Abstraction and Reformulation in Artificial Intelligence

Robert C. Holte HOLTE@CS.UALBERTA.CA
Department of Computing Science,
University of Alberta,
Edmonton, Alberta, Canada, T6G 2E8

Berthe Y. Choueiry CHOUEIRY@CSE.UNL.EDU
Department of Computer Science and Engineering,
University of Nebraska-Lincoln,
Lincoln, NE, USA, 68588-0115

Abstract

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.

1. Introduction

In the early days of Artificial Intelligence (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 Artificial Intelligence 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 analyzing 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 unsolvability, of the modified problem directly gives the answer to the original problem, while 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

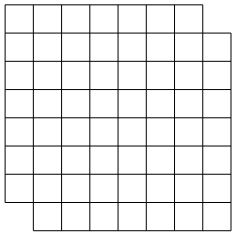

Figure 1: *8×8 mutilated array.*

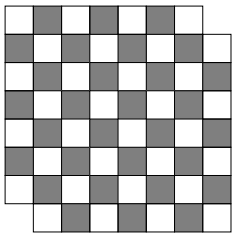

Figure 2: *Adding colour to the 8×8 mutilated array.*

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): unsolvability of the modified problem guarantees unsolvability of the original, but not vice versa. Just because humans solve the mutilated array problem in this way does not oblige

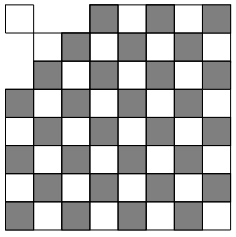

Figure 3: *8×8 array with B=W that cannot be tiled.*

computers to do so. But, 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 Artificial Intelligence *change of representation* or *reformulation*. Herb Simon argued that there was an intimate connection between reformulation and problem-solving (Simon, 1981):

> *"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."*

Abstraction is a kind of reformulation, but not all reformulations are abstractions. For example, the adding 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 there has not yet been given a precise, universal definition of abstraction. 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 which fall into the intuitive category of "abstractions" while at the same time being specific enough to exclude reformulations which 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 which might be thought to arise strictly in the realm of human reasoning arise in much the same form in the purely computational realm of Artificial Intelligence. While there are undoubtedly many purely psychological reasons underlying abstraction in human behaviour, there are certainly compelling computational reasons motivating the use of abstraction.

Secondly, this article contributes to the overall discussion of the nature of abstraction by providing examples of the abstraction processes that are 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.

This article does not give a comprehensive survey of work in Artificial Intelligence relevant to the subject of this special 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, Bredèche, & Saitta, 2002). The present article focuses on techniques that do not involve learning: they create abstractions by analyzing 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; Mallory, Porter, & Kuipers, 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 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 Artificial Intelligence 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 completely work out how to get from Paris to Sydney before at all considering the route to take

within Paris. The main implementations of this abstraction technique are reported in (Sacerdoti, 1974), (Knoblock, 1991), and (Holte, Mkadmi, Zimmer, & MacDonald, 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, since the function that estimates solution length is called the heuristic function. Heuristic search dates to the 1960s (Doran & Michie, 1966; Hart, Nilsson, & Raphael, 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 (Prieditis, 1993). In English it can be paraphrased roughly 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, since 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 hold between the lengths of their solutions. This is understandable, in the context of heuristic search, since 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). In order 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 so as 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, since 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 of the original 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, Codognet, & Rossi, 2000). Beyond this general use of abstraction, particular attention has been paid to modeling and the automatic detection and use of symmetry relations.

A familiar example of a Constraint Satisfaction Problem (CSP) is the 4-queens problem, where the task is to place 4 queens on a 4×4 chess board in such a way that no two queens attack each other. In general, a CSP is defined by three elements: (1) a set of decisions to be made, (2) a

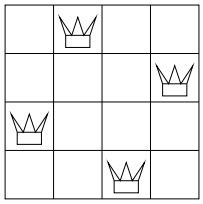

Figure 4: *A solution to the 4-queens problem.*

set of choices available for each decision, and (3) 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 *modeling* 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 Artificial Intelligence and also the closest to the research in Psychology and Cognitive Science.

