THE ROYAL SOCIETY

Published online 10 June 2003

Abstraction and reformulation in artificial intelligence

Robert C. Holte[1*] and Berthe Y. Choueiry[2]

[1]*Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada*
[2]*Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA*
(*choueiry@cse.unl.edu*)

This paper contributes in two ways to the aims of this special issue on abstraction. The first is to show that there are compelling reasons motivating the use of abstraction in the purely computational realm of artificial intelligence. The second is to contribute to the overall discussion of the nature of abstraction by providing examples of the abstraction processes currently used in artificial intelligence. Although each type of abstraction is specific to a somewhat narrow context, it is hoped that collectively they illustrate the richness and variety of abstraction in its fullest sense.

Keywords: artificial intelligence; abstraction; reformulation; search; constraint satisfaction

1. INTRODUCTION

In the early days of AI much attention was paid to human reasoning. One reason for this was the aim of developing computational models of human cognition. This branch of AI later joined with like-minded researchers in philosophy, psychology and the neurosciences to create cognitive science. The second reason for AI researchers to study human reasoning was to obtain suggestions about how to program a computer to perform a particular cognitive task, such as playing chess. The aim, in this case, was not to develop a theory of human cognition, but a working program that performed the given task with human competence or better. The computer was not constrained to mimic human thought, but human methods, even if they could be encoded only approximately, seemed a natural starting point for the development of the computer programs. For example, in 1950 Claude Shannon proposed encoding the knowledge and basic thought processes of human chess experts to create an expert-level chess-playing program (Shannon 1950).

In analysing effective, creative, human problem solving it became clear that success often hinged on looking at a given problem from different points of view. One naturally starts with the point of view suggested by the problem statement, but if that does not lead to success, a good problem solver will change the problem to a more perspicuous form. For example, many of the techniques in George Polya's influential book *How to solve it* (Polya 1945) are of this kind: replace a problem by its generalization, introduce an auxiliary element, adopt a good notation, etc. In some cases the solution, or insolubility, of the modified problem directly gives the answer to the original problem, whereas in others solving the modified problem is meant to give insight into the original problem or guidance about how it might be solved. To illustrate the general idea consider the 8×8 mutilated array shown in figure 1. The aim is to determine if it is possible to exactly cover the array with 2×1 tiles. This version of the problem is difficult for humans, even though there is a very simple solution. The solution can be seen clearly by making two modifications to the original problem. The first is to add colour to the squares in a checkerboard fashion as shown in figure 2. The second modification is to forego reasoning about how to place the tiles on the array and reason instead about the number of black (B) and white (W) squares and how these numbers change when a tile is laid on the array. However a tile is laid, B and W are each reduced by 1, so the original question has been transformed to 'is it possible to reduce B and W to 0 by subtracting 1 from each of them some number of times?' It is now obvious that this is impossible if B and W are different, as they are in the mutilated array. This proves that the array cannot be exactly covered with 2×1 tiles. It is worth noting that B and W being equal does not guarantee that the array can be covered by tiles (see figure 3): insolubility of the modified problem guarantees insolubility of the original, but not vice versa. Just because humans solve the mutilated array problem in this way does not oblige computers to do so. However, as a matter of fact, AI programs for problem solving or theorem proving solve the modified version of the mutilated array problem much more efficiently than they solve the original. This was first observed in the 1960s (McCarthy 1964; Newell 1965) and is still true today.

The general idea of changing the statement, or representation, of a given problem is called, in AI, change of representation or reformulation. Herb Simon argued that there was an intimate connection between reformulation and problem solving:

> 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

* Author for correspondence (holte@cs.ualberta.ca).

One contribution of 16 to a Theme Issue 'The abstraction paths: from experience to concept'.

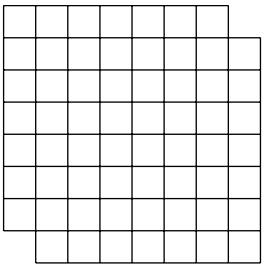

Figure 1. 8×8 Mutilated array.

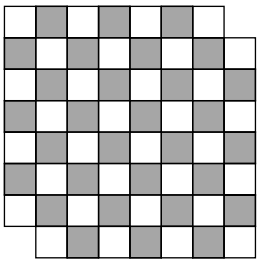

Figure 2. Adding colour to the 8×8 mutilated array.

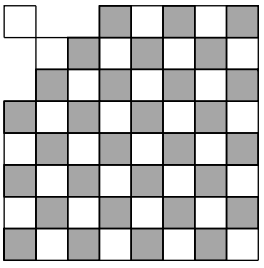

Figure 3. 8×8 Array with B = W that cannot be tiled.

> solving—solving a problem simply means representing it so as to make the solution transparent.
>
> (Simon 1981, p. 153)

Abstraction is a kind of reformulation, but not all reformulations are abstractions. For example, the addition of colour to the mutilated array is certainly not an abstraction, but the mapping from an array to a pair of numbers (B and W) is an abstraction. Precise definitions of 'abstraction', distinguishing it from the all-encompassing notion of reformulation, exist in certain specific contexts, but a precise, universal definition of abstraction is not yet available. The failure to produce a satisfactory definition of abstraction reflects the difficulty of giving a characterization that is general enough to encompass the great diversity of techniques that fall into the intuitive category of 'abstractions' while at the same time being specific enough to exclude reformulations that intuitively are not abstractions.

