“Do as I do, not as I say”: A Transparent Constraint Acquisition and Satisfaction Approach

Mave Houston¹
N. Hari Narayanan
Richard Chapman

Technical Report CSSE03-08

July 11, 2003

¹ Author to whom correspondence should be addressed.
“Do as I do, not as I say”: A Transparent Constraint Acquisition and Satisfaction Approach

Mave Houston, N. Hari Narayanan and Richard Chapman
Intelligent & Interactive Systems Laboratory
Department of Computer Science and Software Engineering
Auburn University, USA
Email: {maveh, narayan, chapman}@eng.auburn.edu

Abstract

Interactive Constraint Satisfaction (ICS) is an approach to solving Constraint Satisfaction Problems (CSP) interactively with the user [3]. This class of problems includes subjective, everyday problems such as finding a house, as well as complex scientific problems. A difficult issue in ICS is how to acquire constraints from the user, when the user may not be able to clearly and accurately articulate them. We present a Transparent Interactive Constraint Acquisition and Satisfaction approach to address this issue. It employs a combination of clustering and inductive generalization to unobtrusively acquire multiple and/or compound constraints from the user and apply them to a CSP. We describe the application of the approach to hardware/software partitioning and future research directions.

1 Introduction

There are a number of problems characterized by a problem solving process in which users must process large amounts of data in order to arrive at a solution. In addition, the solution must also satisfy a number of constraints. These include problems such as real estate or vehicle searches, where user preferences are subjective, and scientific problems such as predictive toxicology or hardware/software partitioning. In these cases, the size of the solution search space is sufficiently large to warrant automated support, and constraints acquired from the user are necessary to solve the problem [4]. This class of problems benefits from ICS because user interaction can facilitate the attainment of a more optimal solution. ICS research is concerned with the interaction model between the user and the system, which employs the constraint satisfaction method.

A difficult issue in ICS is how to acquire constraints from the user, when the user may not be able to clearly and accurately articulate them. We present a Transparent Interactive Constraint Acquisition and Satisfaction approach to address this issue. This approach employs a combination of clustering and inductive generalization to unobtrusively acquire multiple and/or compound constraints from the user and apply them to a CSP.

Recently, researchers have begun to focus on constraint acquisition, an issue that arises during the problem-solving phase of complex CSPs, where the satisfaction of numerous constraints must be met. Realizing that users may be unable to specify constraints during the problem solving process, researchers have devised several methods to elicit constraints from the user. These methods can be loosely classified into two groups: Interactive Constraint Acquisition (ICA), and Interactive Constraint Acquisition & Satisfaction (ICAS). Interactive Constraint Acquisition acquires constraints explicitly from the user and operates under the assumption that while the user may be unable to articulate constraints, the user can identify a constraint as positive or negative
if they are presented with examples of them. To that end, O'Connell, O'Sullivan, and Freuder use a 'learning by asking questions' approach to acquire constraints from the user [7,8]. The authors contend that users have difficulty specifying arbitrary constraints. They assume that if presented with an example of the constraint, the user can then determine if it as a positive or negative example of the constraint they would like to specify. They also assume that the hypothesis space is known in advance, even though in general this is untrue. They claim that there is significant anecdotal evidence in many real-world applications to support this assumption. The interaction begins with a query. The user answers the query, and may supply an example for the system to use. The generalization is obtained by asking questions to further refine the version space for the constraint in question. This generalization can be rejected by the user, further refining the version space of the constraint the user wishes to articulate.

The aforementioned approach does not address the possibility that users may be unable to discern between positive and negative constraints. While likely true for subjective problems, the assumption that the user would not make mistakes when discerning between positive and negative constraints may not be a valid one for very complex scientific problems. For example, in the domain of hardware/software partitioning, where cost information may not be fully available or known to the user at the time of partitioning, it is possible that the user may incorrectly identify constraints as positive or negative due to incorrectly estimated cost information. This is true because these cost information estimates rely on the particular partitioning estimation model that the partitioning tool employs. In order to disentangle the interface of the tool from the underlying system, the partitioning estimation model employed is not provided to the user [13]. Another limitation is that this explicit constraint acquisition approach does not appear to support the timely acquisition of multiple or compound constraints.