CSPs are almost always modeled as follows. Each decision is represented by a variable. The a_d choices available for a particular decision d are represented by the integers $1...a_d$, called the "domain" of the variable. Thus, choosing alternative k for decision d is modeled as assigning value k to variable V_d. Each constraint is modeled 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 value 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 way of modeling the 4-queens problem.

There are other ways of modeling 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. Since 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 pigeon-hole problem where one has to place $(n + 1)$ objects into n pigeon-holes, such that each pigeon-hole 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 ways of assigning each object to each pigeon-hole. The 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 pigeon-holes – the unsolvability 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, Finkelstein, & Purdom, Jr., 1988; Puget, 1993; Backofen & Will, 1999; Gent & Smith, 2000; Hentenryck, 2002).

The pigeon-hole 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 (Freuder, 1991; Hubbe & Freuder, 1989; Haselböck, 1993; Ellman, 1993; Choueiry, Faltings, & Weigel, 1995; Freuder & Sabin, 1995; Weigel, Faltings, & Choueiry, 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, decisions that tightly interact can, under some conditions, be considered together, in isolation of 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 lumped 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 have been shown to be unsolvable too, as in the example of the mutilated array problem. On the other hand, if a solution is found for the tractable problem it can be used to guide 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). Binary constraints are preferable in general to non-binary ones because they are easier to handle and also 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.

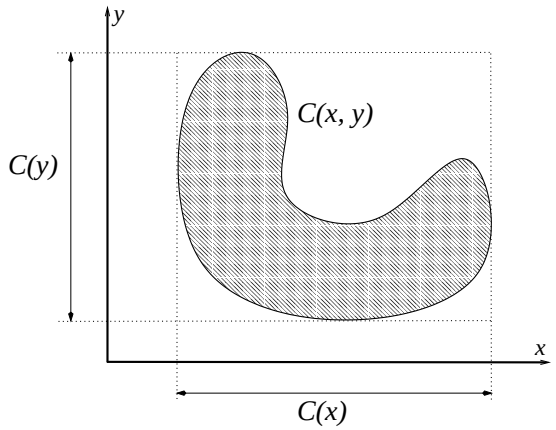

Figure 5: *Projection of 2-d constraints onto individual dimensions.*

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 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 6 variables) to create binary constraints (involving 2 variables).

4. Reasoning about Physical Systems

The goal of this area of Artificial Intelligence is to construct a model of the dynamic behavior of a physical system, and then process the model in order to predict or explain the behavior of the system under given conditions. For example, it aims at predicting the behavior of an electro-mechanical 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 system 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, Iwasaki, & McIlraith, 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 describe as an electrical conductor, which is in turn either an ideal-conductor or a resistor (Nayak & Joskowicz, 1996). The resistor can be modeled 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 behavior of the system that need to be explained or predicted. The final model must be detailed enough to capture correctly the behavior 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 modeling, into a global model that describes the behavior of the system (Falkenhainer & Forbus, 1991; Rickel & Porter, 1994; Levy, Iwasaki, & Fikes, 1997).

This global model can be a system of algebraic, differential equations or a set 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 done 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 Artificial Intelligence and Engineering: the representation is 'simplified' in order 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 the very 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

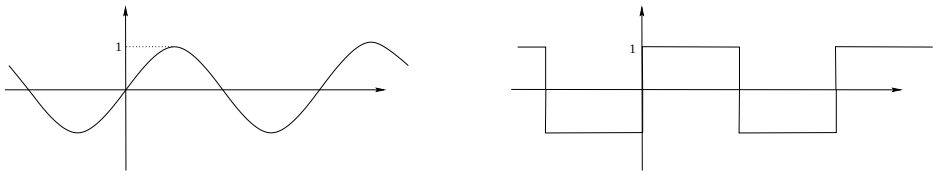

Figure 6: *Sine function and a square function.*

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, since 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 engineer quite often 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). While 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, Marriott, & Søndergaard, 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 Michael Lowry (Lowry, 1989). The impact of abstrac-

tion 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, Tenenberg, & Yang, 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 theories of abstraction in Artificial Intelligence can be found in this special issue in the article by Jean-Daniel Zucker (Zucker, 2003).