It is hoped that this article contributes in two ways to the aims of this special issue on abstraction. The first is to show that issues that might be thought to arise strictly in the realm of human reasoning arise in much the same form in the purely computational realm of AI. Although there are undoubtedly many purely psychological reasons underlying abstraction in human behaviour, there are certainly compelling computational reasons motivating the use of abstraction.

Second, this article contributes to the overall discussion of the nature of abstraction by providing examples of the abstraction processes that are currently used in AI. Although each type of abstraction is specific to a somewhat narrow context, it is hoped that, collectively, they illustrate the richness and variety of abstraction in its fullest sense.

This article does not give a comprehensive survey of work in AI relevant to the subject of this special *Transactions* issue. Most importantly, no mention is made of learning, the process of extracting general rules of behaviour or knowledge from experience. The field of machine learning is large and very active, and includes highly relevant topics such as the creation of abstract symbolic representations (Drummond 1999, 2002) and the invention of useful novel concepts (Muggleton 1988; Saitta & Zucker 1998; Zucker *et al.* 2002). The present article focuses on techniques that do not involve learning: they create abstractions by analysing the given problem, not by accumulating and then generalizing a wealth of experience. A second important topic omitted from this paper is the use of abstraction to summarize the results of complex computations so that they can be understood by humans, as is done, for example, by Gruber & Gautier (1993) and Mallory *et al.* (1996) for qualitative simulations, by Huang (1994) for proofs of mathematical theorems, and by Shahar (1997) for medical applications. Even within this narrower focus, the article is not a survey of all known abstraction methods. It presents the methods that are at present the most common and most successful. To see the full range of techniques, the reader may consult the *Proceedings of the International Symposium on Abstraction, Reformulation, and Approximation* (Choueiry & Walsh 2000; Koenig & Holte 2002).

2. PLANNING AND PROBLEM SOLVING

The earliest and most widespread and successful use of abstraction in AI is in planning and problem solving. These are tasks in which the aim is to find a sequence of 'actions' that transforms an initial 'state' into some desired goal state. The most familiar example of a planning task is route planning. The aim, in this case, is to find a route from a starting point to a destination, and the actions are defined by the roads or other allowable means of transportation. A second familiar example is solving a puzzle, such as Rubik's Cube. The aim here is to find a sequence of moves that transforms the initial, scrambled arrangement of the puzzle into the orderly arrangement that serves as the goal.

As early as 1961 Minsky proposed using abstraction to solve such problems (Minsky 1961). Problem solving proceeds in two stages. The first stage focuses on the most important or most difficult aspects of the problem, entirely ignoring all other details. Once the abstract problem is solved, its solution is used, in the second stage, as a basis for the full solution to the original problem. In planning a trip from the Eiffel Tower to the Sydney Opera House, for example, one would work out the entire route from Paris to Sydney before considering the route to take within Paris. The main implementations of this abstraction technique are reported in Sacerdoti (1974), Knoblock (1991) and Holte *et al.* (1996).

Technically, the biggest challenge in this approach is guaranteeing that it is possible to 'flesh out' the abstract solution into a complete solution. This is guaranteed in certain special circumstances (for example, in Holte *et al.* 1996) but not in general, and when it does not hold, this approach can be much less efficient than methods that do not use abstraction (Bacchus & Yang 1994).

However, there is an entirely different way to use the very same abstractions to guide problem solving. Instead of using the abstract solution as a skeleton for the full solution, one can simply use the length of the abstract solution as an estimate of the length of the final solution. This may seem to be throwing away a great deal of information, but by not trying to conform to the abstract solution step by step, the technical problem just mentioned is avoided.

This alternative approach is called heuristic search, because the function that estimates solution length is called the heuristic function. Heuristic search dates to the 1960s (Doran & Michie 1966; Hart *et al.* 1968), but the idea of using abstraction to create a heuristic function was not proposed until 1979 (Guida & Somalvico 1979; Gasching 1979). This approach to creating heuristic functions was popularized by the influential book *Heuristics* (Pearl 1984) but not fully mechanized for several years (Mostow & Prieditis 1989; Prieditis 1993).

A formal, general definition for abstraction in the context of heuristic search is given by Prieditis (1993). In English, it can be roughly paraphrased as follows: problem A is an abstraction of problem B if the solution to A is guaranteed to be shorter than, or the same length as, the solution for B.

This is broader than the intuitive notion of abstraction, because there does not have to be any relationship at all between the structure or content of the original problem (B) and the abstract problem (A). All that is required is that a certain relationship should hold between the lengths of their solutions. This is understandable, in the context of heuristic search, because only the length of the abstract solution is used. In reality, however, systems that use abstraction to create heuristics automatically invariably construct abstract problems that are intimately related to the original problem. Indeed, the method was originally described as using a simplified version of a problem to estimate solution length.

