rarely used by any team; this may be due to team members not wanting to appear critical of each other.

G2. Understanding Effects on Problem-Solving

Another key goal of this study was to look at how the experimental treatment affected the problem-solving process. In CoSolve we can gather a fair amount of information on the problem-solving process by examining the structure of the tree. For instance, seeing that a node has numerous children indicates that the team explored several options around that node. Creating the first child of a node exemplifies initial exploration from that node, whereas creating more children demonstrates trying out

alternatives. Finally, information on how quickly solvers created nodes hints at how much thought and planning they gave to node creation.

Below are some of the metrics examined:

Fraction of Leaf Nodes: A leaf node in CoSolve is simply a node that has no children. The significance of a leaf node is that usually no solver decided to build off of it, for whatever reason, perhaps because a node was seen to be 'bad' and its path not worth further exploring, or perhaps because no options were possible, such as for an error node or an end-game node. It is also possible that a solver intended to build from it, but ran out of time. In any case, these are 'dead end' states that were never built upon. This value is the number of leaf nodes divided by the total number of nodes.

Top-heaviness: This metric indicates where the bulk of the tree's nodes are. A high value (with a maximum of 1) indicates that the majority of the nodes were created toward the top of the tree near the root; that the tree is 'top-heavy,' which suggests that the team was more concerned with exploring more in the early stages of the game. A low value (with a minimum near 0) indicates that the majority of the nodes were created lower in the tree, which suggests that the team was mostly concerned with exploring the end stages of the game. The definition of this metric is provided in chapter 4.

Narrowness: The narrowness metric indicates whether the tree is broad (a value near 0) or narrow (a value near 1). A broad tree suggests the exploration of many possibilities at each level, whereas a narrow tree suggests a more focused effort on fewer branches. The definition of this metric is provided in chapter 4.

Number of Same-Author Connected Regions: A *same-author connected region* is a connected region of the tree where all the nodes have the same author. A small number of such connected regions would indicate that solvers tended to work mostly in their own areas, whereas a large number would indicate that solvers interacted more in node creation, and did not maintain much 'private' space in the tree. This metric can be viewed to be, in some ways, a collaborative metric, but is listed here in problem-solving as it also indicates the 'size' of consecutive regions created by a solver.

Time in Seconds to Create First Three Nodes: The other metrics in this section look only at the structure of the tree, but this metric looks at time: how long it

took each solver to create his first three nodes. The number three was chosen somewhat arbitrarily as a point at which the construction had begun in earnest. A smaller value for this metric would indicate that solvers began creating numerous nodes very quickly, whereas a larger number would indicate that they created nodes more slowly at the beginning, possibly due to creating annotations or observing the actions of their team members instead. This value is an average over all the members of the team, and is given in seconds.

As seen in Table 5.4, most of these values appear to be similar between the control and experimental conditions, at least in terms of statistical significance. This matches anecdotal observations of the trees and problem-solving processes themselves: while they do differ, the variations do not seem to be based on the condition group. Looking at the screenshots of the final solving sessions, shown earlier in this chapter, it is difficult picking out visual characteristics for classifying a zoomed out view of the tree into either the control or experimental groups. There simply did not appear to be strong characteristics that were visibly different between the groups.

Table 5.4: This table shows data obtained for several problem-solving metrics in the study. Statistically significant differences have their p values marked with asterisks.

	Cont. 1	Cont. 2	Cont. 3	Exp. 1	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.	p values
Fraction of Leaf Nodes	.27	.19	.22	.22	.15	.32	.23	.19	.459
Top-heaviness	.54	.44	.43	.39	.35	.62	.47	.45	.86
Narrowness	.16	.80	.11	.14	.10	.23	.12	.16	0.424
Number of Same Author Connected Regions	15	26	7	13	58	25	16	32	.333
Time to Create First Three Nodes (sec)	350.7	202	67	171	443.7	827	206.56	480.6	.256

The first two metrics, looking at the number of leaf nodes and top-heaviness, look at whether the full-Consultant had an impact on the tree structure, which it doesn't appear to have had. The last metric, 'Time to Create First Three Nodes,' while not statistically significantly different, is suggestive of the control teams working much more quickly than the experimental teams, which matches anecdotal observations. Figure 5.8 backs up this suggestion by showing the early trees constructed by the different teams.

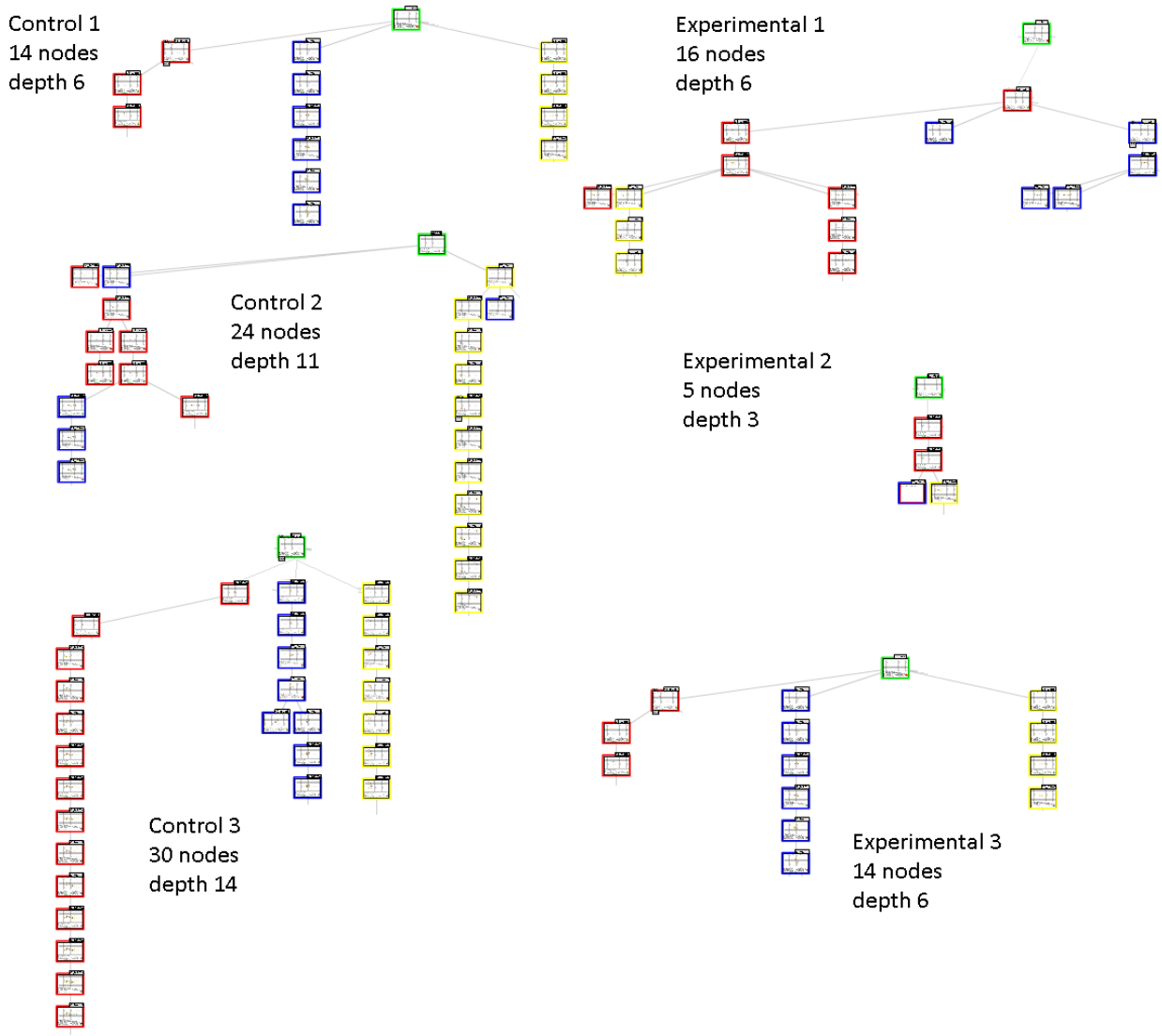


Figure 5.8: Each team's solving session after they've been working on it for five minutes. Control teams have their trees shown on the left side, and experimental teams have their trees shown on the right. Each tree is labeled with its size and depth at this point. The trees in the figure were created using the History Tool, another tool I created for the analysis of solving sessions. In it a user can play, fast-forward, rewind solving sessions, allowing for examination of patterns over time. While not conclusive, these observations are suggestive of the differences in the groups' initial speeds.

Figure 5.8 shows what each solving session looked like after the first five minutes, with the control teams on the left and experimental teams on the right. Notice how the control teams' trees tend to be larger, and the least deep of the control trees is

the deepest of the experimental. The first five minutes, again chosen somewhat arbitrarily, supply some insight into solvers' initial actions, before they settled into routines. One can also see the beginnings of turn-taking patterns: two control trees, and only one experimental, display no turn-taking, as solvers just work on their own branches.

G3. Understanding Effects on Vocabulary

One of the initial aims of the CoSolve system was to help teach solvers a common design process vocabulary; the idea was that graph terms like 'tree,' 'branch,' 'node,' et cetera could be used as part of a common language to discuss the process of design for a team whose members had different technical backgrounds. Certainly this knowledge should help solvers communicate regarding the CoSolve system and may help solvers pick up terminology relevant to trees and design as well.

To evaluate the effect of the full-Consultant on the usage of CoSolve vocabulary, the study was designed with pre and post tests evaluating knowledge of CoSolve terminology so that learning gains, if present, could be observed. The vocabulary words tested were mostly standard tree terms, though some had slightly altered meanings in CoSolve, so a strong knowledge of trees in graph theory wouldn't guarantee a perfect score. The test was divided into two parts, each of which consists of six questions. The first half, 'vocabulary description,' required the participant to match different terms (tree, state, etc.) with their descriptions. The second half, 'structure,' asked the participant to examine a tree and answer questions regarding its structure ('What is the depth of node x?' or 'How many children does node x have?).

