AI Magazine Volume 20 Number 4 (1999) (© AAAI)

The CP'98 Workshop on Constraint Problem Reformulation

Jeremy Frank

On 30 October 1998, Mihaela Sabin and I presented the Constraint Problem Reformulation Workshop in conjunction with the Fourth International Conference on the Principles and Practices of Constraint Programming held in Pisa, Italy. Roughly 30 people attended this workshop. This article summarizes the papers presented at the workshop and highlights some of the questions and issues raised during the discussion.

The task of selecting the best representation for solving a problem by means of automatically reformulating the problem is a core challenge in AI. Saul Amarel (1968) outlined the impact of different representations of the missionaries and cannibals problem on the performance of the algorithms used to solve the problem. He also proposed automating the reformulation process and, consequently, the process of selecting representations:

> The general problem of representation is concerned with the relationship between different ways of formulating a problem to a problem solving system and the efficiency with which the system can be expected to find a solution to the problem. An understanding of the relationship between problem formulation and problem solving efficiency is a prerequisite for the design of procedures that can automatically choose the most "appropriate" representation of a problem….

Recent work on reformulation has led to improved problem-solving performance, especially for planning (Bylander 1997; Kautz and Selman 1996). Reformulation has also proven useful in solving tractable problems; for example, the proof of tractability of 2-SAT involved the reformulation of 2-SAT as a graph problem (Papadimitriou 1994). However, not enough is known about how and when to apply reformulation.

The workshop organizers invited researchers to address the following questions: What is reformulation? How can it be defined formally? Is reformulation different from modeling or search? When and how does one distinguish among them, if at all? What can be gained from reformulation, for example, problem-solving speed, improved understanding of the problem domain, new algorithms, or heuristics on original representations? What are some successes and failures of reformulation? Can these successes and failures be generalized to lead to a theory of reformulation?

On 30 October 1998, Mihaela Sabin and I ran the Constraint Problem Reformulation Workshop in conjunction with the Fourth International Conference on the Principles and Practices of Constraint Programming held in Pisa, Italy. The goals of the workshop were to discuss the nature of constraint problem reformulation and the benefits and difficulties in reformulating constraint problems and to summarize and understand the recent work in this area.

Presentations

A total of 8 papers were submitted to the workshop. Although the papers covered many different topics, they shared several common themes concerning reformulation. The major themes of the papers were the definition and use of reformulations, constraint propagation, problem decomposition, multiple reformulations, algorithm paradigms, heuristics, and human interfaces.

A full list of the papers can be found at ic-www.arc.nasa.gov/ic/people-frank/workshop.html.

Defining and Using Reformulations

The papers presented many different types of reformulation. Some reformulations transform a problem into a different problem, others transform problems by adding constraints, and still others re-express the constraints in a different way. Most of the papers directly discuss performance enhancements as a result of reformulation. Reformulation was used to enable constraint propagation and heuristics, which are designed to limit search. Other reformulations enabled using different algorithm paradigms to solve problems. Additionally, reformulation using problem decomposition is useful both as a method to improve algorithm performance and as a way to reveal structure in problems, thereby improving understanding of the problem domain.

Constraint Propagation

Informally, constraint-propagation procedures add constraints to a problem by using limited, but efficient, inference procedures. These procedures either directly eliminate values, resulting in less branching during search or the discovery of inconsistencies, or add constraints that can aid later propagation. Several papers discuss reformulations that exploit differences in constraint propagation between representations. Paul Shaw, Kostas Sterigou, and Toby Walsh presented a reformulation that takes advantage of the power of generalized arc consistency on all-diff constraints over k variables as opposed to binary constraints. Simin He and Bo Zhang transformed SAT problems into polynomial equations to perform a type of resolution. Berhard Seybold, F. Metzger, and G. Ogan showed how functional constraints in constraint-satisfaction problems (CSPs) can be

propagated effectively using path consistency, leading to the elimination of redundant constraints. Henry Kautz and Bart Selman showed how certain types of constraint propagation can be done in SAT encodings of STRIPS planning problems. Berthe Choueiry and Guevara Noubir discussed how to remove constraints among given variables to partition their domains into equivalence classes, which can be used in a new adaptive strategy for hierarchical arc consistency.

Heuristics

Reformulation can lead to either heuristics or bounds that can also be used to limit search. For example, Alexander Bockmayr and Yannis Dimopolous used linear programming approximations to bound plan length in planning problems by allowing variables representing plan steps beyond a certain point to have noninteger values. The solution to the resulting mixed integer–linear programming problem provides a bound on the quality of the plan.