Alternatively, Interactive Constraint Acquisition & Satisfaction implicitly derives constraints from the user in an intermediate step toward solving the constraint satisfaction problem [4]. The assumption here is that users can articulate a constraint. Constraint acquisition may be implicit or explicit, but the main focus is on acquiring the partial model of the CSP and then solving the resultant CSP. Freuder and Wallace present a suggestion strategy for constraint-based matchmaker agents that concentrates both on the interaction between the user and the system, and on the development of a partial model of the CSP defined by that interaction [4]. The authors make the point that perhaps the two goals of efficiently solving the customer's problem and making that problem explicit are potentially divergent. Their research explores the tradeoffs that could be made to partially ameliorate this problem. Essentially, the authors are highlighting the differences between ICA and ICAS. Though this method avoids long dialogues between the user and the system by not asking the user to explicitly specify constraints, it assumes that the user can articulate further constraints when presented with a configuration that does not meet their requirements. This assumption may not be valid in domains such as hardware/software partitioning.

To address these limitations, we present an alternate method that focuses on a “do as I do, not as I say” Transparent ICAS approach. This alternate approach employs a combination of clustering and concept acquisition or inductive generalization to acquire multiple and/or compound constraints from the user. These constraints are used to develop a partial model of the resultant CSP, which is then resolved. The approach is transparent because users are intuitively guided to initiate the problem-solving process. Constraints are implicitly obtained without asking the user to specify...
them directly. This approach was originally developed for hardware/software partitioning, and relies on clustering to acquire constraints and to define a partial CSP.

The rest of this paper is organized as follows: Section 2 describes the rationale for the approach and a detailed description. Section 3 provides a detailed example of the application of the approach to hardware/software partitioning, and Section 4 contains conclusions and directions for future work.

2 Rationale and Description of Approach

The "do as I do, not as I say" transparent approach to ICAS considers interactive problem solving from a different perspective. Instead of asking the user to positively or negatively identify examples of constraints, the user is prompted to begin the problem-solving process themselves. Work that the user begins is used to ascertain constraints and solve the resultant CSP simultaneously. The user is not asked to explicitly verify positive or negative constraints in order to minimize user error. It is assumed that the user knows enough about the problem to begin the problem solving process but that user can make mistakes if asked to identify positive or negative constraints.

There are four phases to the Transparent ICAS approach, which was developed for the domain of hardware/software partitioning. Hardware/software partitioning is defined as an m-way classification problem where the solution is a division of an embedded hardware/software system specification of tasks into either hardware or software such that certain constraints are satisfied.

Humans have typically done this partitioning task by hand, using a number of effective rules of thumb, for many years. However, human designs are generally not optimal, and automated optimization algorithms can do better in some, but not all cases. The automated designs typically make no sense to a human designer, since the semantics of the system were not understandable by the algorithm. Further, the human's rules of thumb can also lead to more efficient designs in some cases, for reasons not well understood.

A widely accepted means of tackling the hardware/software partitioning problem involves clustering. Users can pre-cluster and pre-partition part of the partitioning problem space, reducing the solution search space and yielding specific examples of constraints at the same time. Essentially, the user is beginning the problem-solving process. One previous hardware/software partitioning approach involves this pre-processing step, where users can cluster part of the system specification and then the system follows behind the user and clusters any remaining nodes [5]. One limitation of this approach is that even though the user can increase his/her interaction with the system via a pre-clustering step, the system does not fully utilize the user's knowledge. Instead, the system performs its own mutually exclusive clustering routine independent of the user's interaction. Also, the system does not allow the user to preset any partitions, even though the user may have ideas about feasible or desired partitions.

In order to make the most of the user's knowledge and the optimization algorithms available, we combine these approaches by letting the algorithm provide its computational power to assist the human designer, and by employing a combination of clustering and inductive generalization techniques to simultaneously and unobtrusively acquire constraints from the user. This involves examining the user's clusters and initial partitions and learning how to mimic them, essentially acquiring multiple and compound constraints and then using that information to define a partial CSP. A first step in this direction involves studying the different means of clustering.
This produces a better and a more understandable design.