The state of the art in using abstraction to create heuristic functions is the pattern database technique introduced by Joe Culberson and Jonathan Schaeffer in 1994 (Culberson & Schaeffer 1994, 1996, 1998). It is well illustrated by its application to Rubik's Cube (Korf 1997). To estimate how many moves it will take to unscramble the cube from some specific scrambled state, consider the simpler problem in which all the corner pieces have been painted black to be indistinguishable from each other. To unscramble this abstract cube is a much easier matter—it only requires getting all the edge pieces into their goal positions. The number of moves it takes to do that is obviously an underestimate of the number of moves required to solve the original problem, which must unscramble the corners as well as the edges. By itself this is a rather poor estimate of solution length, because the corners are in no way taken into account, but it can be combined with a different abstraction of the problem in which the edge pieces are all painted black and the corners must be solved. Together these are very effective in guiding the search for a solution to the original problem.

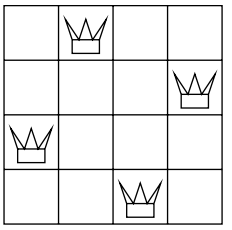

Figure 4. A solution of the 4-queens problem.

The general idea illustrated in this example, by painting all the corner pieces black, is called domain abstraction. Its essence is to transform a problem by making distinct elements of the problem indistinguishable: the eight corner pieces are distinguishable from each other in the original problem but not after they have been painted black. Note, however, that there are still eight corner pieces. Domain abstraction does not change the number of problem elements, it just reduces their distinguishability. A different abstraction technique, called projection, takes the opposite approach: it abstracts a problem by eliminating problem elements. Projection has recently proven very useful in creating heuristic functions for planning (Edelkamp 2001).

3. CONSTRAINT PROCESSING

Constraint processing is a paradigm for expressing and solving decision problems in engineering and management. As in planning, the use of an abstract solution as a skeleton to guide the construction of a full solution has also been exploited in this area (Bistarelli *et al.* 2000). Beyond this general use of abstraction, particular attention has been paid to modelling and the automatic detection and use of symmetry relations.

A familiar example of a CSP is the 4-queens problem, where the task is to place four queens on a 4×4 chessboard in such a way that no two queens attack each other (figure 4). In general, a CSP is defined by three elements: (i) a set of decisions to be made; (ii) a set of choices available for each decision; and (iii) a set of constraints that restrict the acceptable combination of choices for the decisions. The CSP task is to find a consistent solution, i.e. a choice for each decision such that all the constraints are satisfied. In the 4-queens problem, the decisions are where to place each queen, the choices for each queen are any of the 16 squares, and the constraints are that the queens must not be placed on the same square or on squares in the same row, column or diagonal.

The process of modelling a problem by constructing a representation that can be processed by a computer is an important abstraction process at which humans excel but computers do not. This is perhaps the most challenging aspect of abstraction in AI and the closest to the research in psychology and cognitive science.

CSPs are invariably modelled as follows. Each decision is represented by a variable. The a_d choices available for a particular decision d are represented by the integers $1 \ldots a_d$, called the 'domain' of the variable. Thus, choosing

alternative k for decision d is modelled as assigning value k to variable V_d. Each constraint is modelled as a set of allowable combinations of assignments of values to variables.

For example, one natural way to model the 4-queens problem in this framework is to define a decision variable for each square on the board. The square can be either empty (value 1) or have a queen (value 2). The constraints specify that exactly four of the decision variables have a value of 2 ('queen in this square') and that there cannot be two queens in the same row, column or diagonal. Because there are 16 variables (one for each square) and each can take on two possible values, there are a total of 2^{16} (65 536) possible assignments of values to the decision variables in this method of modelling the 4-queens problem.

There are other ways of modelling the 4-queens problem within the CSP framework. One alternative is to treat each row on the board as a decision variable. The values that can be taken by each variable are the four column positions in the row. This formulation yields 4^4 (256) possibilities, which is significantly fewer than the previous one. Bernard Nadel explored eight different possible formulations of this simple problem (Nadel 1990), and found some were much easier to solve than others using a standard CSP solver. This example illustrates how the initial formulation, or model, affects the number of possibilities to be examined, and thus the efficiency of problem solving.

Most of the research on abstraction in constraint processing has focused on manipulating a given formulation of a problem. Because the problem is represented by its variables, their domains and the constraints, the various abstractions explored operate on a combination of one or more of these components.