Algorithm Paradigms

Reformulation can take advantage of different algorithm paradigms that are more effective in some domains. Kautz and Selman reformulated planning problems to take advantage of backtracking and local search to solve propositional representations of planning problems. Bockmayr and Dimopolous also reformulated planning problems to take advantage of linear programming; in conjunction with problem decomposition, they were able to find bounds on solutions to planning problems to prune search more effectively.

Problem Decomposition

Problem decomposition involves dividing problems into subproblems that can be solved separately, leading to both increased performance as a result of problem-size reduction and implicit parallelism. Choueiry and Noubir discussed a reformulation that effectively generates a conjunctive decomposition of a CSP into two independent subproblems, thus reducing the overall computational complexity of solving the original problem yet guaranteeing, under well-specified conditions, the preservation of satisfiability. Seybold, Metzger, and Ogan showed that certain CSPs can be divided into functional and nonfunctional subproblems; the efficient handling of functional CSPs resulted in a decomposition of the problem variables that lead to better problem-solving performance for these CSPs. Sabin and Eugene Freuder showed how certain representations implicitly decompose problems into subproblems and discuss the pitfalls of these representations. These pitfalls include unnecessary constraints and unintended inconsistencies. Jean-Louis Bouquard and Christian Lenté decomposed a problem with a single resource by solving an equivalent problem on multiple resources, thereby improving problem-solving performance.

Multiple Reformulations

Several reformulations are sometimes even more effective than a single reformulation. Kautz and Selman used multiple reformulations to solve planning problems, first by fixing plan length to produce a GRAPHPLAN representation, then by translating into SAT representation. This procedure allowed constraint propagation in both domains to effectively reduce the complexity of the resulting SAT representation even more.

Human Interfaces

Some reformulations are designed to improve the ability of humans to interact with automated problem solvers. Bouquard and Lenté indicated that representing sequence-dependent setup cost in scheduling is not possible using constraints in standard constraint solvers; reformulation allows them to encode the problem easily in these constraint-solver packages. Shaw, Sterigou, and Walsh showed that representing a problem with *k*-ary constraints results in a shorter and more intuitive description than representing the same problem with binary constraints. Symmetries and interchangeability are also useful for supporting interaction with users by summarizing families of partial solutions that are qualitatively equivalent. Choueiry and Noubir discussed the approximation of the interchangeability relations among the values of a CSP variable, allowing families of solutions to be found efficiently and summarized easily.

Issues

The workshop presentations stimulated considerable discussion. This discussion revealed that there is still considerable room for improvement in understanding reformulation. The following subsections present the current state of knowledge concerning reformulation.

What Is Reformulation?

Although a formal definition of reformulation remains elusive, there are several intuitive notions of what distinguishes reformulation from other techniques. In contrast to abstraction and approximation, reformulation is intended to modify the formulation of a problem yet keep the essence of the problem unchanged. Although solutions might be lost or added in abstraction, reformulation only transforms the solution space. The reformulated problem necessarily belongs to the same complexity class as the original problem. However, the reformulation can take advantage of algorithms with improved average-case complexity.

Reformulation is similar to modeling, that is, the task of formally stating a problem in a rigorous, mathematical way. However, reformulation is also related to the problem-solving process itself. These concepts occupy a continuum, with modeling at one end, search and problem solving on the other end, and reformulation in the middle. What distinguishes reformulation from modeling and reformulation from search and problem solving?

There are ways to distinguish modeling from reformulation. One perspective is that the modeling effort ceases when the problem is represented formally. Reformulation, then, is the translation of one formal representation of a problem into a different formal representation of the same problem, subject to the constraints mentioned in the previous paragraph. Reformulation is useful because modeling is difficult, and it is often easier to manipulate a formal model than it is to redo the modeling effort. For exam-

ple, the original model might contain excessive representational power, and the reformulation might involve eliminating the excess power. Automated reformulation requires an initial representation and target representations encoded in software; thus, modeling is the step that occurs before automated reformulation is possible.

It is more difficult to distinguish reformulation from problem solving and search. One can argue that search is merely a sequence of reformulations, with the last reformulation exposing the solution. Herbert Simon (1969) wrote that

> all mathematics exhibits in its conclusions only what is already implicit in its premises.... Hence all mathematical derivation can be viewed simply as change in representation, making evident what was previously true but obscure.
>
> This view can be extended to problem solving—solving a problem simply means representing it so as to make the solution transparent.