Clustering is a widely accepted approach to reducing the computational time of hardware/software partitioning [13]. There are many different approaches to clustering, but all of them can be classified under two groups: partitional clustering and hierarchical clustering. Partitional clustering divides the data into k clusters that optimize a desired cost function. Hierarchical clustering involves nesting a sequence of clusters. Agglomerative hierarchical clustering places objects in their own respective clusters. Those clusters are then merged until there is only one large cluster remaining. Divisive hierarchical clustering reverses the process, subdividing a large cluster into smaller pieces. Both of these clustering approaches use distance functions based on cost functions to determine similarity [1,6,11,14].

With respect to hardware/software partitioning, many of the above clustering approaches assume that the attributes of the data points are numeric. In instances where partitioning is fully automated, this is not a poor assumption. Consider the case where a user is attempting to cluster operations to minimize total execution time through a critical path. This cluster may include some homogenous operations, as well as others that seem unrelated. To ascertain the user's intent, categorical or numerical clustering alone would not address this instance. Moreover, the user's clustering decisions may not be based on numeric metrics or simple categorical attributes solely. Additionally, there are some approaches which consider the most simple of all categorical clusters – those based on similarity attributes, such as the type of operation to be clustered. These approaches, however, still do not utilize user knowledge, nor do they implement more complex categorical clustering methods. Since the metrics or attributes the user considers to cluster data would sometimes be categorical and not numerical, the above clustering approaches would be unable to mimic the user's clustering actions.

Thus, a better approach would combine both clustering approaches, depending on the type of initial clustering choices the user makes. If the user's clustering choices are based on numerical attributes, then clustering via cost functions would be fine. However, if the user's clustering choices are not based on numerical attributes, then categorical clustering would be necessary. Determining what type of clustering the user employed could be accomplished by an elimination process.

Additionally, the initial clusters and partitions that the user specifies can be considered specific constraints. It is for this reason that this problem can be modeled as an ICAS problem, where the local constraints (clusters and partitions) set by the user can then be examined for concepts and those concepts can then be applied to the rest of the partitioning problem space that the user did not cluster. In this case, we would want to generalize the user's clusters and partitions across all of the nodes of the specification. After these user-defined partitions and clusters are generalized, they are applied to the rest of the problem space in the form of clusters, partially defining the CSP to be solved. The resultant CSP is partial because it is unlikely that all of the clusters would be assigned to hardware or software before the constraint satisfaction process begins.

Our approach of Transparent ICAS is applied to hardware/software partitioning in four phases:

1. **User-initiated Problem Solving phase:** The user can begin to solve the partitioning problem by grouping nodes of a specification graph into clusters, or to the specific "superclusters" of hardware or software.
2. **Knowledge/Concept Acquisition phase:** During this step, clusters
from phase one are examined to separate numerical clusters from categorical clusters by the process of elimination. Once the type of cluster has been identified, the similarity metric or attribute is determined for that cluster.

3. **Inductive Generalization phase:** Through unsupervised learning, concepts gleaned from phase two are generalized to constraints in the form of clusters that are applied to the rest of the problem space. This produces a partial CSP.

4. **CSP Resolution phase:** The partial CSP is passed to a constraint satisfaction solver, which searches for a solution. If conflicts are found, users are prompted to retract certain constraints until no more conflicts arise and a satisfactory solution is found.

This approach has been developed specifically for hardware/software partitioning, but it can be applied to any other problem that reduces to the binary classification problem and requires user interaction.

### 3 Example: Hardware/Software Partitioning

An example is provided to illustrate the Transparent ICAS approach. This example is in the area of hardware/software partitioning. Suppose that a user has created the system specification for a new embedded device. The user has space for x amount of hardware area, uses a specific software processor. Typical time constraints are for total time thorough the critical path. Of course, communication overhead affects the above constraints and must be considered as well. These are resources that the user has available.

We are developing a tool called PHOENICS (Partitioning Hardware and sOftwarE via Non-intrusive Interactive Constraint Satisfaction) based on the Transparent ICAS approach to help solve such partitioning problems. The user would open the tool and then select partitioning from the options menu, or from a convenience button in the toolbar underneath the menu bar. Then the user would select the specification that describes the system they wish to partition. Then the user will begin the task of clustering the system specification (functions/lines of code) into groups and even placing some of these groups of one or more items into either hardware or software.