After solvers had taken the pretest, the terms were used and described (though not formally defined) for them in an introductory activity that taught them about various aspects of CoSolve. The terms are also used in places in CoSolve and in the full-Consultant interfaces, so usage of aspects of CoSolve could improve this knowledge. The posttest was identical to the pretest in content. The data for the pre/post tests only include teams Cont. 2, Exp. 2, Cont. 3 and Exp. 3; the tests underwent modification after running teams Cont. 1 and Exp. 1, due to problems observed in the early version of the test. As the data for Cont. 1 and Exp. 1 aren't directly compatible, they are left

out of this analysis. T-tests are not given for these metrics, due to the exclusion of two teams.

Metrics relevant to vocabulary are described below. They include pre/post test scores for the overall tests as well as for the 'description' and 'structure' subdivisions, and the learning gains computed from all of these.

Overall Pretest / Posttest Scores: There were 12 items on the pre and post tests, each worth one point.

Description Section Pretest / Posttest Scores: The first half of the questions on the pre/post test involved matching terms with their descriptions; these metrics list their values on the pre and post test. The questions in this category were worth six points on each test.

Structure Section Pretest / Posttest Scores: The second half of the questions on the pre/post test involved looking at a tree and answering questions regarding structure ('What is the depth of x?' or 'How many children does x have?); these metrics list their values on the pre and post test. The questions in this category were worth six points on each test.

Overall Learning Gains: The vocabulary learning gain of a solver is computed as their overall posttest score minus their overall pretest score. This value is the average of solver's learning gains.

Description Section Learning Gains: This metric is the sum of the learning gains on the 'vocabulary description' questions.

Structure Section Learning Gains: This metric is the sum of the learning gains on the 'vocabulary structure' questions.

Table 5.5: The data of the pretests, posttests and learning gains (defined as post score minus pre score) for the question categories of ‘description’ and ‘structure,’ as well as overall. Teams Control 1 and Experimental 1 are left out, as they used an early version of the tests, and so the data from those tests were not easily compatible.

	Cont. 2	Cont. 3	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.
Overall Pretest Score	8.33	8.33	6	9	8.33	7.5
Description Section Pretest Score	3.67	3	3.33	3.67	3.33	3.5
Structure Section Pretest Score	4.67	5.33	2.67	5.33	5	4
Overall Posttest Score	10.67	10	10.33	12	10.33	11.17
Description Section Posttest Score	5	4.67	4.33	6	4.83	5.17
Structure Section Posttest Score	5.67	5.33	6	6	5.5	6
Overall Learning Gains	2.33	1.67	4.33	3	2	3.67
Description Section Learning Gains	1.33	1.67	1	2.33	1.5	1.67
Structure Section Learning Gains	1	0	3.33	.667	0.5	2

Table 5.5 shows that the learning gains of both groups were similar, and were on average positive for both the control and experimental groups; solvers did learn from the experience, though it could have been from the introductory materials as well as the activity. In terms of individual learning gains, accessible by a link in appendix 5 due to space issues, it can be seen that one participant had a learning gain of zero, and two had negative learning gains; all three of these participants were in the control group. This is interesting and encouraging for the full-Consultant, and may be due to those in the control group guessing on both the pre and post tests, whereas those in the experimental group had the correct answers reinforced by the full-Consultant.

Also note that, while the experimental group had lower scores on the structure section pretest than the control group, all solvers in the experimental group had a perfect score on the structure section in the post-test (visible in Table 6.5, since the averages for Experimental 2 & 3 on the post-test structure section are both six, the maximum score). This provides additional evidence that using the full-Consultant may help reinforce ideas and concepts through its usage.

The difference between the control and experimental groups for the 'Vocabulary Structure Learning Gains' metric seems large, and so invites more inspection. Investigating the questions in that category, it turns out that there was one structure question that showed high learning gains for the experimental group; this question (#9) asked participants how many children a certain node had in the illustration. In the experimental group, half the participants missed it on the pretest and all got it right on the posttest (as mentioned, all experimental group participants got a perfect score on the posttest structure section). In the control group, on the other hand, one participant got it wrong on the pretest, and two got it wrong on the posttest; the learning gain was actually negative for the control group on that question. The data on this question is shown in Table 5.6.

Additional work was done looking at solvers' usage of keywords in annotations in order to find more evidence of increased vocabulary understanding. The occurrence of keywords (contained in the pre and posttests) in annotations was tallied up and compared as 'keywords per annotation.' The data shows that these values were

virtually identical between the groups, and so the full-Consultant doesn't appear to have influenced this. The data for this can be found in the appendix 5.

Table 5.6: Pre and post test average scores are shown for control and experimental groups on question #9, which asks the participant how many children a node has in a given illustration. The exact question asked can be seen in appendix 5.

	Cont. 2	Cont. 3	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.
Question #9 Avg. Learning Gains	0	-.33	.67	.33	-.167	.5
Question #9 Avg. Pretest	.67	1	.33	.67	.833	.5
Question #9 Avg. Posttest	.67	.67	1	1	.67	1

The learning gains in Table 5.6 suggest that the full-Consultant can help solvers learn and reinforce CoSolve vocabulary terms; in this case, the use of 'children' to refer to nodes created from a given parent node. It is far from conclusive, but it is still of interest in that it hints at the potential to reinforce information through the full-Consultant interface. This is encouraging in that the full-Consultant is not an explicit tutorial, it is merely information on-demand, and yet it still has some learning potential.

It would be interesting to study this in more depth in its own right, and look at conveying additional concepts. Note that the effects observed here are over a very short time frame (within a couple of days of the pretest and activity), and so we can't say anything about the evidence regarding long-term learning gains.

G4. Understanding Effects on Metacognition and Team Awareness

Aspects of metacognition have been of interest since the early stages of CoSolve, and adding openness had the potential to improve metacognition of solvers using the system. Related to metacognition is the idea of 'team awareness,' as it involves individuals 'knowing the mind of the team.'

Metacognition, though important, is somewhat elusive, both in terms of pinning down a definition and in terms of measurement. At its core, metacognition is 'thinking about thinking,' and pulling a quantitative metric from one's thinking processes can be difficult. Adding prompts to the effect of "what are you thinking right now" could have been used, but would have slowed things down and likely interfered with the task being performed. Instead I opted for gathering hints of metacognitive from solvers' annotations and from wrap-up questionnaire items, so that the activity itself would not impacted by the measurement.

One way to examine metacognition in solvers in CoSolve is to look for instances of reflection, where a solver produces evidence that he consciously thought about his own knowledge. An example of this could be if a solver created an annotation saying that he had a better developed problem-solving technique than his team-mates. A tagging scheme, described in appendix 3 was applied to the annotations in order to look for evidence of metacognitive activity; annotations were tagged as 'Meta' if they suggested that the author was reasoning about his knowledge.

Table 5.8: Data of tagging annotations as evidence of metacognition. This shows what fraction of a team's annotations were tagged as metacognitive in nature.

	Cont. 1	Cont. 2	Cont. 3	Exp. 1	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.	p values
Fraction of 'Meta' Annotations	.026	0	0	.048	0	.018	.009	.022	.468

As seen in the table, very few annotations were tagged as being evidence of metacognition; only three annotations total were marked as metacognitive (one in the

control group and two in the experimental group, each on different teams). As such, there is little to say about how metacognition is affected by the full-Consultant. The instances of metacognition observed were all statements expressing the annotation author's observation of a deficiency of their own knowledge, which is an instance of self reflection.

Another possibility for assessing metacognition is in the post-activity interviews, where participants were asked about their thoughts and experiences with the system. However, this was done at the prodding of the researchers, and so counting occurrences would not be meaningful; the questions asked would themselves confound the data.

To measure team awareness and accuracy of knowledge of solvers, two questions on the wrap-up questionnaire asked participants to guess what percentage of the nodes and annotations in their team's solving session were their contribution: "Roughly what percentage of the nodes/annotations created were yours, when compared to those of your team-mates?" This value was then compared against the actual percentage of nodes/annotations created by that solver. A measure of the inaccuracy of these guesses, and so an indication of their awareness of their own contributions, is shown in the following table.

Table 5.9: Data of how accurate solvers in the different condition groups were at guessing their node and annotation contributions to their solving session. Values listed are percent errors. The percent error is computed as = | 100(guess-actual)/actual |.

	Cont. Avg. Percent Error	Exp. Avg. Percent Error
Node Guess	15.96%	11.65%
Annotation Guess	30.67%	29.23%

As shown, the numbers for the experimental group were slightly more accurate when guessing their contributions in terms of number of nodes created. The two groups had very similar error rates regarding annotation guesses. Interestingly, the guesses for both groups were much more accurate when guessing node contributions, despite the

fact that there were substantially fewer annotations than nodes, which would have made that count easier to keep track of.

All in all, no strong differences showed up between the control and experimental groups for any metrics related to metacognition. Tagging of evidence of metacognition resulted in very few 'hits' and knowledge accuracy only varied slightly between groups.

G5. Understanding Solvers

Finally, one of the goals of this work was to try to understand solvers in CoSolve: what actions they performed; what actions they avoided, what they liked, what they disliked, etc. The answers to this series of questions could then be used to correct disruptive features in CoSolve, and add in new features that could make using the tool more pleasant and more productive. Some of these have been partially answered using earlier data (such as frequency of node creation, annotations, etc.) but will here be examined in more depth.

To understand how solvers used and reacted to CoSolve, three areas were examined:

- Profiles of solvers' usage of the system
- Questionnaire data collected on how well they liked (or disliked) certain features
- Information on solver behavior as a function of time

To better understand how solvers used the system, profiles were built describing solver behavior. These profiles were built by looking at bigrams of actions performed by solvers: pairs of consecutive actions performed. From this I constructed transition tables for solvers, describing the likelihood of performing action β given that that solver had just performed action α. This provides a rough model of how solvers acted while solving. For instance, if we observe events x, y and z, in that order, we would have one instance of the (x,y) bigram and one of the (y,z) bigram. This gives us information on what a solver will do, given his previous action.

Tables 5.10 and 5.11 show transition tables showing $P(\beta \mid \alpha)$, the conditional probability that action β will be the next action performed given that α was the last action performed. The tables shown are averages for each of the two condition groups.