The most common abstraction applied to domains in constraint processing is based on symmetry. For example, solutions to the N-queens problem can be transformed into other solutions based on symmetries about the horizontal axis, the vertical axis and various diagonals (Mozetič 1991). When the symmetry is known in advance, it can be exploited by the problem solver to avoid unnecessary exploration of equivalent solutions. Another typical example is the pigeonhole problem where one has to place $(n + 1)$ objects into n pigeonholes, such that each pigeonhole contains, at most, one object. To a human it is immediately obvious that this is impossible, but to reach this conclusion one must abstract the problem to a counting argument, much as was done to solve the mutilated array problem. In the absence of any such abstraction, it is necessary to try all the different methods of assigning each object to each pigeonhole. There are a vast number of possibilities to consider. However, if the problem solver is instructed how to exploit the symmetries—all objects act identically, as do all pigeonholes—the insolubility of the problem can be established with very little computation. Such symmetries have been studied since 1874 (Glaisher 1874) and have recently received increased attention (Fillmore & Williamson 1974; Brown *et al.* 1988; Puget 1993; Backofen & Will 1999; Gent & Smith 2000; Hentenryck 2002).

The pigeonhole problem is an example of symmetries based on an equivalence relation among the values for the variables. A set of values in the domain of a particular variable are said to be equivalent if they all produce identical results in the context of the problem being solved. This notion of equivalence allows a variable's domain to be partitioned into equivalence classes, where all the values in an equivalence class are equivalent. This allows CSP problems to be solved much more quickly because instead of considering all the different values in the domain it is necessary only to consider one representative of each class (Hubbe & Freuder 1989; Freuder 1991; Ellman 1993; Haselböck 1993; Choueiry *et al.* 1995; Freuder & Sabin 1995; Weigel *et al.* 1996; Weigel & Faltings 1997; Choueiry & Noubir 1998). *Dynamic bundling* is a technique for efficiently discovering equivalence relations during problem solving (Choueiry & Davis 2002). It has been shown to yield multiple solutions to a CSP with significantly less effort than is necessary to find a single solution.

A common abstraction that is applied to the variables in a CSP is decomposition. For example, under some conditions decisions that tightly interact can be considered together, in isolation from the rest of the problem. Another technique for abstracting variables is aggregation. Consider a graph-colouring problem where the nodes of a graph need to be assigned a colour such that no two adjacent nodes have the same colour. The natural formulation of this problem as a CSP has the nodes as the decision variables and the colours as the possible values. Nodes that are not directly connected to one another but are connected to the same other nodes in the graph can be given the same colour in any solution to the problem. Thus, the variables representing these nodes can be pooled together as a single variable. This aggregation, which can be applied repeatedly, reduces the number of variables in the CSP, consequently reducing the cost of finding a solution. This procedure will not necessarily find all possible solutions, but it is guaranteed to find a solution to the problem if one exists.

Finally, there are at least two types of constraint abstraction. In the first type, one or more constraints, judiciously chosen, can be eliminated to transform a given difficult problem into a tractable one. If the tractable problem is shown to be unsolvable, the original problem will also have been shown to be unsolvable, as in the example of the mutilated array problem. By contrast, if a solution is found for the tractable problem it can be used as a guide to finding a solution to the original problem.

The second type of constraint abstraction consists in replacing a non-binary constraint (i.e. a constraint that applies to several variables simultaneously) by a network of binary constraints (i.e. constraints that apply to two variables at the same time). In general, binary constraints are preferable to non-binary ones because they are easier to handle, and because most known techniques in constraint processing have been developed for binary constraints. This process is called constraint decomposition and it can be done exactly or approximately.

An efficient form of approximate constraint decomposition is projection. The simplest way to illustrate projection is by a geometric example in which there are two continuous variables, x and y, and a constraint $C(x, y)$ specifying the allowable combinations of values for x and y. Any such constraint can be pictured as a region, or

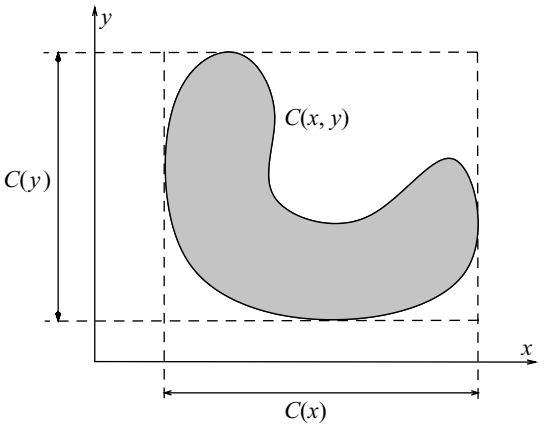

Figure 5. Projection of two-dimensional constraints onto individual dimensions.

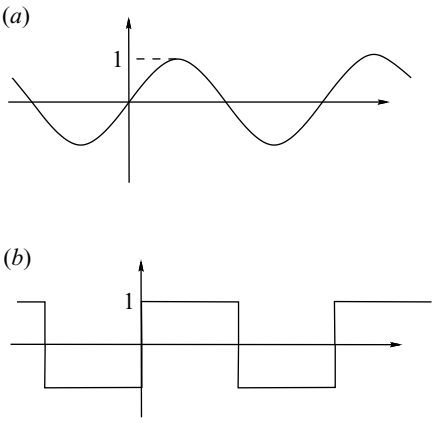

Figure 6. (*a*) Sine function and a (*b*) square function.

regions, in the x–y plane, as shown in figure 5—only (x, y) combinations in the shaded region are permitted by constraint $C(x, y)$. As the figure suggests, C might be a very complex shape. It can be approximated by two very simple constraints, $C(x)$ and $C(y)$ created by projecting the $C(x, y)$ region onto the x and y axes, respectively. The two projections in this example are simple intervals on the x and y axes. This idea generalizes to projecting any number of dimensions onto any smaller number of dimensions, and thus can be used to project non-binary constraints (e.g. involving six variables) to create binary constraints (involving two variables).