A constraint that the user may be attempting to specify is that all floating-point operations must be executed in hardware. For example, in the design of computer graphics processors, it is widely known that division is an operation that is executed faster if implemented in hardware. Thus, a user might cluster as many operations or functions containing division operations into hardware. This would be an example of a simple categorical cluster, since there would be no numerical metric associated with this cluster, unless one considers hardware estimation time costs over software estimation costs. Then one could consider the metric to be both categorical and numeric. Alternatively, suppose the user (whom we assume has designed or who fully understands the system specification) decides to cluster a group of functions and procedures together merely because it is simpler to deal with them if clustered together. There would be no numerical metric associated with this cluster. Therefore, the cluster would be categorical.

The system specification (written in VHDL) is presented in three forms to aid the user in partitioning. The three views are the dependence flow graph view, the GML_DFG graph representation in textual format and the VHDL source view. The first view seen by the user is a dependence flow graph, which represents the control structure and data dependencies inherent in the VHDL behavioral description and represented textually in a GML_DFG graph file format [2, #]. The user clusters the
behavioral description by clicking on operations in the dependence flow graph representation. These different views are designed to assist the user in the partitioning process. Figures 3.1 and 3.2 show examples of these.

entity adder is
  port (a_i : IN REAL;
       i_o : OUT REAL);
end adder;

architecture behavioral of adder is
begin
  process
    variable a, i : REAL;
  begin
    a := a_i;
    a := a + a_i;
    i_o <= a;
  end process;
end behavioral;

Figure 3.1 VHDL behavioral description

Figure 3.2 GML_DFG representation

The user can do as little or as much clustering as he/she wishes. PHOENICS will then perform a consistency check to determine if any of the current partitions contradict each other. If there are no conflicts, then the tool will examine the clusters to determine how the cluster was formed. The question is: are the clusters numerical or categorical? In either case, a similarity metric (or attribute, in the case of categorical clustering) must be determined so the tool can then mimic that same clustering method over the rest of the clusters that have the same similarity.

Figure 3.3 VHDL source code of a half-adder

After this process, the constraint solver can iteratively assign the remaining functions and clusters to hardware or software until a solution is found. If no solution is found, PHOENICS prompts the user to retract assertions from the largest cluster that he/she created. This process of reassigning the largest clusters will continue until a solution is found.

In short, the interaction model goes as follows:

1. The user selects clusters and partitions from a graphical representation of the system (or problem) specification. If there are any inconsistencies, the tool asks the user to retract his/her last assertion and assert new clusters/partitions. This process repeats until the user is finished asserting clusters/partitions and there are no inconsistencies.
2. The tool decides which clustering methods the user is employing via concept acquisition.
3. The tool then applies the appropriate descriptive generalization methods (includes clustering tasks together and clustering tasks into hardware or software) to the rest of the data, where applicable.
4. The tool then searches for the solution constructively.
If a conflict occurs, then the user is prompted to backtrack to the largest cluster and reassign the tasks to hardware or software. This process continues until a solution is found.

4 Conclusions and Future Work

We have presented a new, transparent approach to ICAS. This approach, however, has only been applied to constraint satisfaction problems that can be algorithmically reduced to binary classification, such as hardware/software partitioning.

There are other CSPs that could benefit from the “do as I do, not as I say” approach. For example, in the field of archaeometry, scientists work to attribute works of art to a particular artist. In the case of painting, radiological methods including X-ray, digital radiography, computer-tomography and color analyzing are utilized in art authentication [12]. This information is considered along with information about the artist’s signature style to determine with some certainty the origination of the work. What makes this problem interesting is that subtle expert knowledge is necessary in order to correctly authenticate the works of art. Even though art authentication can be expressed as a CSP, it does not algorithmically reduce to an instance of binary classification. Further research is required to determine how to extend our approach to this problem.

Though work has been presented to suggest a general framework for learning constraints using artificial intelligence and on user interaction principles for constraint satisfaction [7,8], work has not focused specifically on defining a specialized methodology for ICAS tool development or determining a classification of CSPs that benefit from one method of ICAS over another. The very nature of some complex scientific problems seems to suggest that certain methods would be more applicable than others, but further work is required to fully explore the possibilities. Our further research will extend the current Transparent ICAS approach so that it can be applied to a wider class of problems and ascertain the applicability of the approach to other problem domains.

5 References