All solver actions were classified into one of the following 10 categories:

- Annoself: Placing an annotation on one's own node
- Noderoot: Creating a node that is a child of the root
- Annopeer: Placing an annotation on a team-mate's node
- Nodescon: Creating a node that is the child of the last node created by this solver (scon stands for 'self consecutive')
- Nodesncn: Creating a node that is the child of a node by the author, but not the last node created by the author (sncn stands for 'self not consecutive')
- View: Changing the view of the CitySim game
- Annoroot: Placing an annotation on the root node of the tree (which is always created by a researcher)
- Refr: Refresh event
- Nodepeer: Creating a node that is a child of a team-member's node
- Cons: Use of some feature of the Consultant (either mock or full, depending on condition)

Table 5.10 shows the profile (in terms of transition probability percentages) for the average solver in the control condition. Table 5.11 shows the profile (in terms of transition probabilities) for the average solver in the experimental condition. More information on these bigrams and the method used to compute them can be found in appendix 5.

Table 5.10: This table shows the profile (in terms of transition probabilities given as percentages) for the average solver in the control condition. To look up P(β | α), the likelihood of performing action β given that the solver last performed action α, find the column that matches α and the row that matches β. The diagonal P(α | α) is highlighted for convenience.

	anno self	node root	anno peer	node scon	node sncn	view	anno root	refr	node peer	cons
annoself	6.0	0.0	7.8	4.4	2.4	0.3	4.4	2.7	4.8	2.8
node root	3.9	0.0	7.8	0.2	0.0	0.0	16.8	1.1	1.1	6.7
anno peer	2.2	0.0	7.8	0.0	0.0	0.1	4.4	1.4	1.1	3.7
node scon	22.8	59.3	7.8	46.2	37.3	14.7	8.2	27.8	44.8	8.2
node sncn	10.3	0.0	7.8	10.9	23.9	3.5	4.4	6.9	6.7	15.3
view	11.6	0.0	14.6	11.5	11.6	70.3	4.4	27.3	4.9	17.5
anno root	2.2	0.0	7.8	0.1	0.0	0.0	16.8	6.0	1.1	7.4
refr	30.6	40.7	16.4	24.9	22.8	7.4	22.9	10.5	23.5	16.6
node peer	3.1	0.0	13.3	0.2	0.0	1.7	5.7	3.5	10.9	8.7
cons	7.1	0.0	9.0	1.7	1.9	1.8	11.9	12.7	1.1	13.2

There are a number of similarities between the average profiles of the two condition groups: both have almost a 50% chance of creating a consecutive self node immediately after having created a consecutive self node, for instance. There are also some substantial differences. For instance, the experimental group is almost four times more likely to use the Consultant given that using the Consultant had been their previous action, and the experimental group was over 20 times more likely to use the Consultant immediately after creating a peer node, possibly to see how the turn-taking action affected the values shown in the Consultant.

Table 5.11: This table shows the profile (in terms of transition probabilities shown as percentages) for the average solver in the experimental condition. To look up P(β | α), the likelihood of performing action β given that the solver last performed action α, find the column that matches α and the row that matches β. The diagonal P(α | α) is highlighted for convenience.

	anno self	node root	anno peer	node scon	node sncn	view	anno root	refr	node peer	cons
anno self	1.1	2.2	4.9	1.7	2.7	1.1	3.3	5.3	4.6	0.9
node root	0.0	2.2	2.2	0.2	1.1	0.0	3.3	2.1	0.0	1.5
anno peer	2.8	2.2	3.0	0.4	3.9	1.4	3.3	3.0	1.9	1.0
node scon	15.1	35.6	18.6	44.5	38.4	12.2	12.6	10.7	17.9	8.7
node sncn	2.8	2.2	3.8	5.4	16.7	2.2	3.3	2.6	0.0	1.8
view	13.8	7.8	6.9	15.8	9.9	70.7	3.3	31.4	22.4	11.5
anno root	0.0	7.8	2.2	0.1	1.1	0.4	11.5	0.6	1.4	2.1
refr	39.5	21.7	41.9	20.4	13.6	7.2	25.9	19.6	19.0	17.9
node peer	6.8	2.2	10.9	0.3	1.1	2.6	5.6	10.2	6.8	3.1
cons	18.3	16.1	5.4	11.1	11.4	2.1	27.8	14.7	25.9	51.3

While rough, these models of solvers in the control and experimental groups offer some insight into how the tools were used. These models represent, to the best of my knowledge, the first real attempt to model solvers in CoSolve, and so can help guide future changes and additions to the system.

Another point of interest is how often each of these 10 actions, used in the transition tables above, were performed by solvers in different groups. In table 5.12 the prior probabilities are giving, showing how often each action was performed overall. This prior probability information is also produced by the tool that generates the bigram transition tables.

Many of the probabilities in the table are similar between the two conditions. Some of the interesting differences include the higher likelihood of the experimental group creating peer node and annotations, as well as using the Consultant. This provides additional support for earlier evidence that the experimental group was more likely to work on the nodes of peers.

Table 5.12: The prior probabilities of performing each action, shown by condition group. The probabilities are listed as percentages.

	Control Condition	Experimental Condition
anno self	2.39	2.28
node root	0.92	0.58
anno peer	0.54	1.47
node scon	30.61	22.27
node sncn	9.55	3.81
view	32.34	34.63
anno root	1.29	0.62
refr	15.71	15.22
node peer	1.70	3.22
cons	4.95	15.90

The wrap-up questionnaire and interview asked questions of what aspects were helpful or harmful in using the tools given. The common complaints given by solvers did not appear to be tied to the control or experimental groups; these are primarily user interface issues, including:

- Lag: Delays between performing actions and seeing results, sometimes up to a few seconds. These delays increased as the tree grew in size. This is, and has been, a point of concern, as it definitely detracts from satisfaction with the system. This was consistent across groups.
- Navigational issues: The navigation controls for CoSolve involve dragging the mouse to pan and clicking icons to zoom in or out, which many solvers reported as somewhat cumbersome. This too was consistent across groups.

- Bugs: With the software being a research prototype, there are still a number of bugs present, causing strange interactions and requiring the occasional browser refresh. These too were consistent across groups.

All of these no doubt impacted how the system was used; for instance, it seemed that the lag at the end of the session, when the tree was large, slowed down problem-solving efforts. Since the problems were consistent for all teams, it is unlikely that these biased the differences seen between the control and experimental group. Anecdotally, when solvers were asked what features they would like to see added to the interface, a couple of solvers in the control group mentioned features that were similar to the 'score' highlighting found in the full-Consultant, to which they did not have access.

The wrap-up questionnaire was also designed to get data on how well participants liked individual aspects of the system, as well as how well they liked the experience overall. Table 5.12 below shows comparisons between teams for different groups of questions. The items shown in the table are aggregates created by combining together a series of positive or negative questions for each topic. The specific questions can be found in the appendix 5. All questions in the table below were Likert scale questions where the answers ranged from 1 (Strongly Disagree) to 5 (Strongly Agree).

Tree Visualization Aggregate: This aggregate looks at how well the participants viewed the tree visualization, when considered separately from the full-Consultant, menus and other interface items. The aggregate is made up of four questions, two positive items (where a high value would indicate that participants liked aspects of the tree visualization) minus two negative items (where a high value would indicate that participants disliked aspects of the tree visualization).

CoSolve Interface Aggregate: This aggregate looks at how the participants liked the controls for panning, zooming, node creation and annotation creation. This aggregate is made up of three positive items and two negative items.

CoSolve Consultant Aggregate: This aggregate looks at how well the participants liked the features of the Consultant (either mock or full), including

highlighting, team and individual statistics. This aggregate is made up of three positive items and three negative items.

Table 5.13: Three aggregations from questionnaire items regarding how well participants liked different aspects of the system.

	Cont. 1	Cont. 2	Cont. 3	Exp. 1	Exp. 2	Exp. 3	Cont. Avg.	Exp. Avg.	p values
Tree Visualization Aggregate	3.33	2.33	2.33	2.67	.33	1.33	2.67	1.44	.180
CoSolve Interface Aggregate	6.67	5.67	5.67	8.33	6	5.67	6	6.67	.500
CoSolve Consultant Aggregate	.67	1.33	1.67	5	4	3.33	1.22	4.11	.00699*

From the data in this table we see that, while was not a significant difference in how the different condition groups felt about the tree visualization and CoSolve interaction interface, the experimental group found the CoSolve Consultant to be much more helpful than the Control group. Recall that, while the experimental group had more features available in the Consultant, both had access to some Consultant features, such as ‘Created by’ highlighting. This suggests that these additional features were seen to be helpful.

Another way to examine solver behavior is as a function of time. The data described so far has primarily been collections of values over the course of the entire solving session, but this leaves unanswered the question of how behaviors changed over time. Figure 5.9 shows three graphs showing how the control and experimental groups, on average, differed over time.