4. REASONING ABOUT PHYSICAL SYSTEMS

The goal of this area of AI is to construct a model of the dynamic behaviour of a physical system, and then process the model to predict or explain the behaviour of the system under given conditions. For example, it aims at predicting the behaviour of an electromechanical device given its components, their functionalities, the way they are connected and the operating conditions. In the area of reasoning about physical systems, abstraction is usually identified with the concept of simplification. Examples of the use abstraction in reasoning about physical systems are abundant at the three main stages of the reasoning process, namely when choosing or building a model of the physical system, processing the model and explaining the output (Choueiry *et al.* 2003).

A physical system can be described at various levels of abstraction, from the atomic level to the macroscopic one, depending on the intended use of the model. For instance, a wire can be described as an electrical conductor, which is, in turn, either an ideal conductor or a resistor (Nayak & Joskowicz 1996). The resistor can be modelled as a constant, thermal or temperature-dependent resistor, etc. The appropriate level of detail depends on the reasoning task to be performed and the aspects of the behaviour of the system that need to be explained or predicted. The final model must be detailed enough to capture correctly the behaviour sought, but ideally would include no unnecessary details. These considerations guide the selection of model fragments for the various components of the device and their composition, by compositional modelling, into a global model that describes the behaviour of the system (Falkenhainer & Forbus 1991; Rickel & Porter 1994; Levy *et al.* 1997).

This global model can be a system of algebraic, differential equations or a set of qualitative, causal relations (Nayak & Joskowicz 1996). Before being processed, this model is usually further simplified. For example, negligible terms can be dropped, mathematical equations can be made linear, and systems of equations can be decomposed. Such simplifications are common abstractions carried out by engineers and mathematicians. They may affect the precision of the ultimate result, but are typically carried out to reduce the computational cost of the subsequent step, which aims at finding a solution. This illustrates the typical use of abstraction in AI and engineering: the representation is 'simplified' to speed up the problem-solving process while knowingly affecting the quality of the outcome within a (sometimes bounded) threshold.

It is important to recognize that some of the commonly used 'simplification' techniques only result in simplification in certain circumstances: in different circumstances, exactly the same technique might complicate matters instead of simplifying them. For example, consider the sine function $y = \sin(x)$ and its approximation by the square function shown in figure 6. These functions are being shown in what is called the temporal domain—the x-axis represents time. In this domain the square function is clearly an abstraction of the sine function, because all y-values in $[0, 1]$ are abstracted to 1, and all values in $[-1, 0)$ are abstracted to -1. As long as the problem solving remains in the temporal domain, the square function constitutes an abstraction of the sine function. However, physicists and electrical engineers frequently transform problems from the temporal domain into what is called the frequency domain. Oddly, when this is done, the square function becomes much more complex than the sine function. It is thus important to make explicit the precise measure of simplicity used and the conditions under which it holds.

In summary, abstraction is ubiquitous in reasoning about physical systems. In a detailed study, abstraction techniques have been shown to be elaborate combinations of the following 'elementary' operations: replacement, decomposition, aggregation and focusing (Choueiry *et al.* 2003). Although most of the above techniques have been discussed in the AI literature, they originate from, and are useful in, a variety of science and engineering fields.

5. THEORIES OF ABSTRACTION

Theorem proving, where an abstract proof is used to guide building a concrete proof, is one of the early areas of AI where a formal definition and characterization of the abstraction process has been attempted (Plaisted 1981; Cremonini *et al.* 1990; Giunchiglia & Walsh 1992; Nayak & Levy 1995). These efforts have led to the development of formal theories of abstraction, which fall into two categories, semantic (Nayak & Levy 1995) and syntactic (Giunchiglia & Walsh 1992). Typically, these theories are restricted to logical systems and to techniques that preserve consistency and correctness of proofs.

These logical theories fail to provide the vocabulary necessary to characterize the practical aspects of the abstraction process. Another shortcoming is that they ignore the relevance, in the abstraction process, of the problem-solving goals, although one notable exception to this is the formal theory of behaviour-preserving equivalence defined by Lowry (1989). The impact of abstraction on the efficiency of the reasoning process and the quality of its outcome has led to the proposal of 'practical' theories of abstraction (Weld & Addanki 1990; Knoblock *et al.* 1991; Struss 1993; Davis 1995; Choueiry *et al.* 2003). For example, the theory proposed in Choueiry *et al.* (2003) attempts to clearly specify and precisely relate the elements in the problem that are being abstracted away. It also provides the means to express logical, quantitative and qualitative criteria to assess the success of the abstraction process.

An extensive discussion of the theories of abstraction in AI can be found in this special issue in the article by Zucker (2003).