Intuitively, however, reformulation is different from an elementary search step; refining this intuition is the subject of future research.

The Uses of Reformulation

The workshop presentations showed that reformulation has many uses. For example, a number of different reformulations were shown to improve problem solver performance in terms of speed of solvers or memory use. Reformulation can also improve the human interface to problem solvers. Choueiry and Noubir demonstrated a reformulation that enables compact representation of families of solutions and showed how these solutions behave when additional constraints are imposed on the problem instance. Three papers used reformulation to promote software reuse: First, the reformulation by Bouquard and Lenté permitted the use of a standard constraint in a commercial constraint solver; second, the paper by Bockmayr and Dimopolous permitted the use of standard linear and mixed-programming solvers; and third, the paper by Selman and Kautz enabled the use of publicly available SAT solving programs.

Amarel's work on representation was aimed at providing a system that would be capable of automatically selecting the most appropriate representation for solving a particular problem. The *customer,* that is, the human who posed the problem, is thus insulated from the task of becoming an expert in modeling or problem solving; the system encodes all the necessary knowledge and machinery to select the best representation. In this way, reformulation is used to make it possible for more people to use sophisticated representations and problem-solving techniques.

Reformulations can also reveal properties in the original domain. For exam-

ple, some of the constraint propagation observed in the SAT representations of planning problems can be incorporated into planners that operate on the original planning problems. Knowledge of these aspects of the original domain can, in turn, lead to better modeling.

Theories of Reformulation

Clearly, even now, systems capable of automatically selecting the most appropriate formulation for a problem are still not widely available. However, the statement of the ultimate goal of reformulation brings up a number of important issues. First, what is "appropriate"? Defining a framework in which representations can be assessed is an important challenge.

A second issue is generalizing results concerning the success and failure of reformulations. There are currently few results to draw on for generalization, and some of the results are apparently contradictory. For example, Bacchus and van Beek (1998) showed that rewriting *k*-ary CSPs as binary CSPs tended to decrease the time to solution of backtracking search, yet for the limited case of quasigroup completion, the powerful all-diff constraint propagation makes the *k*-ary representation superior. However, there are some general guidelines to reformulation that are known. For example, reformulation by adding constraints aids enforcement of consistency, but reformulation by eliminating constraints aids decomposition and increases the chances of structural tractability. As more specific results are known, more generalizations will become possible.

Conclusion

Reformulation is becoming increasingly popular as a way of solving hard problems. In recent years, the success stories of reformulation have kindled new interest, and as more and more techniques are developed to solve problems, reformulation will become more popular as researchers attempt to make use of the library of existing results. There is clearly much work to be done, from characterizing the conditions under which reformulation is successful to defining frameworks for automatic reformulation and clarifying the role reformulation plays in problem solving. However, as the previous discussion indicates, reformulation has the power to do much more than simply allow the reuse of results. It can also lead to improved problem-solving systems and improved understanding in a domain. In closing, we look forward to continued progress in the area of reformulation.

Acknowledgments

I would like to thank all the workshop participants and my cochair, Mihaela Sabin, for making the workshop a success. I would also like to thank Lise Getoor for her help in formulating the workshop proposal and Berthe Choueiry for her help in writing the workshop summary.

References

Amarel, S. 1968. On Representations of Problems of Reasoning about Actions. *Machine Intelligence* 3:2–22.

Bacchus, F., and van Beek, P. 1998. On the Conversion between Nonbinary and Binary Constraint-Satisfaction Problems. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, 326–333. Menlo Park, Calif.: American Association for Artificial Intelligence.

Bylander, T. 1997. A Linear Programming Heuristic for Optimal Planning. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, 694–699. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1194–1200. Menlo Park, Calif.: American Association for Artificial Intelligence.

Papadimitriou, C. 1994. *Complexity Theory*. Reading, Mass.: Addison Wesley.

Simon, H. A. 1969. *The Sciences of the Artificial: The Karl Compton Lecture Series*. Cambridge, Mass: The MIT Press.

Jeremy Frank is a researcher in the Planning and Scheduling Group at NASA Ames Research Center. He holds a B.A. in mathematics from Pomona College and a Ph.D. in computer science from the University of California at Davis. His current interests include local search algorithms, techniques for solving planning problems with complex resource and temporal constraints, and properties of phase transitions in combinatorial search problems. His e-mail address is frank@ptolemy.arc.nasa.gov.