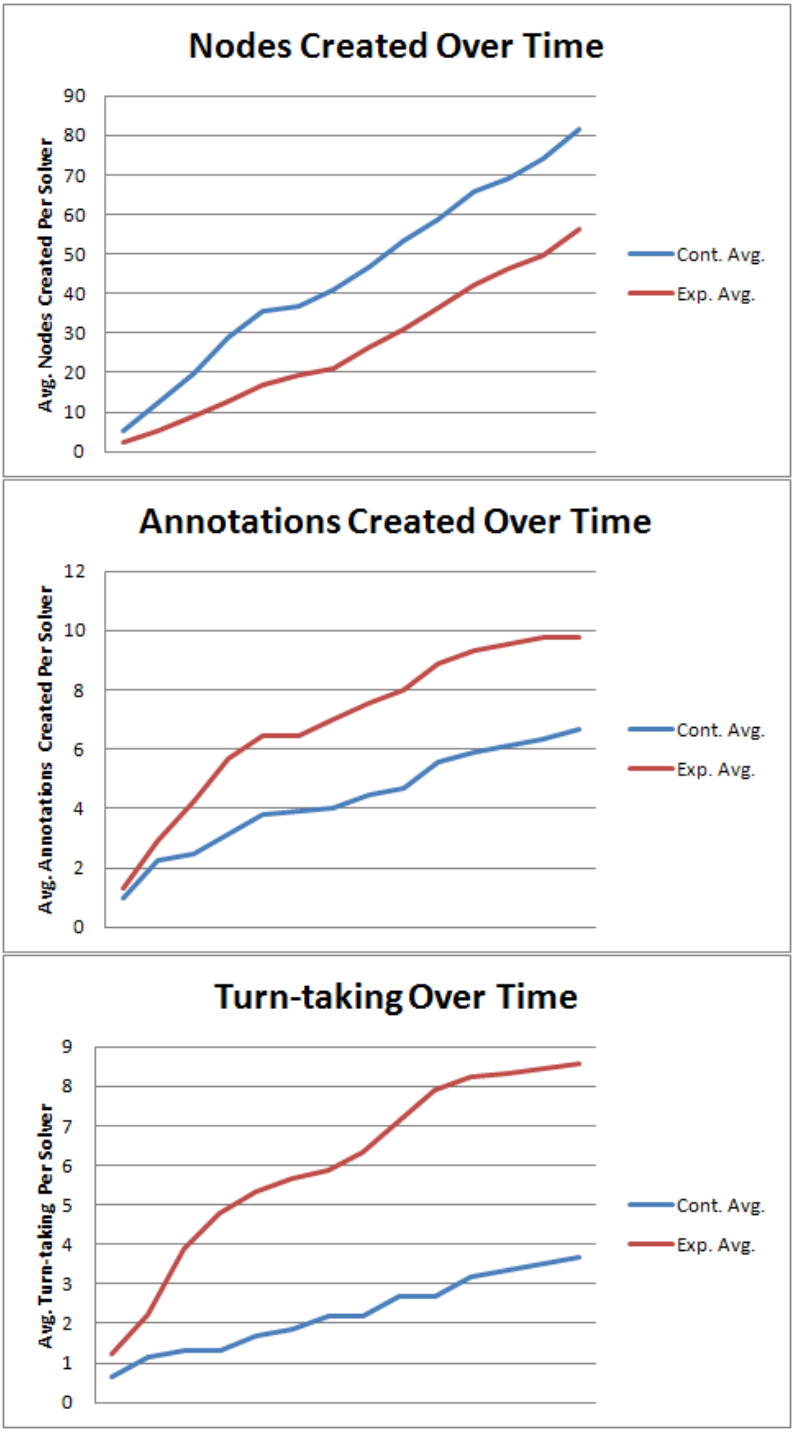


Figure 5.9: Graphs of three metrics of solvers over time, from the beginning to the end of the session. The diminished rates around the 1/3 time mark are in large part due to the breaks, during which no activity took place. The timing of the second break, near the 2/3 time mark, is less regular, as the first break length differed between teams, and so the beginning of the second break likewise differed.

Looking at the graphs, the latter portions of each appear to be nearly parallel. That is, the rates of node creation, annotation creation and turn-taking appear to be similar at the end of the session between the control and experimental groups. On the other hand, at the beginning of the session, the control group has a much steeper line for node creation, and the experimental group has steeper lines for annotation and turn-taking. One possible reason for this is the time crunch; anecdotally, we saw solvers rushing to finish towards the end of the session. This is supported by the fact that, on a Likert scale item on the wrap-up questionnaire, all participants but one indicated that they would have liked more time with which to complete the activity. It is possible that, given more time we would see more consistent graphs.

5.4. Study Summary

From the data presented here we see that the experimental group made use of the full-Consultant, and that everyone made use of CoSolve itself, and that there were some statistically significant differences between the control and experimental groups. The greatest differences were seen in the area of collaboration, where the experimental group demonstrated tighter interaction, especially on the solution path. While increased collaboration of this sort may not be completely free from disadvantages–such as slower progress, though this result was not confirmed statistically in the study–it seems to have led solvers to generate solutions that are more the product of the team than those that lacked openness.

Problem-solving metrics, which look at different shapes and styles in the development of the tree, turned out to not differ much: while large differences were seen between teams, they did not appear to map to one condition group or the other. There was, however, the suggestion that solvers in the experimental group created nodes at a slower rate, at least initially. Overall, the full-Consultant does not appear to have had much of an effect on how solvers approached problem-solving.

Regarding effects on vocabulary, including learning gains observed from the pre and post test, there were no large differences overall, but there was an interesting

difference in the learning gains between groups for one question, asking the participant to identify how many children a certain node has. This result is encouraging, as it suggests that seeing this term reinforced in the full-Consultant helped solvers learn it; there is then hope that other terms could be further emphasized with some chance of success.

Metacognition was investigated two ways in this study: tagging of annotations showing evidence of metacognition, and evaluating accuracy of guesses of contributions. Neither result showed strong differences between the control and experimental groups. If the openness of the CoSolve Consultant has an effect on measures of metacognition, they are smaller or of a different nature than those sought.

The final goal is that of understanding solvers in Cosolve, and the results from this study represent a substantial step on the path to this, allowing me to build models of use for solvers in each condition group. In addition, the results indicate that solvers did find the full-Consultant more helpful in the experimental group, that these features were appreciated by solvers.

Taken all together, this data provides evidence for the claim that the full-Consultant can influence solvers in several ways, many of them positive. And regarding the negative effects seen, none appear to be severe, though it would be prudent to investigate this more in future work. At the same time, there were a number of expected influences were not seen, and it is likely that, with deeper investigation, the details of these and additional effects could be determined. Overall this work offers a number of insights into CoSolve solvers.

In addition, through this study and this analysis I have created a collection tools to assist in the understanding of CoSolve activity. Various metrics of problem-solving and collaboration behavior in CoSolve, tools for generating graphs and analyzing results, and CoSolve log processing are all additional contributions of this work. The tools and metrics created are almost entirely independent of the particular problem solved (the CitySim game, in this case). The different analysis tools used to examine

solvers' behavior from different angles seem useful at understanding solvers using the CoSolve Consultant, but other solvers of CoSolve as well.

As one final point here, it should be stated that the results seen here are in part the products of the study details: the CitySim problem, the group size, the time limits given, etc. My impression is that the time limits (which were necessary for the logistics of running the study) had the strongest impact, and that they reduced collaboration and possibly reduced usage of the Consultant: the leisurely and often heavily collaborative group interactions seen at the beginning of some sessions seemed to give way to a more product-driven result as time became more and more scarce.

Along these lines, the study focuses on solver behavior shortly after being introduced to the tools. It is likely that effects after prolonged use would yield additional information. Additionally, examining solvers under other circumstances could yield other insights; the next chapter, which includes a discussion of future work, examines this idea.

Chapter 6: General Discussion and Conclusion

6.1. Summary of Thesis and Findings

This work has presented two primary contributions of my work:

1. The CoSolve Consultant, a suite of tools for adding openness to CoSolve, and an evaluation of it
2. A suite of tools for the analysis of CoSolve problem-solving sessions

The first contribution allows solvers to look more closely at their CoSolve solving sessions and improve awareness of their behavior and the behavior of their teammates. The study showed evidence of closer collaboration as a result of the CoSolve Consultant and increased discussion of strategy.

The second contribution is the creation of tools for the analysis of problem-solving sessions and of solvers. Prior to the creation of these tools, it was unclear as to how CoSolve solving sessions should be evaluated; now there exists methods for examining the collaboration, problem-solving, history and behavior of solvers. I would not go as far as to argue that these are the perfect tools for the job, or that these are the correct aspects of the solving process to analyze, but these tools are able to show us much more than we had been able to see before. There are, of course, questions regarding the relevance of these techniques beyond CoSolve; this topic will be addressed in section 6.2.

6.2. Extension Beyond CoSolve

The contributions of this work may seem, at a glance, to be restricted to the CoSolve system, but I argue that this is not the case. One of CoSolve's strongest points is the presentation of the problem-solving history as a state-space search tree. Taking this further, one could imagine applying this perspective to problem-solving outside of CoSolve. Take, for instance, someone trying to solve the Eight Queens problem on a physical chessboard. If another person were to observe her, that person could follow the moves, take notes, and sketch the problem-solving history as a search tree. In other words, most problems being solved actually go through a similar problems of

making moves, undo-ing moves, starting over and backing up; CoSolve just visualizes this and makes it easier to ‘move around’ this space, in addition to some other features.

Therefore it seems reasonable that most of the open features and analysis tools could be applied to collaborative problem-solving outside of CoSolve. A search tree is being generated by someone solving the Eight Queens problem, even if she isn’t aware of it, and so the process can be assigned *narrowness* and *top-heaviness* metrics, for instance. And the collaborative solving of the Eight Queens problem, if a team of solvers sat down with separate boards to solve it together, could be viewed in terms of turn-taking. Of course, not all real problem-solving is as clean as a tree of discrete nodes with explicit states and authors, and so watching two solvers discuss how to handle the Eight Queens problem would likely be open to interpretation as to what tree represents it. Nonetheless, given some reduction of the problem-solving process into a tree of nodes and authors, the metrics and tools presented here could be applied.

6.3. Future Work

6.3.1. Studying CoSolve Usage

There remain a number of questions to be answered regarding the usage of CoSolve, independent of the Consultant, but the most relevant in my mind is the question of how CoSolve compares to more standard techniques for collaborative problem-solving. At the moment we have data on usage of CoSolve and the CoSolve Consultant, but it would be desirable to have proof that CoSolve does offer problem-solving or collaboration advantages over existing computer-based collaborative techniques. One way to examine this would be run a study where teams of three worked to solve problems, perhaps something like CitySim, and the experimental group used CoSolve to do so, while the control group used a more traditional ‘single state’ view, in which each solver sees a single state and can apply an operator to it or perhaps ‘undo’ the last move, backing up a level. To ensure a fair comparison, the single state view could be outfitted with controls for saving and loading states, and for sharing states between solvers. This more closely resembles traditional problem-solving in that only one state is viewed at a time and that ‘backups’ can be kept and used (analogous to previous nodes in the tree that can be jumped to), but must be made explicitly.

It is my belief that solvers using CoSolve would come up with better solutions when solving for CitySim, and this proposed study could shed some light on the answer. But the aim of the study would be more than coming up with a simple yes or no answer to the question of whether CoSolve was more effective than traditional methods: the goal would be to see how they differed, and perhaps tease out characteristics of problems suitable for one or the other mode of solving.

6.3.2. Refining the CoSolve Consultant

This thesis has presented evidence that the CoSolve Consultant can benefit solvers, primarily in terms of increased collaboration. There are, however, still additional areas that can be addressed, and, in light of study observations and feedback, there are refinements that can be made. For instance, one common behavior seen in the study was that of solvers comparing nodes along a row in order to find the 'best' node at that level. One way to make it more convenient for solvers to perform this would be to add a highlighting technique that makes such row-based comparisons.