6. CONCLUSIONS

This paper has surveyed the most common notions of abstraction currently in use in three areas of AI. Even in a single area, there are diverse methods of abstraction, and different methods for using the information produced by abstraction. If there is a theme common to all of these it is the very general idea of reducing a problem by eliminating, shrinking, grouping or simplifying one or more of the various elements in the problem definition.

In all the methods surveyed, the goal of abstraction is to speed up a computation. Only in special cases is speed-up guaranteed. Most often, it is impossible to predict ahead of time whether the computational overheads involved in creating and using abstractions are greater or smaller than the computational savings that result from exploiting the information produced by the abstraction.

Systems that use abstractions have varying degrees of autonomy in terms of the abstractions they use. Some systems are able to use abstractions provided by humans, but are not able to generate the abstractions (e.g. Culberson & Schaeffer 1996; Korf 1997). Other systems are able to build abstractions, in some cases taking into account dynamically occurring circumstances during their execution, but are very specific in the abstractions they build and the conditions under which abstractions are built. Choueiry & Davis (2002) is an example of this type of system. Finally, there are systems that search for good abstractions in a broad, and sometimes very diverse, family of possible abstractions (e.g. Korf 1980; Prieditis 1993; Hernádvölgyi 2001). Even the most autonomous of these systems depends, to a significant extent, on being given an initial problem formulation that is amenable to the types of abstraction the system employs. Thus, there is still a need for humans to apply their creative ingenuity and insight to creating a good problem formulation, although it is hoped that by automating abstraction and other reformulation techniques the burden on humans has been reduced.

R.C.H. thanks the Natural Sciences and Engineering Research Council of Canada for the financial support that made this research possible. B.Y.C. is supported by a CAREER award no. 0133568 from the National Science Foundation.

REFERENCES

Bacchus, F. & Yang, Q. 1994 Downward refinement and the efficiency of hierarchical problem solving. *Artif. Intell.* **71**, 43–100.

Backofen, R. & Will, S. 1999 Excluding symmetries in constraint-based research. In *5th Int. Conf. Principles and practice of constraint programming, CP '99, Alexandria, VA, USA, October 1999. Lecture notes in artificial intelligence 1713* (ed. J. Jaffar), pp. 73–87. Springer.

Bistarelli, S., Codognet, P. & Rossi, F. 2000 An abstraction framework for soft constraints and its relationship with constraint propagation. In *4th Int. Symp. on Abstraction, Reformulation and Approximation, SARA 2000, Horseshoe Bay, USA, 26–29 July 2000. Lecture notes in artificial intelligence 1864* (ed. B. Y. Choueiry & T. Walsh), pp. 71–86. Springer.

Brown, C. A., Finkelstein, L. & Purdom Jr, P. W. 1988 Backtrack searching in the presence of symmetry. In *Applied algebra, algebraic algorithms and error-correcting codes* (ed. T. Mora), pp. 99–110. Springer.

Choueiry, B. Y. & Davis, A. M. 2002 Dynamic bundling: less effort for more solutions. In *5th Int. Symp. on Abstraction, Reformulation and Approximation, SARA 2002, Kananaskis, Alberta, Canada, 2–4 August 2002. Lecture notes in artificial intelligence 2371* (ed. S. Koenig & R. Holte), pp. 64–82. Springer.

Choueiry, B. Y. & Noubir, G. 1998 On the computation of local interchangeability in discrete constraint satisfaction problems. In *Proc. AAAI-98, Madison, WI*, pp. 326–333. (Revised version KSL-98-24, ksl-web.Stanford.edu/KSL_Abstracts/KSL-98-24.html.)

Choueiry, B. Y. & Walsh, T. (eds) 2000 *Proc. 4th Int. Symp. on Abstraction, Reformulation and Approximation, SARA 2000, Horseshoe Bay, USA, 26–29 July 2000. Lecture notes in artificial intelligence 1864.* Springer.

Choueiry, B. Y., Faltings, B. & Weigel, R. 1995 Abstraction by interchangeability in resource allocation. In *Proc. 14th IJCAI, Montreal, Quebec, Canada, 20–25 August 1995*. pp. 1694–1701. Morgan Kaufmann.

Choueiry, B. Y., Iwasaki, Y. & McIlraith, S. 2003 Towards a practical theory of reformulation for reasoning about physical systems. *Artif. Intell.* (Submitted.)

Cremonini, R., Marriott, K. & Søndergaard, H. 1990 A general theory of abstraction. In *Proc. 4th Aust. Joint Conf. Artificial Intelligence, Australia*, pp. 121–134.

Culberson, J. C. & Schaeffer, J. 1994 Efficiently searching the 15-puzzle. Technical report, Department of Computing Science, University of Alberta.

Culberson, J. C. & Schaeffer, J. 1996 Searching with pattern databases. In *Advances in artificial intelligence, 11th Biennial Conf. of Canadian Society for Computational Studies of Intelligence, AI '96, Toronto, Canada, 21–24 May 1996. Lecture notes in artificial intelligence 1081* (ed. G. McCalla), pp. 402–416. Springer.

