Evaluation of Representations in AI Problem Solving

Eugene Fink
Computer Science, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
e.fink@cs.cmu.edu, www.cs.cmu.edu/∼eugene

Abstract – *We formalize the concept of domain representation in artificial intelligence, and propose a framework for evaluating representation and comparing alternative representations.*

Keywords: Representation, utility, search, artificial intelligence.

1 Introduction

The performance of all reasoning systems crucially depends on problem representation; the same task may be easy or difficult, depending on the way we describe it. Researchers in psychology and artificial intelligence have accumulated much evidence on the importance of appropriate representations for both humans and artificial-intelligence systems; however, the notion of "good" representations has remained at an informal level.

We propose a formal definition of representations, and a framework for their quantitative evaluation. The development of this framework is part of a project focused on automatic improvement of representations [4, 5, 6]. We review several approaches to defining representation (Section 2) and give a formal definition based on Simon's view of representation as "data structures and programs operating on them" (Section 3). We then define a utility of problem solving (Section 4), and propose a measure of representation quality based on this utility (Section 5).

2 Alternative definitions

Informally, a problem representation is a certain view of a problem and an approach to solving it. Although researchers agree in their intuitive understanding of this concept, they have not yet developed its standard formalization. We outline several views of representation, summarized in Figure 1.

Representation...

- includes a machine language for the description of reasoning tasks and a specific encoding of a given problem in this language (Amarel [3]).
- is the space expanded by a solver algorithm during its search for a solution (Newell and Simon [12]).
- is the state space of a given problem, formed by all legal states of the simulated world and transitions between them (Korf [9]).
- "consists of both data structures and programs operating on them to make new inferences" (Simon [10]).
- determines a mapping from the behavior of an artificial-intelligence system on a certain set of inputs to the behavior of another system, which performs the same task on a similar set of inputs (Holte [7]).

Figure 1: Definitions of representation in artificial intelligence; note that these definitions are not equivalent.

Problem formulation: Amarel was first to point out the impact of representation on the efficiency of search [1, 2, 3]. He considered some problems of reasoning about actions in a simulated world, and discussed their alternative formulations in the input language of a search algorithm. He showed that a specific formulation of a problem determines its *state space,* that is, the space of possible states of the simulated world and transitions between them, and that the efficiency of problem solving depends on the size of this space.

Van Baalen adopted a similar view in his work on a theory of representation design [14]. He defined representation as a mapping from concepts to their syntactic description in a formal language, and implemented a program that automatically improved descriptions of simple reasoning tasks.

Problem space: Newell and Simon investigated the role of representation in human problem solving [12]. They observed that a human subject always encodes a given task in a *problem space,* that is, "some kind of space that represents the initial situation presented to him, the desired goal situation, various intermediate states, imagined or experienced, as well as any concepts he uses to describe these situations to himself" [12].

They defined representation as the subject's problem space, which determines partial solutions considered during search for a complete solution. This definition is also applicable to artificial-intelligence systems since most systems use the same principle of searching among partial solutions.

Observe that the problem space may differ from the state space of the simulated world. The subject may disregard some of the allowed transitions and, on the other hand, consider impossible world states. For instance, when people work on difficult versions of the Tower-of-Hanoi puzzle, they sometimes attempt illegal moves [13]. Moreover, the problem solver may abstract from the search among world states and use an alternative view of partial solutions.

State space: Korf described a formal framework for automated improvement of representations [9]. He developed a language for describing search problems, and defined representation as a specific encoding of a problem in this language. The encoding included the initial state of the simulated world and operations for transforming the state; hence, it defined the state space.

Korf pointed out the correspondence between the problem encoding and state space, which allowed him to view representation as a space of states and transitions between them. Korf's notion of representation did not include the behavior of a search algorithm. Since the performance depended not only on the state space but also on the search strategies, representation in his framework did not uniquely determine the efficiency.

Data and programs: Simon suggested a general definition of representation as "data structures and programs operating on them" [10]. When describing the behavior of human subjects, he viewed their initial encoding of a given problem as a "data structure," and the available productions for modifying it as "programs." Since the problem encoding and rules for changing it determined the subject's search space, this view was similar to the earlier definition by Newell and Simon [12].