While there are more features that could be added, there are some existing features that could be improved. Some windows and some highlighting techniques were rarely used; solvers apparently did not find them useful. Removing them or modifying them to serve the same role but be more useful could benefit the tool. A study aimed directly at observation of usage of different aspects of the Consultant could help answer this.

6.3.3. Using the CoSolve Analysis Tools to Examine Other Solving Sessions

The analysis tools presented in this thesis were created in the context of answering questions observed about solver behavior in the study conducted. It would be interesting to apply them to other solving sessions and other problems. One avenue of particular interest is that of truly asynchronous collaborative solving sessions conducted over a long period of time, say weeks or longer. It is likely that additional patterns in solver behavior would be observed when solvers had more time to solve the problem. First because we would potentially see larger trees, but also because leaving the solving session and coming back later would require the solver having means to get

up-to-date, and to refresh her memory on previous work done. Collaboration would likely take on a different character, due to the asynchronous nature, and it could be useful to observe this and add to the list of metrics of collaboration in CoSolve. I would expect to see differences in tree shape and in collaboration styles, and the analysis tools could be refined or extended to take into account any new patterns seen.

6.4. Conclusion

Collaborative problem-solving is an area of great potential, and CoSolve is a novel point in that space. Yet along with these new possibilities there are also new problems that arise, making more difficult the already hard task of remote collaboration. Adding openness to the system is one way to address these problems, giving solvers tools for inquiry and for understanding of themselves, their team and of the tool itself.

Chapter 7: References

Ackermann, E. 2001. Piaget's constructivism, Papert's constructionism: What's the difference?
http://learning.media.mit.edu/content/publications/EA.Piage%20 %2 0Papert.pdf

Ainsworth, S., Fleming, P. 2006. Evaluating authoring tools for teachers as instructional designers. Computers in Human Behavior, 22, 131-148.

Aleven, V., McLaren, B., Roll, I., and Koedinger, K. 2004. Toward Tutoring Help Seeking: Applying Cognitive Modeling to Meta-cognitive Skills. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 227-239.

Andersen, E., Liu, Y., Snider, R., Szeto, R., Cooper, S. and Popovic, Z. 2011. On the Harmfulness of Secondary Game Objectives. Proc. Foundations of Digital Games.

Anderson, J.R., Boyle, C.F., Corbett, A.T., and Lewis, M.W. 1990. Cognitive Modeling and Intelligent Tutoring. Artificial Intelligence, 42, 7-49.

Anderson, J.R., and Pelletier, R. 1992. A development system for model tracing tutors. Journal of Artificial Intelligence in Education, 4(4), 397-413.

Arroyo, I., and Woolf, B.P. 2005. Inferring learning and attitudes from a Bayesian Network of log file data. Artificial Intelligence in Education, IOS Press, 2005.

Artino, A. (2008). Cognitive Load Theory and the Role of Learner Experience: An Abbreviated Review for Educational Practitioners. AACE Journal, 16(4). 425-439

Balakrishnan, A., Fussell, S. and Kiesler, S. 2008. Do Visualizations Improve Synchronous Remote Collaboration? Proc. CHI 2008. NY: ACM Press.

Barron, B. 2003. When Smart Groups Fail. The Journal of the Learning Sciences, 12(3), 307-359. Lawrence Erlbaum Associates, Inc.

Benson,N., 2004. The INFACT Gnome Environment: A Platform for Autonomous Agents Generating Automatic Student Assessments and Feedback. http://www.cs.washington.edu/ole/benson-thesis.pdf

Biswas, G., Leelawong, K., Belynne, K., Viswanath, K., Schwartz, D., and Davis, J. 2004. Developing Learning by Teaching Environments that support Self-Regulated Learning.
macs1.vuse.vanderbilt.edu/./papers/BettyITS2004.pdf

Brabham, D. 2008. Crowdsourcing as a Model for Problem Solving. Convergence: The International Journal of Research into New Media Technologies. Sage Publications.

Bratko, I. 2000. Modelling Operator's Skill by Machine Learning. Information Technology Interfaces, 23-30.

Bull, S. 2004. Supporting Learning with Open learner Models. Proc of 4th Hellenic Conference in Information and Communication Technologies in Education, Athens, 47-61.

Bull, S., Brna, P., and Pain, H. 1995. Extending the Scope of the Student Model. User Modeling and User Adapted Interaction. 5(1), 45-65.

Bull, S., and Kay, J. 2005. A Framework for Designing and Analysing Open Learner Modelling. Proc of Workshop on Learner Modelling for Reflection, Artificial Intelligence in Education, Amsterdam, 81-90.

Bull, S., and Mabbott, A. 2006. 20000 Inspections of a Domain-Independent Open Learner Model with Individual and Comparison Views. Intelligent Tutoring Systems, Springer-Verlag Berlin Heidelberg, 422-432.

Bull, S., Mangat, M., Mabbott, A., Abu Issa, A.S., and Marsh, J. 2005. Reactions to Inspectable Learner Models: Seven Year Olds to University Students. Proc of Workshop on Learner Modelling for Reflection, Artificial Intelligence in Education. Amsterdam, 1-10.

Bull, S., and Nghiem, T. 2002. Helping Learners to Understand Themselves with a Learner Model Open to Students, Peers and Instructors. Proc of Workshop on Individual and Group Modelling Methods that Help Learners Understand Themselves, Intelligent Tutoring Systems. Springer-Verlag, Berlin Heidelberg, 5-13.

Bull, S., and Pain, H. 1995. Did I say what you think I said, and do you agree with me?. Artificial Intelligence in Education, AACE, Charlottesville, VA., 501-508.

Collazos, C., Guerrero, L., Pino, J., Renzi, S., Klobas, J., Ortega, M., Redondo, M., and Bravo, C. 2007. Evaluating Collaborative Learning Processes using System-based Measurement. Educational Technology & Society, 10 (3), 257-274.

Conati, C. and Maclare, H. 2004. Evaluating a Probabilistic Model of Student Affect. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 55-66.

Cook, R., and Kay, J. 1994. The justified user model: a viewable, explained user model. User Modeling, Hyannis, MA, 145-150.

Cox, M. 2005. Metacognition in computation: A selected research review. Artificial Intelligence 169, pg 104-141, Elsevier.

Desmarais, M., Fu, S., and Pu, X. 2005. Tradeoff analysis between knowledge assessment approaches. Artificial Intelligence in Education, IOS Press, 125, 209-216.

Dimitrova, V. 2002. Interactive Cognitive Modelling Agents – Potential and Challenges. Workshop on Individual and Group Modellling Methods that help Learners Understand Themselves, Intelligent Tutoring Systems.

Dimitrova, V. 2003. STyLE-OLM: Interactive Open Learner Modelling. International Journal of Artificial Intelligence in Education. IOS Press, 2003, 13(1), 35-78.

Dimitrova, V. 2003. Using Dialogueue Games to Maintain Diagnostic Interactions. Lecture Notes in Computer Science. Springer, Berlin/Heidelberg. 117-121.

Engelmann, T. and Hesse, F. 2010. How digital concept maps about collaborators' knowledge and information influence computer-supported collaborative problem solving. Computer Supported Collaborative Learning. 5:299-319.

Fan, S., Robison, T., and Tanimoto, S. 2012. CoSolve: A System for Engaging Users in Computer-Supported Collaborative Problem Solving. Proc. Visual Languages and Human Centric Computing. IEEE, 1-3.

Fan, S., Johnson, B., Liu, Y., Robison, T., Schmidt, R., and Tanimoto, S.. 2010. Analyzing a Process of Collaborative Game Design Involving Online Tools. Proc. Visual Languages and Human Centric Computing. IEEE, 75-78.

Flavell, J.H. 1979. Metacognition and Cognitive Monitoring. American Psychologist. Vol. 34, No. 10, 906-911.

Falmagne, J.C., Cosyn, E., Doignon, J.P., and Thiery, N. 2003. The Assessment of Knowledge in Theory and in Practice.
http://repositories.cdlib.org/imbs/26

Fischer, G. 1990. Communication Requirements for Cooperative Problem Solving Systems. Information Systems, Vol. 15, No. 1, 21-36.

Gama, C. 2004. Metacognition in Interactive Learning Environments: The Reflection Assistant Model. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 668-677.

Greer, J., and Zapata-Rivera, J. 1999. Visualization of Bayesian Learner Models. Proc of the workshop on Open, Interactive, and other Overt Approaches to Learner Modelling, Artificial Intelligence in Education.

Green, T., and Blackwell, A. 1998. Cognitive Dimensions of Information Artefacts: a tutorial.
http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf

Heer, J., Card, S.K., Landay, J.A. 2005. prefuse: A Toolkit for Interactive Information Visualization.

Henze, N., and Nejdl, W. 1999. Student Modeling for KBS Hyperbook System using Bayesian Networks.
http://www.kbs.uni-hannover.de/paper/99/adaptivity/adaptivity.html

Herman, I., Melancon, G., Scott Marshall, M. 2000. Graph Visualization and Navigation in Information Visualization: A Survey. IEEE Transactions on Visualization and Computer Graphics, Vol. 6, No. 1, 24-43.

Janssen, J., Erkens, G., Kanselaar, G., and Jaspers, J. 2006. Visualization of participation: Does it contribute to successful computer-supported collaborative learning? Computers & Education. Elsevier Ltd.

Janssen, J., Erkens, G., and Kanselaar, G. 2006. Visualization of agreement and discussion processes during computer-supported collaborative learning. Computers in Human Behavior. Elsevier.

Jermann, P. and Dillenbourg, P. 2007. Group mirrors to support interaction regulation in collaborative problem solving. Computers & Education. Elsevier.

Kalyuga, S. (2007). Enhancing Instructional Efficiency of Interactive E-learning Environments: A Cognitive Load Perspective. Educational Psychology Review, 19. 387-399.

Kalyuga, S., Ayres, P., Chandler, P., & Sweller, J. (2003). The Expertise Reversal Effect. Educational Psychologist, 38(1). 23-31.

Kay, J. 1995. The um uoolkit for cooperative user modelling. User Modeling and User Adapted Interaction, 4, 149-196.

Kay, J. 1997. Learner Know Thyself: Student Models to Give Learner Control and Responsibility. Computers in Education, AACE, 17-24.

Kay, J., and McCalla, G. 2003. The Careful Double Vision of Self. International Journal of Artificial Intelligence and Education, IOS Press, 11-18.