6. Conclusions

This paper has surveyed the most common notions of abstraction currently in use in three areas of Artificial Intelligence. 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 speedup 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 get built. (Choueiry & Davis, 2002) is an example of this type of system. Finally, there are systems which 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 depend 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 create a good problem formulation, although it is hoped that by automating abstraction and other reformulation techniques the burden on the human has been reduced.

7. Acknowledgments

The first author would like to thank the Natural Sciences and Engineering Research Council of Canada for the financial support that made this research possible. The second author is supported by a CAREER award #0133568 from the National Science Foundation.

References

Bacchus, F., & Yang, Q. (1994). Downward Refinement and the Efficiency of Hierarchical Problem Solving. *Artificial Intelligence*, *71*(1), 43–100.

Backofen, R., & Will, S. (1999). Excluding Symmetries in Constraint-Based Search. In *Principles and Practice of Constraint Programming, CP'99. Lecture Notes in Artificial Intelligence 1713*, pp. 73–87. Springer Verlag.

Bistarelli, S., Codognet, P., & Rossi, F. (2000). An Abstraction Framework for Soft Constraints and Its Relationship with Constraint Propagation. In Choueiry, B. Y., & Walsh, T. (Eds.), *4th*

International Symposium on Abstraction, Reformulation and Approximation (SARA 2000), Vol. 1864 of *Lecture Notes in Artificial Intelligence*, pp. 71–86. Springer Verlag.

Brown, C. A., Finkelstein, L., & Purdom, Jr., P. W. (1988). Backtrack Searching in the Presence of Symmetry. In Mora, T. (Ed.), *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pp. 99–110. Springer-Verlag.

Choueiry, B. Y., & Davis, A. M. (2002). Dynamic Bundling: Less Effort for More Solutions. In Koenig, S., & Holte, R. (Eds.), *5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, Vol. 2371 of *Lecture Notes in Artificial Intelligence*, pp. 64–82. Springer Verlag.

Choueiry, B. Y., Faltings, B., & Weigel, R. (1995). Abstraction by Interchangeability in Resource Allocation. In *Proc. of the 14 th IJCAI*, pp. 1694–1701, Montréal, Québec, Canada.

Choueiry, B. Y., Iwasaki, Y., & McIlraith, S. (2003). Towards a Practical Theory for Reformulation for Reasoning About Physical Systems. *Artificial Intelligence*. To appear.

Choueiry, B. Y., & Noubir, G. (1998). On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. In *Proc. of AAAI-98*, pp. 326–333, Madison, Wisconsin. Revised version KSL-98-24, `ksl-web.stanford.edu/KSL_Abstracts/KSL-98-24.html`.

Choueiry, B. Y., & Walsh, T. (Eds.). (2000). *Abstraction, Reformulation, and Approximation, proceedings of the 4th International Symposium, SARA 2000. Lecture Notes in Artificial Intelligence 1864*. Springer.

Cremonini, R., Marriott, K., & Søndergaard, H. (1990). A General Theory of Abstraction. In *Proceedings of the 4th Australian Joint Conference on Artificial Intelligence*, pp. 121–134, Australia.

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. *Advances in Artificial Intelligence (Lecture Notes in Artificial Intelligence 1081)*, 402–416.

Culberson, J. C., & Schaeffer, J. (1998). Pattern Databases. *Computational Intelligence*, *14* (3), 318–334.

Davis, E. (1995). Approximation and Abstraction in Solid Object Kinematics. Technical report TR706, New York University, New York, NY.

Doran, J. E., & Michie, D. (1966). Experiments with the Graph Traverser Program. *Proceedings of the Royal Society of London, Series A*, *294*, 235–259.

Drummond, C. (1999). *A Symbol's Role In Learning Low Level Control Functions*. Ph.D. thesis, Computer Science Department, University of Ottawa, Canada.

Drummond, C. (2002). Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks. *Journal of Artificial Intelligence Research*, *16*, 59–104.

Edelkamp, S. (2001). Planning with Pattern Databases. *Proceedings of the 6th European Conference on Planning (ECP-01)*.