If we apply Simon's definition in other research contexts, the notions of data structures and programs may take different meanings. The general concept of "data structures" encompasses any form of a system's input and internal representation of related information. Similarly, the term "programs" may refer to any strategies and procedures for processing a given problem.

System's behavior: Holte developed a framework for the analysis and comparison of learning systems, which included rigorous mathematical definitions of task domains and their representations [7]. He considered representations of domains rather than specific problems, which distinguished his view from the earlier definitions.

A domain in Holte's framework includes a set of elementary entities, a collection of primitive functions that describe the relationships among entities, and legal compositions of primitive functions. For example, we may view the world states as elementary objects and transitions between them as primitive functions. A domain specification may include not only a description of reasoning tasks, but also a system's behavior on them.

Representation is a mapping between two domains that encode the same reasoning tasks. It may describe a system's behavior on two different encodings of a problem. Alternatively, it may show the correspondence between the behavior of two different systems that perform the same task.

A **problem solver** is an algorithm that performs some type of reasoning tasks. When we invoke this algorithm, it inputs a given problem and searches for a solution; it may solve the problem or report a failure.

A **problem description** is an input to a solver. In most search systems, it includes allowed operations, initial world state, logical statement describing goals, and possibly heuristics for guiding the search.

A **domain description** is the part of a problem description that is common for a certain class of problems. It usually does not include a specific initial state or goals.

A **representation** is a domain description with a problem solver that uses this description. A representation change may involve improving a description, selecting a new solver, or both.

Figure 2: Definitions of a problem solver, domain description, and representation.

3 Domain representation

We follow Simon's view of representation as "data structures and programs operating on them"; however, the notion of data structures and programs in the proposed framework differs from his definition in the research on human problem solving. We summarize the related terminology in Figure 2.

A *problem solver* is an algorithm that inputs a problem, domain description, and time bound, and then searches for a solution. It terminates when it finds a solution, exhausts its search space, or hits the time bound.

A *problem description* is an input to a solver, which encodes a certain reasoning task. This notion is analogous to Amarel's "problem formulation," which is part of his definition of representation.

A problem description in artificial intelligence usually consists of two main parts, a *domain description* and *problem instance.* The first part includes the properties of a simulated world, which is called the *problem domain.* For example, the domain of the Tower-of-Hanoi puzzle may specify the legal states and moves. The second part encodes a particular reasoning task, which may include an initial state of the simulated world and a goal specification. For example, a problem instance in the Tower-of-Hanoi domain may include the initial positions of all disks and their desired final positions. Thus, a domain description is the part of a problem description that is common for all problems in the domain.

A *representation* consists of a domain description and a solver algorithm that operates on this description. If the algorithm does not make random choices, the representation uniquely defines the search space for every problem, which relates this definition to Newell and Simon's view of representation as a search space.

4 Gain function

We describe a framework for evaluating representations, which accounts for the number of solved problems, running time, and solution quality.

We assume that the application of a problem solver leads to one of three outcomes: finding a solution, terminating with failure after exhausting the search space, or hitting a time bound. Note that a failure may sometimes be more valuable than a time-bound interrupt; for example, if the search algorithm is complete, then a failure indicates that the given problem has no solution, which may be useful information.

We pay for running time and get a reward for solving a problem; the reward may depend on a specific problem and its solution. We may also get a reward for finding out that the search space has no solution, but we never get a reward for a time-bound termination. The overall *gain* is a function of a problem, time, and search result. We denote it by $gn(prob, time, result)$, where *prob* is the problem, *time* is the running time, and *result* may be a solution or one of two unsuccessful outcomes: a failure to solve a problem (denoted fail) or a time-bound interrupt (denoted intr). Note that fail and intr are not variables; hence, we do not italicize them.

Basic constraints: A user has to provide a specific gain function, thus encoding value judgments about different outcomes. We impose three constraints on the allowed gain functions.

1. The gain decreases with the time:
For every *prob*, *result*, and $time_1 < time_2$,
$gn(prob, time_1, result) \geq gn(prob, time_2, result)$.