Ko, A. 2006. Debugging by Asking Questions bout Program Output. ICSE. ACM.

Ko, A. and Myers, B. 2004. Designing the Whyline: A Debugging Interface for Asking Questions about Program Behavior. CHI.

Koedinger, K.R., Aleven, V., Heffernan, N., McLaren, B., and Hockenberry, M. 2004. Opening the door to non-programmers: authoring intelligent tutor behavior by demonstration. Intelligent Tutoring Systems. Springer-Verlag, Berlin 162-174.

Maass, S. 1983. Why Systems Transparency? Psychology of Computer Use, Green, T. R.

Mabbott, A., and Bull, S. 2004. Alternative Views on Knowledge: Presentation of Open Learner Models. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 689-698.

Mabbott, A., and Bull, S. 2006. Student Preferences for Editing, Persuading, and Negotiating the Open Learner Model. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 481-490.

Mason, B.J., and Bruning, R. Providing Feedback in Computer-based Instruction: What the Research Tells Us.
http://dwb.unl.edu/Edit/MB/MasonBruning.html

Mayer, R., & Moreno, R. (2003). Nine Ways to Reduce Cognitive Load in Multimedia Learning. Educational Psychologist, 38(1), 43-52.

McGilly, K. (ed.) 1994. Classroom lessons - Integrating cognitive theory and classroom practice, 51-74. Cambridge, MA: MIT Press, 1994.

Mitrovic, A., and Martin, B. 2002. Evaluating the Effects of Open Student Models on Learning. Adaptive Hypermedia and Adaptive Web-Based Systems, Springer-Verlag, Berlin Heidelberg, 296-305.

Morales, R., Pain, H., and Conlon, T. 1999. From Behaviour to Understandable Presentation of Learner Models: A Case Study. Proc of the workshop on Open, Interactive, and other Overt Approaches to Learner Modelling, Artficial Intelligence in Education. Le Mans, France.

Morales, R., Pain, H., and Conlon T. 2001. Effects of Inspecting Learner Models on Learners Abilities. Artificial Intelligence in Education. IOS Press, 434-444.

Morales, R., Pain, H., Conlon, T. 2000. Understandable Learner Models for a Sensorimotor Control Task. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 222-231.

Morales, R. 2000. Exploring Participative Learner Modelling and Its Effects on Learner Behaviour. Ph.D. thesis, University of Edinburgh, Department of Artificial Intelligence, 2000.

Moreno, R. (2004). Decreasing Cognitive Load for Novice Students: Effects of Explanatory versus Corrective Feedback in Discovery-Based Multimedia. Instructional Science, 32. 99–113.

Myers, B., Weitzman, Ko, A., Chau, D.H. 2006. Answering Why and Why Not Questions in User Interfaces. CHI. ACM.

Penichet, V.M.R., Marin, I., Gallud, J.A., Lozano, M.D., Tesoriero, R. 2007. A Classification Method for CSCW Systems. Electronic Notes in Theoretical Computer Science, 168, 237-247.

Reye, J. 2004. Student Modelling based on Belief Networks. International Journal of Artificial Intelligence in Education. IOS Press, 2004, 14, 1-33.

Russell, S., and Norvig, P. 1995. Artificial Intelligence: A Modern Approach. Prentice-Hall.

Robison, T. Transparent Array Devices for Visual Programming. Doctoral Consortium for Visual Languages and Human Centric Computing, 2005.

Robison, T. Composition of Communication Services in a Visual Programming Language. Doctoral Consortium for Visual Languages and Human Centric Computing, 2006.

Robison, T., and Tanimoto, S. Controlling Transparency in an Online Learning Environment. Proc. of Visual Languages and Human Centric Computing, 2007.

Robison, T., and Tanimoto, S. Towards a More Transparent Tutor: Opening up Assessment and Control Processes to Learners. In the Workshop for Metacognition and Self-Regulated Learning in Educational Technologies at Intelligent Tutoring Systems, 2008.

Self, J. 1990. Bypassing the Intractable Problem of Student Modelling. Intelligent Tutoring Systems: at the Crossroads of Artificial Intelligence and Education. Norwood, N.J., 107-123.

Simon, H.A. 1996. *The Sciences of the Artificial*, 3rd edition, Cambridge, MA. MIT Press.

Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. Cognitive Science, 12. 257-285.

Tanimoto, S. 2005. Dimensions of Transparency in Open Learner Models. Proc Int. Workshop on Learner Modelling for Reflection, to Supporter Learner Control, Metacognition and Improved Communication between Teachers and Learners, Artificial Intelligence in Education. Amsterdam, 2005.

Tanimoto, S. 2005. Dimensions of Transparency in Open Learner Models. Proc. workshop on Learner Modelling for Reflection, to Support Learner Control, Metacognition and Improved Communication between Teachers and Learners, AIED.

Tanimoto, S. 2006. Transparent software methodologies in image processing. Proc. Scandinavian Conference on Image Analysis. Umea, Sweden.

Tanimoto, S. 2007. Improving the prospects for educational data mining. Proc workshop on Data Mining for User Modelling, held in conjunction with User Modelling 2007.

Tanimoto, S. 2008. Enhancing state-space tree diagrams for collaborative problem solving. Proc DIAGRAMS 2008, Herrsching, Germany.

Tanimoto, S., Robison, T., and Fan, S. 2009. A Game-Building Environment for Research in Collaborative Design . Proc at the IEEE Symposium on Computational Intelligence in Games. pp.96-103.

Tanimoto, S., Carlson, A., Husted, J., Hunt, E., Larsson, J., Madigan, D. and Minstrell, J. 2002. Text forum features for small group discussions with facet-based pedagogy. Proc. CSCL, International Society of the Learning Sciences.

Tanimoto, S. 2003. Programming in a data factory. Proc. Human Centric Computing Languages and Environments, IEEE.

Wenger, E. 1987. Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge. Lost Altos, CA: Morgan Kaufmann Publishers, Inc.

Wilson, J. 1999. Defining Metacognition: A step towards recognizing metacognition as a worthwhile part of the curriculum.

Zapata-Rivera, J., and Greer, J. 2001. Externalizing Learner Modelling Representations. Workshop on External Representations in AIED: Multiple Forms and Multiple Roles, Artificial Intelligence in Education. San Antonio, Texas, 2001.

Zapata-Rivera, J., and Greer, J. 2004. Interacting with Inspectable Bayesian Student Models. International Journal of Artificial Intelligence in Education, IOS Press, 14, 1-37.

Zapata-Rivera, J., and Greer, J. 2000. Inspecting and Visualizing Distributed Bayesian Student Models. Intelligent Tutoring Systems, Springer-Verlag, Berlin Heidelberg, 544-553.

Zapata-Rivera, J., Neufeld, E., and Greer, J. 1999. Visualization of Bayesian Belief Networks. IEEE Visualization 1999 Late Breaking Hot Topics Proceedings. 85-88.

Appendix 1: CitySim Rules Shown to Solvers on Demand

Overview

CitySim is a city-building game in which you start with an empty city and place buildings each turn. Your objective is to earn as much money as you can in 30 turns. There are six different types of buildings you can place, each with its own cost and varying effects; it is up to you to decide what to place where. An overview of the buildings is given below, but the overall idea is that increasing *population* and the presence of *jobs* will increase increase *employment*, which will give you *income* from taxes. Of course, you have to keep people sufficiently happy as well.

Building Types:

- (R)esidential:
    - Buildings in which people can live
    - Increases *population* by 1 in the area around the building
    - Residential buildings cost $2 to build
- (I)ndustrial:
    - Factories that produce lots of jobs, but also pollute the nearby area
    - Increases *jobs* by 2 but decreases *happiness* by 1 in the area around the building
    - Industrial buildings cost $1 to build
- (T)heater:
    - People can go and enjoy shows here
    - Increases *happiness* by 2 in the area around the building
    - Theater buildings cost $5 to build
- (C)ommercial:
    - Provide some jobs, and people love to shop
    - Increases *jobs* by 1 and *happiness* by 1 in the area around the building
    - Commercial buildings cost $10 to build
- Ta(X) Bureau:

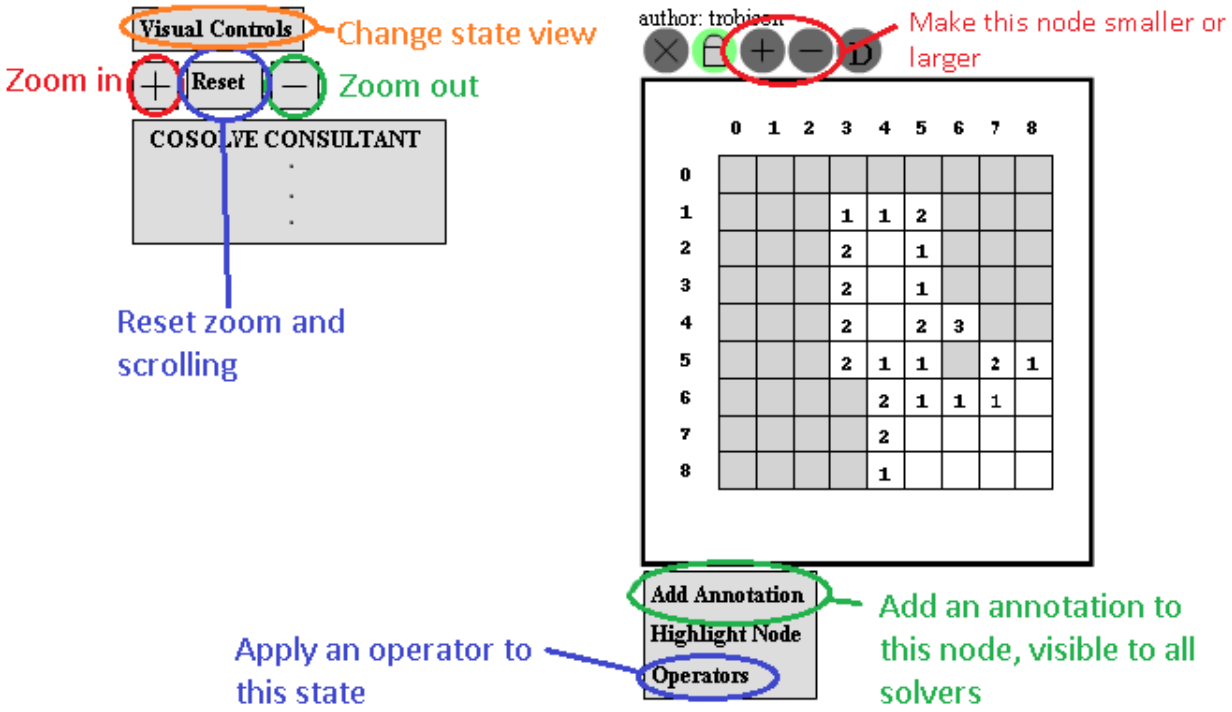
    - Lets you collect more taxes
    - Increases tax-revenue but decreases *happiness* for the entire board
    - Tax Bureaus cost $1 to build
- (D)epartment of Roads:
    - Improves all the roads, letting people travel further for work and play
    - Increases the area of positive effects of all other buildings, but not negative effects, such as the reduced *happiness* of the industrial building
    - These buildings can greatly increase the money-making power of your city, and are easily the most expensive
    - The Department of Roads is unique in its cost. It's initial cost is $100, and building a successive one costs 10 times the amount of the previous one