Culberson, J. C. & Schaeffer, J. 1998 Pattern databases. *Comput. Intell.* **14**, 318–334.

Davis, E. 1995 Approximation and abstraction in solid object kinematics. Technical report TR706. New York University.

Doran, J. E. & Michie, D. 1966 Experiments with the graph traverser program. *Proc. R. Soc. Lond.* A **294**, 235–259.

Drummond, C. 1999 A symbol's role in learning low level control functions. PhD thesis, Computer Science Department, University of Ottawa, Canada.

Drummond, C. 2002 Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *J. Artif. Intell. Res.* **16**, 59–104.

Edelkamp, S. 2001 Planning with pattern databases. In *Proc. 6th Eur. Conf. on Planning (ECP-01)*.

Ellman, T. 1993 Abstraction via approximate symmetry. In *Proc. 13th IJCAI, Chambéry, France*, pp. 916–921.

Falkenhainer, B. & Forbus, K. D. 1991 Compositional modeling: finding the right model for the job. *Artif. Intell.* **51**, 95–143.

Fillmore, J. P. & Williamson, S. 1974 On backtracking: a combinatorial description of the algorithm. *SIAM J. Comput.* **3**, 41–55.

Freuder, E. C. 1991 Eliminating interchangeable values in constraint satisfaction problems. In *Proc. AAAI-91, Anaheim, CA*, pp. 227–233.

Freuder, E. C. & Sabin, D. 1995 Interchangeability supports abstraction and reformulation for constraint satisfaction. In *Symp. on Abstraction, Reformulation and Approximation, SARA '95, Ville d'Esterel, Canada, August 1995.*

Gasching, J. 1979 A problem similarity approach to devising heuristics: first results. In *Proc. 11th IJCAI*, pp. 301–307.

Gent, I. P. & Smith, B. M. 2000 On reformulation of constraint satisfaction problems. In *Proc. 14th ECAI, Berlin*, pp. 599–603.

Giunchiglia, F. & Walsh, T. 1992 A theory of abstraction. *Artif. Intell.* **57**, 323–389.

Glaisher, J. 1874 On the problem of the eight queens. *Phil. Mag. Ser. 4* **48**, 457–467.

Gruber, T. B. & Gautier, P. O. 1993 Machine-generated explanations of engineering models: a compositional modeling approach. In *Proc. 13th IJCAI, Chambéry, France*, pp. 1512–1508.

Guida, G. & Somalvico, M. 1979 A method for computing heuristics in problem solving. *Info. Sci* **19**, 251–259.

Hart, P., Nilsson, N. J. & Raphael, B. 1968 A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet.* **4**, 100–107.

Haselböck, A. 1993 Exploiting interchangeabilities in constraint satisfaction problems. In *Proc. 13th IJCAI, Chambéry, France*, pp. 282–287.

Hentenryck, P. V. (ed.) 2002 *Proc. 8th Int. Conf. on Principle and Practice of Constraint Programming* (*CP '02*), *Ithaca, NY, USA, 9–13 September 2002. Lecture notes in computer science 2470*. Springer.

Hernádvölgyi, I. T. 2001 Searching for macro operators with automatically generated heuristics. In *Advances in Artificial Intelligence—Proc. 14th Biennial Conf. Can. Soc. Comput. Stud. Intell. (LNAI 2056)*, pp. 194–203.

Holte, R. C., Mkadmi, T., Zimmer, R. M. & MacDonald, A. J. 1996 Speeding up problem-solving by abstraction: a graph-oriented approach. *Artif. Intell.* **85**, 321–361.

Huang, X. 1994 Reconstructing proofs at the assertion level. In *Proc. 12th Conf. on Automated Deduction, Nancy, France, 26 June–1 July 1994* (ed. A. Bundy), pp. 738–752. Springer.

Hubbe, P. D. & Freuder, E. C. 1989 An efficient cross product representation of the constraint satisfaction problem search space. In *Proc. AAAI-92, San Jose, CA*, pp. 421–427.

Knoblock, C. A. 1991 Automatically generating abstractions for planning. *Artif. Intell.* **68**, 243–302.

Knoblock, C. A., Tenenberg, J. D. & Yang, Q. 1991 Characterizing abstraction hierarchies for planning. In *Proc. AAAI-91, Anaheim, CA*, pp. 692–697.

Koenig, S. & Holte, R. C. (eds) 2002 *Proc. 5th Int. Symp. on Abstraction, Reformulation and Approximation, SARA 2002, Kananaskis, Alberta, Canada, 2–4 August 2002. Lecture notes in artificial intelligence 2371*. Springer.

Korf, R. E. 1980 Toward a model of representation changes. *Artif. Intell.* **14**, 41–78.

Korf, R. E. 1997 Finding optimal solutions to Rubik's Cube using pattern databases. In *Proc. 14th Nat. Conf. on Artificial Intelligence (AAAI-97)*, pp. 700–705.