Ellman, T. (1993). Abstraction via Approximate Symmetry. In *Proc. of the 13 th IJCAI*, pp. 916–921, Chambéry, France.

Falkenhainer, B., & Forbus, K. D. (1991). Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, *51*, 95–143.

Fillmore, J. P., & Williamson, S. (1974). On Backtracking: A Combinatorial Description of the Algorithm. *SIAM Journal on Computing*, *3 (1)*, 41–55.

Freuder, E. C. (1991). Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proc. of AAAI-91*, pp. 227–233, Anaheim, CA.

Freuder, E. C., & Sabin, D. (1995). Interchangeability Supports Abstraction and Reformulation for Constraint Satisfaction. In *Symposium on Abstraction, Reformulation and Approximation, SARA'95*, Ville d'Esterel, Canada.

Gasching, J. (1979). A Problem Similarity Approach to Devising Heuristics: First Results. *IJCAI*, 301–307.

Gent, I. P., & Smith, B. M. (2000). On Reformulation of Constraint Satisfaction Problems. In *Proc. of the 14 th ECAI*, pp. 599–603, Berlin, Germany.

Giunchiglia, F., & Walsh, T. (1992). A Theory of Abstraction. *Artificial Intelligence*, *57*, 323–389.

Glaisher, J. (1874). On the Problem of the Eight Queens. *Philosophical Magazine, series 4*, *48*, 457–467.

Gruber, T. R., & Gautier, P. O. (1993). Machine-generated Explanations of Engineering Models: a Compositional Modeling Approach. In *Proc. of the 13 th IJCAI*, pp. 1502–1508, Chambéry, France.

Guida, G., & Somalvico, M. (1979). A Method for computing Heuristics in problem solving. *Information Sciences*, *19*, 251–259.

Hart, P., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*, 100–107.

Haselböck, A. (1993). Exploiting Interchangeabilities in Constraint Satisfaction Problems. In *Proc. of the 13 th IJCAI*, pp. 282–287, Chambéry, France.

Hentenryck, P. V. (Ed.). (2002). *Proceedings of* 8^{th} *International Conference on Principle and Practice of Constraint Programming (CP'02)*, Vol. 2470 of *Lecture Notes in Computer Science*. Springer Verlag, Ithaca, NY.

Hernádvölgyi, I. T. (2001). Searching for Macro Operators with Automatically Generated Heuristics. *Advances in Artificial Intelligence - Proceedings of the Fourteenth Biennial Conference of the Canadian Society for Computational Studies of Intelligence (LNAI 2056)*, 194–203.

Holte, R. C., Mkadmi, T., Zimmer, R. M., & MacDonald, A. J. (1996). Speeding Up Problem-Solving by Abstraction: A Graph-Oriented Approach. *Artificial Intelligence*, *85*, 321–361.

Huang, X. (1994). Reconstructing Proofs at the Assertion Level. In Bundy, A. (Ed.), *Proc. 12th Conference on Automated Deduction*, pp. 738–752. Springer-Verlag.

Hubbe, P. D., & Freuder, E. C. (1989). An Efficient Cross Product Representation of the Constraint Satisfaction Problem Search Space. In *Proc. of AAAI-92*, pp. 421–427, San Jose, CA.

Knoblock, C. A. (1991). Automatically Generating Abstractions for Planning. *Artificial Intelligence*, *68*(2), 243–302.

Knoblock, C. A., Tenenberg, J. D., & Yang, Q. (1991). Characterizing Abstraction Hierarchies for Planning. In *Proc. of AAAI-91*, pp. 692–697, Anaheim, CA.

Koenig, S., & Holte, R. C. (Eds.). (2002). *Abstraction, Reformulation, and Approximation, proceedings of the 5th International Symposium, SARA 2002. Lecture Notes in Artificial Intelligence 2371.* Springer.

Korf, R. E. (1997). Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 700–705.

Korf, R. E. (1980). Toward a Model of Representation Changes. *Artificial Intelligence*, *14*, 41–78.

Levy, A. Y., Iwasaki, Y., & Fikes, R. (1997). Automated Model Selection for Simulation Based on Relevance Reasoning. *Artificial Intelligence*, *96*, 351–394.