Rules

- Your *score* for each node is the amount of *money* in that node
- Your *money* starts at $20 and increases by your *income* each turn
- The *employment* of a cell is the minimum of the *population* and *jobs* of the cell
- *Employment* is 0 for any cell with *happiness* of -2 or lower, despite the *population* and *job* values of the cell
- Your *income* each turn is the sum of the *employment* of all grid cells multiplied by the *tax rate*
- The *tax rate* starts at 2, and can be increased by adding Tax Bureaus
- Building area of effect
    - Tax Bureaus and Departments of Roads effect the entire grid
    - All other buildings initially have effects in a 3x3 area centered on the building
    - For each Department of Roads built, the area of effect increases as follows:
        - 0 Departments of Roads: 3x3 area
        - 1 Department of Roads: 5x5 area
        - 2 Departments of Roads: 7x7 area
        - 3 Departments of Roads: 9x9 area

        - 4 Departments of Roads: 11x11 area
    - Note that the area of negative effects is always 3x3; the only negative effect in the game is the industrial building's -1 *happiness* effect
- Each cell of the 10x10 grid has a value for *happiness*, *jobs*, *population* and *employment*, each of which is based on nearby buildings
- You can view these values in the CoSolve Flash Client by clicking on the 'Visual Controls' menu, and then 'State Views'

Appendix 2: CoSolve Help Shown to Solvers on Demand



Appendix 3: Annotation Tagging Schemes

Note: None of the annotation tagging categories are mutually exclusive.

Strategy:STRAT:

Any statement/proposal concerning strategy of problem-solving. This can a statement of future work, a suggestion/command, statement of previous work or question on strategy.

Notes:

- This category does not include 'pure' evaluations that do not discuss strategy, such as "This is a good node"
- This category can include collaborative strategies ("Let's work together")
- Should not include mere statements ("We need $100 more to buy this building")

Positive Examples:

- "Let's build roads"
- "Use RIR tiling"
- "My highest score was 14759, trying a new path to top that now" [the initial part is not strategy, but the second part states plans for the future]
- "Let's work over here"

Reply: REPLY

This annotation is a reply to another annotation; either it references another, or it assumes the context of previous conversation. It's difficult to give positive/negative examples, as this tag is all about context in the discussion. Use the notes to figure it out.

Notes:

- needs to suggest that author has been reading annotations

    - this tag marks evidence that author has read other annotations
- any sort of 'I agree' or 'thanks' probably falls into this category
- praise can fall into this category, but likely won't if it's praise of a node
- DO NOT use if they are replying to their own annotation
- something general, like 'we should try RIR' shouldn't, even if prev. ann. were discussing strategy; it stands alone
- if it's not a REPLY, it's a stand-alone message by default
- general announcements do not fall into this category
- if, in a threaded forum, it would go in an existing thread, it should be tagged as REPLY

Metacognitive: META:

Mark an annotation META if it is evidence that the author was thinking about their **cognitive** progress/knowledge; **not** their progress/knowledge in the game. All instances of metacognition observed in the study were admissions of ignorance, though other types would have been counted, had they been present.

Positive Examples:

- "I don't know what the tax bureau does" [knows limitations of own knowledge]
- "I think I'm getting the hang of this game" [grasp on own knowledge]
- "p3 has a better strategy than I do" [own knowledge vs team-mate; evaluation of team-mate's knowledge, not work]
- "I'm still learning" [assessment of own knowledge]
- "I'm going to try to add building x next to building y, because I don't know what it's going to do" [somewhat unclear, but admission of ignorance of how this will work]

Appendix 4: Consultant Metrics

Below is a list of metrics visible to solvers using the full CoSolve Consultant. This items in this list were shown per team:

- Score of team's highest scoring node
- Number of nodes created
- Height of tree
- Average branching factor of tree
- Narrowness
- Top-heaviness
- Turn-taking on solution path
- Authorship entropy

The following metrics were shown per solver

- Nodes explored
- Deepest depth
- User operators used
- Exploration score for solver: an aggregate of 'nodes explored,' 'deepest depth' and 'user operators used'
- User annotations created
- User turn-taking
- User influence: a metric related to turn-taking, but instead of measuring how often this solver has built off of the work of others, it measures how often others have built off of this solver's work
- User collaboration score: an aggregate of 'user annotations created,' 'user turn-taking' and 'user influence'
- Score of solver's highest scoring node
- Number of positive, neutral and negative annotations created
- Number of positive, neutral and negative annotations placed on solver's nodes
- Time elapsed since last operator was applied

- Time elapsed since last annotation was created
- Consultant Usage: a metric showing how much the solver has been using the CoSolve Consultant. This value exists and is shown purely to incentive solvers to use the Consultant during the study
- Count of nodes available upon refresh

Highlights available to solvers using the CoSolve Consultant:

- Score
- Solution Path
- Annotation Count
- Recently Created
- Created By
- Annotation Balance

Appendix 5: Study Materials

The forms and materials used for the study can be found on the following pages of the appendices. Data can be found here:

http://socrates.cs.washington.edu/cosolve_study_11-12/

Background Questionnaire

Participant ID _____

	Please indicate how much experience you have with the following:	***Never***	***A little / Tried 1-2 times***	***Some experience***	***Fairly experienced***	***Pro / Very Experienced***
1	Using computers for common tasks, like checking your email, surfing the web, installing programs, etc.	1	2	3	4	5
2	Using design software such as Photoshop for images, DreamWeaver for web pages, Flash for animations or similar software? Include any software for creating music, editing movies, etc.; anything you consider design *In the questions below, list each software and mark your experience with it. Use the back for extra space as needed.*	1	2	3	4	5
3	Design Software:	1	2	3	4	5
4	Design Software:	1	2	3	4	5
5	Studying/using Graph theory (nodes, vertices, edges, breadth-first search, etc.)	1	2	3	4	5
6	Studying/using State-space Search (AI technique)	1	2	3	4	5
7	Playing ***single-player*** computer or video games	1	2	3	4	5
8	Playing ***multi-player*** computer or video games	1	2	3	4	5
9	Playing games in the SimCity series, or any type of city-building simulation game? *If you have experience with such games, please list which ones:*	1	2	3	4	5
10	Participating in online social networks, Twitter, Facebook, etc. *Approx hours per week spent on social networks?* ____	1	2	3	4	5
11	**Reading** online bulletin/message boards, forums, mailing lists, and other group discussion forums, such as UW Catalyst's GoPost	1	2	3	4	5
12	**Posting to** in online bulletin/message boards, forums, mailing lists, and other group discussion forums**,** such as UW Catalyst's GoPost	1	2	3	4	5
13	Using online collaboration tools (collaboration features in Google Docs, etc.), or contributing to online collaborative projects (open source software, Wikipedia, etc.) *If you have experience with them, please list which ones:*	1	2	3	4	5

Background Questionnaire

Participant ID _____

	Please indicate how strongly you agree with the following statements:	***Strongly Disagree***	***Disagree***	***Neutral***	***Agree***	***Strongly Agree***
1	I enjoy working in a group or team.	1	2	3	4	5
2	On most projects, I work better if I am in a group, rather than alone.	1	2	3	4	5
3	Given a choice, I would prefer to work on projects in a group, rather than on my own.	1	2	3	4	5
4	When working on a group project, I often feel free to share my opinions.	1	2	3	4	5
5	In group project ***discussions***, it is important to me that everyone feels they have a chance to participate in the discussion.	1	2	3	4	5
6	When trying to complete a group project, it is more important to me that everyone has a chance to share their opinion, rather than others agreeing with my opinion.	1	2	3	4	5
7	When working in a group, I often feel like I do more work than my group members.	1	2	3	4	5
8	I am often hesitant or reluctant to participate or share my thoughts in ***face-to-face*** group discussions.	1	2	3	4	5
9	I am often hesitant or reluctant to participate or share my thoughts in ***online*** group discussions. (check if you have never participated in an online group discussion, i.e. “n/a” [_])	1	2	3	4	5
10	In ***face-to-face*** group discussions, I express my opinion equally as often as other group members.	1	2	3	4	5
11	In ***online*** group discussions, I express my opinion equally as often as other group members. (Check if n/a [_])	1	2	3	4	5
12	In ***face-to-face*** group discussions, I often wish I could participate more.	1	2	3	4	5
13	In ***online*** group discussions, I often wish I could participate more. (Check if n/a [_])	1	2	3	4	5
14	When discussing a project ***face-to-face***, I am sometimes uncertain how to best participate.	1	2	3	4	5
15	When discussing a project **online**, I am sometimes uncertain how to best participate. (check if n/a [_])	1	2	3	4	5
16	I believe working in a group results in a better solution than working alone.	1	2	3	4	5

Do you consider yourself: [_] Female [_] Male Your age in years: _____

If you are a student…

…what year in school are you (Autumn 2012)? Grad / Undergrad, Year: ____

…what are you studying? _____, OR undecided

If you are not a student but are/were employed in some way, what is your current (or last) job/career position?