Levy, A. Y., Iwasaki, Y. & Fikes, R. 1997 Automated model selection for simulation based on relevance reasoning. *Artif. Intell.* **96**, 351–394.

Lowry, M. R. 1989 Algorithm synthesis through problem reformulation. PhD thesis, Computer Science Department, Stanford University, CA, USA.

McCarthy, J. 1964 A tough nut for proof procedures. Stanford Artificial Intelligence Project memo 16.

Mallory, R. S., Porter, B. W. & Kuipers, B. J. 1996 Comprehending complex behavior graphs through abstractions. In *10th Int. Workshop on Qualitative Physics. AAAI Technical Report WS-96-Ql, Fallen Leaf Lake, CA*, pp. 137–146.

Minsky, M. 1961 Steps toward artificial intelligence. *Proc. IRE* **49**, 8–30.

Mostow, J. & Prieditis, A. 1989 Discovering admissible heuristics by abstracting and optimizing: a transformational approach. In *Proc. 11th IJCAI*, pp. 701–707.

Mozetič, I. 1991 Hierarchical model-based diagnosis. *Int. J. Man–Machine Stud.* **35**, 329–362.

Muggleton, S. 1988 A strategy for constructing new predicates in first order logic. In *Proc. 3rd Eur. Working Session on Learning* (ed. D. Sleeman), pp. 123–130. London: Pitman Publishing.

Nadel, B. 1990 Representation selection for constraint satisfaction: a case study using *N*-queens. *IEEE Expert* **5**, 16–24.

Nayak, P. P. & Joskowicz, L. 1996 Efficient compositional modeling for generating causal explanations. *Artif. Intell.* **83**, 193–227.

Nayak, P. P. & Levy, A. Y. 1995 A semantic theory of abstractions. In *Proc. 14th IJCAI, Montreal, Canada*, pp. 196–203.

Newell, A. 1965 Limitations of the current stock of ideas about problem solving. In *Electronic information handling* (ed. A. Kent & O. E. Taulbee), pp. 195–208. Spartan Books.

Pearl, J. 1984 *Heuristics: intelligent search strategies for computer problem solving*. Reading, MA: Addison-Wesley.

Plaisted, D. A. 1981 Theorem proving with abstraction. *Artif. Intell.* **16**, 47–108.

Polya, G. 1945 *How to solve it: a new aspect of mathematical method*. Princeton University Press.

Prieditis, A. E. 1993 Machine discovery of effective admissible heuristics. *Machine Learn.* **12**, 117–141.

Puget, J.-F. 1993 On the satisfiability of symmetrical constrained satisfaction problems. In *ISMIS '93*, pp. 350–361.

Rickel, J. & Porter, B. 1994 Automated modeling for answering prediction questions: selecting the time scale and system boundary. In *Proc. of AAAI-94, Seattle, WA*, pp. 1191–1198.

Sacerdoti, E. D. 1974 Planning in a hierarchy of abstraction spaces. *Artif. Intell.* **5**, 115–135.

Saitta, L. & Zucker, J.-D. 1998 Semantic abstraction for concept representation and learning. In *Working notes of the Symp. on Abstraction, Reformulation, and Approximation (SARA '98), Pacific Grove, CA*, pp. 103–120.

Shahar, Y. 1997 A framework for knowledge-based temporal abstraction. *Artif. Intell.* **90**, 79–133.

Shannon, C. 1950 A chess-playing machine. *Sci. Am.* **182**, 48–51.

Simon, H. A. 1981 *The sciences of the artificial*, 2nd edn. Cambridge, MA: MIT Press.

Struss, P. 1993 On temporal abstraction in qualitative reasoning (a preliminary report). In *Proc. 7th Int. Workshop on Qualitative Reasoning about Physical Systems, Orcas Island, WA*, pp. 219–227.

Weigel, R. & Faltings, B. 1997 Structuring techniques for constraint satisfaction problems. In *Proc. 15th IJCAI, Nagoya, Japan*, pp. 418–423.

Weigel, R., Faltings, B. & Choueiry, B. Y. 1996 Context in discrete constraint satisfaction problems. In *Proc. 12th Eur. Conf. on Artificial Intelligence, ECAI '96, Budapest, Hungary*, pp. 205–209.

Weld, D. S. & Addanki, S. 1990 Task-driven model abstraction. In *4th Int. Workshop on Qualitative Physics, Lugano, Switzerland*, pp. 16–30.

Zucker, J.-D. 2003 A grounded theory of abstraction in artificial intelligence. *Phil. Trans. R. Soc. Lond.* B **358**, 1293–1309. (DOI 10.1098/rstb.2003.1308.)

Zucker, J.-D., Bredèche, N. & Saitta, L. 2002 Abstracting visual percepts to learn concepts. In *Proc. 5th Int. Symp. on Abstraction, Reformulation and Approximation, SARA 2002, Kananaskis, Alberta, Canada, 2–4 August 2002. Lecture notes in artificial intelligence 2371* (ed. S. Koenig & R. Holte), pp. 256–273. Springer.

GLOSSARY

AI: artificial intelligence
CSP: constraint satisfaction problem