Lowry, M. R. (1989). *Algorithm Synthesis through Problem Reformulation.* Ph.D. thesis, Computer Science Department, Stanford University.

Mallory, R. S., Porter, B. W., & Kuipers, B. J. (1996). Comprehending Complex Behavior Graphs through Abstractions. In *Tenth International Workshop on Qualitative Physics. AAAI Technical Report WS-96-01*, pp. 137–146, Fallen Leaf Lake, CA.

McCarthy, J. (1964). A Tough Nut for Proof Procedures. Stanford Artificial Intelligence Project Memo 16.

Minsky, M. (1961). Steps Toward Artificial Intelligence. *Proc. IRE, 49*(1), 8–30.

Mostow, J., & Prieditis, A. (1989). Discovering Admissible Heuristics by Abstracting and Optimizing: A Transformational Approach. *IJCAI*, 701–707.

Mozetič, I. (1991). Hierarchical Model-Based Diagnosis. *International Journal of Man-Machine Studies, 35*, 329–362.

Muggleton, S. (1988). A Strategy for Constructing New Predicates in First Order Logic. In Sleeman, D. (Ed.), *Proceedings of the 3rd European Working Session on Learning*, pp. 123–130. Pitman.

Nadel, B. (1990). Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens. *IEEE Expert, 5 (3)*, 16–24.

Nayak, P. P., & Joskowicz, L. (1996). Efficient Compositional Modeling for Generating Causal Explanations. *Artificial Intelligence, 83*, 193–227.

Nayak, P. P., & Levy, A. Y. (1995). A Semantic Theory of Abstractions. In *Proc. of the 14 th IJCAI*, pp. 196–203, Montréal, Québec, Canada.

Newell, A. (1965). Limitations of the Current Stock of Ideas about Problem Solving. In Kent, A., & Taulbee, O. E. (Eds.), *Electronic Information Handling*, pp. 195–208. Spartan Books.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison & Wesley.

Plaisted, D. A. (1981). Theorem Proving with Abstraction. *Artificial Intelligence, 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 Learning, 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*, pp. 1191–1198, Seattle, WA.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5*(2), 115–135.

Saitta, L., & Zucker, J.-D. (1998). Semantic Abstraction for Concept Representation and Learning. In *Working notes of the Symposium on Abstraction, Reformulation, and Approximation (SARA'98)*, pp. 103–120, Pacific Grove, CA.

Shahar, Y. (1997). A Framework for Knowledge-Based Temporal Abstraction. *Artificial Intelligence, 90 (1-2)*, 79–133.

Shannon, C. (1950). A Chess-playing Machine. *Scientific American, 182*(2), 48–51.

Simon, H. A. (1981). *The Sciences of the Artificial.* MIT Press, Cambridge, MA.

Struss, P. (1993). On Temporal Abstraction in Qualitative Reasoning (A Preliminary report). In *Proceedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems*, pp. 219–227, Orcas Island, Wa.

Weigel, R., & Faltings, B. (1997). Structuring Techniques for Constraint Satisfaction Problems. In *Proc. of the 15 th IJCAI*, pp. 418–423, Nagoya, Japan.

Weigel, R., Faltings, B., & Choueiry, B. Y. (1996). Context in Discrete Constraint Satisfaction Problems. In *12th European Conference on Artificial Intelligence, ECAI'96*, pp. 205–209, Budapest, Hungary.

Weld, D. S., & Addanki, S. (1990). Task-Driven Model Abstraction. In *4th International Workshop on Qualitative Physics*, pp. 16–30, Lugano, Switzerland.

Zucker, J.-D. (2003). A Grounded Theory of Abstraction in Artificial Intelligence. *Philosophical Transactions of the Royal Society (Biological Sciences)*, *This issue.*

Zucker, J.-D., Bredèche, N., & Saitta, L. (2002). Abstracting Visual Percepts to Learn Concepts. In Koenig, S., & Holte, R. (Eds.), *5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, Vol. 2371 of *Lecture Notes in Artificial Intelligence*, pp. 256–273. Springer Verlag.