_____

CoSolve Test

Participant ID_____

Is this the pretest or the posttest? _____

Instructions

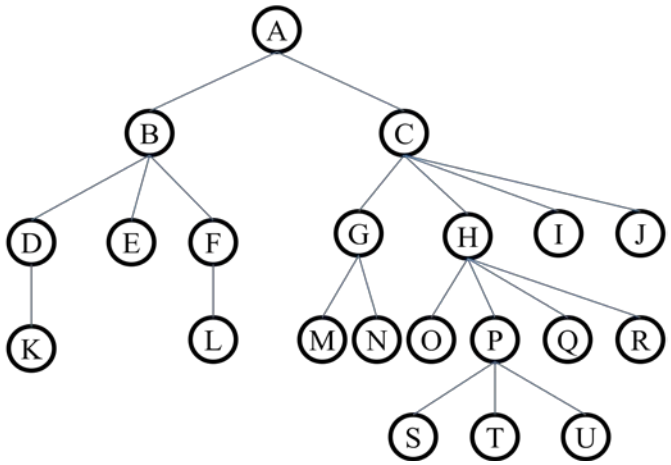
Match each item in the left column below with the best description of it in the context of collaborative problem-solving using a tool like CoSolve. The descriptions are found in the right column; write the description's letter (a, b, c, etc.) in the blank in the left column. If you don't know, it's fine to leave an item blank. Not all of the descriptions will be used.

Term	Description
1. ____ State	a. The node containing the initial state of the problem.
2. ____ Operator	b. A message sent to a user from a CoSolve administrator.
3. ____ Parent	c. Junction in a solving tree representing a moment in the problem-solving process–whether unsolved, partially solved or fully solved.
4. ____ Root	d. A message created by a user and placed on a CoSolve node.
5. ____ Annotation	e. Can create a new node from a previous node.
6. ____ Level	f. All nodes at the same depth.
	g. The result of a user error.
	h. A node to which an operator has been applied.
	i. A unique solution to a problem.
	j. The deletion of a node.

CoSolve Test

Participant ID_____

The next few questions refer to the following graph. Refer to the node by the letters printed inside them.



Question 8. What is the root of this tree?

Question 9. How many children does 'H' have?

Question 10. If 'S' is the highest scoring node, what is the solution path for this tree?

Question 11. What is 'F's parent?

Question 12. List 'G's children.

Question 13. What is the depth of node 'R', assuming the root's depth is 0?

Question 14. Do you have any other comments regarding your answers?

Interview Questions [Roles Version]

1. [get them to guide you through their tree] Walk through your tree and describe the design process to us. Describe what the 'story' is behind each part of the tree; what was your thought process when building this branch?
   **For 4 sections (branches or general regions) of the tree, try to get the following answers. Do for 2 sections created by this user, and 2 by others
      **What the thinking/story behind this part of the tree was
      **Whether it worked out or not; why or why not
      **What was learned

   {OLD QUESTIONS: OPTIONAL….ask as they seem appropriate}
2. If you were to start a similar exercise from scratch – a similar game playing exercise – how would you go about it this time? Assume you were working with the same team members as before.
3. Were you happy with the solution you and your team came up with? Is there more you would have liked to have done?
4. What criteria did you use to evaluate your team's progress? Specifically, 1) what your team had done and 2) what needed to be done?
5. Describe your various reasons for creating new nodes; for instance: to get a higher score, or to experiment with a new technique.
6. Describe your various reasons for creating new annotations; for instance, as a reminder to yourself, or as a message for a team-mate.
7. Describe how your group members communicated with each other.

   {ROLES QUESTIONS}
8. Were you usually aware of what role you were in?
9. How about what role your teammates were in?
10. Did your roles affect your behavior? How so? For example, did you spend most of your time thinking about / trying to complete your role, or did you mostly ignore it? Did you try to complete it quickly, or did you spread it out over the role phase?
11. What was your strategy, thoughts, or feelings when you were in each of the following roles:
    a. Brainstormer
    b. Critic
    c. Supporter
12. What about when your teammates were in each of those roles? How do you think it affected their behavior, and your attitude towards their behavior?
13. How would you have behaved differently without a roles system?
14. Thoughts on the Current Role tab? How often did you use it?
15. Thoughts on the Overall Role tab? How often did you use it?
16. What parts of the flash program were helpful in solving the problem? List the 3 most helpful aspects and why?
17. What parts of the flash program were harmful or distracting in solving the problem? List the 3 most harmful or distracting aspects and why?
18. What additional features can you think of that would make <u>problem-solving</u> in CoSolve easier/better?
19. What additional features can you think of that would make <u>collaboration</u> in CoSolve easier/better?

Participant ID: _____

Wrap-up Questionnaire

Part A. For each of the statements in the boxes below, select on a scale of 1-5 how much you agree or disagree with that statement. Here are some descriptions of terms for clarification:

- *The Citysim Game* refers to just the city building puzzle game itself, independent of CoSolve.
- *The CoSolve Consultant* refers to the toolbox shown below the magnification buttons on the left of your CoSolve solving session; it exists to provide information to problem solvers, and includes node highlighting, but does not include the Annotation List.
- *The CoSolve Interface* refers to the interactions for solving a problem in CoSolve, including panning and zooming controls and interactions for applying operators and annotations.
- *The tree visualization of the problem-solving space* refers to how each state in the process is shown as a node, connected to a parent and possibly children. This does ***not*** include the CoSolve Consultant and could pertain to games or problems ***other*** than the CitySim Game.

	Statement	***Strongly Disagree***	***Disagree***	***Neutral***	***Agree***	***Strongly Agree***
1	*The tree visualization of the problem-solving space* was confusing	1	2	3	4	5
2	*The tree visualization of the problem-solving space* was helpful	1	2	3	4	5
3	*The tree visualization of the problem-solving space* was difficult to navigate	1	2	3	4	5
4	*The tree visualization of the problem-solving space* helped encourage collaboration	1	2	3	4	5
5	Regarding navigation of the tree, I was often lost or disoriented	1	2	3	4	5
6	I fully understand the rules of *the CitySim Game*	1	2	3	4	5
7	I often felt overwhelmed with the number of choices at any given point.	1	2	3	4	5
8	It was difficult to do well in *the CitySim Game*	1	2	3	4	5
9	I believe that my team-mates understood *the CitySim Game* better than I did	1	2	3	4	5
10	*The CitySim Game* was fun	1	2	3	4	5

Participant ID: _____

11	*The CoSolve Interface* was difficult to use	1	2	3	4	5
12	*The CoSolve Interface* was helpful in playing the game	1	2	3	4	5
13	*The CoSolve Interface* was easy to learn	1	2	3	4	5
14	*The CoSolve interface* made collaboration more difficult	1	2	3	4	5
15	*The CoSolve interface* was helpful in understanding my team's problem-solving process	1	2	3	4	5
16	My teammates and I each did an equal portion of the work	1	2	3	4	5
17	I felt like I was competing with members of my team	1	2	3	4	5
18	I was always well aware of what my teammates had done	1	2	3	4	5
19	It was easy to collaborate with my teammates.	1	2	3	4	5
20	I was aware of what my team-mates were doing most of the time.	1	2	3	4	5
21	I had a difficult time keeping up with my team-mates.	1	2	3	4	5
22	I wish my team-mates would have collaborated more.	1	2	3	4	5
23	*The CoSolve Consultant* was difficult to use	1	2	3	4	5
24	*The CoSolve Consultant* was helpful in playing the game	1	2	3	4	5
25	*The CoSolve Consultant* was difficult to learn	1	2	3	4	5
26	*The CoSolve Consultant* was helpful in understanding my team's problem-solving process	1	2	3	4	5
27	*The CoSolve Consultant* helped my team collaborate	1	2	3	4	5
28	The highlighting features of *the CoSolve Consultant* were not useful	1	2	3	4	5
29	I would have liked more time in which to complete the CitySim activity	1	2	3	4	5
30	I needed less time than I was given to complete the CitySim activity	1	2	3	4	5
31	My team-members and I communicated effectively regarding our design process	1	2	3	4	5
32	I have a better understanding of my own problem-solving process now than before playing the game	1	2	3	4	5

33	I enjoy working in a group or team.	1	2	3	4	5
34	On most projects, I work better if I am in a group, rather than alone.	1	2	3	4	5
35	Given a choice, I would prefer to work on projects in a group, rather than on my own.	1	2	3	4	5
36	When working on a group project, I often feel free to share my opinions.	1	2	3	4	5
37	In group project discussions, it is important to me that everyone feels they have a chance to participate in the discussion.	1	2	3	4	5
38	When trying to complete a group project, it is more important to me that everyone has a chance to share their opinion, rather than others agreeing with my opinion.	1	2	3	4	5
39	When working in a group, I often feel like I do more work than my group members.	1	2	3	4	5
40	I am often hesitant or reluctant to participate or share my thoughts in online group discussions.	1	2	3	4	5
41	In online group discussions, I express my opinion equally as often as other group members.	1	2	3	4	5
42	In online group discussions, I often wish I could participate more.	1	2	3	4	5
43	When discussing a project online, I am sometimes uncertain how to best participate.	1	2	3	4	5
44	I believe working in a group results in a better solution than working alone.	1	2	3	4	5
45	My attitude towards working in a group has changed during the course of this activity.	1	2	3	4	5

Part B.
For the following items, please write a sentence or two giving your answer and an explanation.

1. For this study you created a solution for a SimCity like game. Describe the game: what the player does, what the goal is, etc.

2. If you were to start a similar exercise from scratch, would you prefer to work alone or with a team (say, of randomly chosen participants)? Explain.

Participant ID: _____

3. Are you satisfied with the quality of the interaction between yourself and your team-mates? Why or why not? What would you change?

4. Describe your team's collaborative process.

5. Roughly what percentage of the nodes created were yours, when compared to those of your team-mates? Explain.

6. Roughly what percentage of the annotations created were yours, when compared to those of your team-mates? Explain.

7. Roughly what percentage of the overall work (ideas, experimentation, time-spent, nodes/annotations created, etc.) was your contribution, compared to those of your team-mates? Explain.

8. If you had performed this activity by yourself, without any team-mates, do you think your score would have been higher, lower or the same? If higher or lower, how much? Explain.

9. If your team could have communicated verbally, do you think your score would have been higher, lower or the same? If higher or lower, how much? Explain.

10. Did this process help you learn about your problem-solving style? If so, what/how?

Participant ID: _____

11. Did this process help you learn about your collaboration style? If so, what/how?