2. A zero-time interrupt gives zero gain:
For every *prob*,
$gn(prob, 0, \mathsf{intr}) = 0$.

3. The gain of solving a problem or exhausting the search space is no smaller than the interrupt gain:
For every *prob*, *time*, and *soln*,
(a) $gn(prob, time, soln) \geq gn(prob, time, \mathsf{intr})$,
(b) $gn(prob, time, \mathsf{fail}) \geq gn(prob, time, \mathsf{intr})$.

Observe that the gain of doing nothing is always zero (Constraint 2). Furthermore, Constraints 1 and 2 imply that an interrupt without finding a solution never gives a positive gain, whereas Constraints 2 and 3 imply that the gain of instantly finding a solution is nonnegative.

We define a relative *quality* of solutions through a gain function. Suppose that $soln_1$ and $soln_2$ are two solutions for a problem *prob*.

$soln_1$ has higher quality than $soln_2$ if, for every *time*, $gn(prob, time, soln_1) \geq gn(prob, time, soln_2)$.

If $soln_1$ gives larger gains than $soln_2$ for some running times and lower gains for others, then neither of them has higher quality than the other.

Additional constraints: We now discuss three additional constraints, which describe typical special cases. We first observe that practical gain functions usually satisfy the following condition.

4. If $soln_1$ gives a larger gain than $soln_2$ for zero time, it gives larger gains for all other times:
For every *prob*, *time*, $soln_1$, and $soln_2$,
if $gn(prob, 0, soln_1) \geq gn(prob, 0, soln_2)$,
then $gn(prob, time, soln_1) \geq gn(prob, time, soln_2)$.

If gain satisfies this constraint, we can compare the quality of any two solutions; that is, for every problem, its solutions are totally ordered by quality. We can then define a quality function, $quality(prob, result)$, that satisfies the following conditions for every problem and every two results:

- $quality(prob, \mathsf{intr}) = 0$.
- If $gn(prob, 0, result_1) = gn(prob, 0, result_2)$, then $quality(prob, result_1) = quality(prob, result_2)$.
- If $gn(prob, 0, result_1) > gn(prob, 0, result_2)$, then $quality(prob, result_1) > quality(prob, result_2)$.

Note that most domains have natural quality measures that satisfy these conditions. For example, we may define the quality of a solution for the Tower-of-Hanoi puzzle as the number of moves.

We may view gain as a function of a problem, time, and solution quality, denoted $gn_q(prob, time, quality)$, which satisfies the following condition:

For every *prob*, *time*, and *result*,
$gn_q(prob, time, quality(prob, result))$
$= gn(prob, time, result)$.

Note that gain is an increasing function of quality:

If $quality_1 \leq quality_2$, then
$gn_q(prob, time, quality_1) \leq gn_q(prob, time, quality_2)$.

We next consider two other special-case constraints:

5. We can decompose the gain into the payment for time and the reward for solving a problem:

For every *prob*, *time*, and *result*,
$gn(prob, time, result)$
$= gn(prob, time, \mathsf{intr}) + gn(prob, 0, result)$.

6. The sum payment for two interrupts that take $time_1$ and $time_2$ is the same as the payment for an interrupt that takes $time_1 + time_2$:

For every *prob*, $time_1$, and $time_2$,
$gn(prob, time_1, \mathsf{intr}) + gn(prob, time_2, \mathsf{intr})$

$$= gn(prob, time_1 + time_2, \mathsf{intr}).$$

If gain satisfies Constraint 5, then it also satisfies Constraint 4; furthermore, if we define quality as $quality(prob, result) = gn(prob, 0, result)$, then gn_q is a linear function of quality:

$$gn_q(prob, time, quality) = gn(prob, time, \mathsf{intr}) + quality.$$

If gain satisfies Constraint 6, the interrupt gain is proportional to time:

$$gn(prob, time, \mathsf{intr}) = time \cdot gn(prob, 1, \mathsf{intr}).$$

Constraints 5 and 6 together lead to the following decomposition of the gain function:

$$gn(prob, time, result) \\ = time \cdot gn(prob, 1, \mathsf{intr}) + gn(prob, 0, result).$$

5 Representation quality

We derive a utility function for evaluating representations, and then extend this result to account for the use of time bounds and multiple representations.

Utility function: We assume that solver algorithms never make random choices; then, for every problem *prob*, representation uniquely determines the running time, $time(prob)$, and the result, $result(prob)$. Therefore, it also uniquely determines the respective gain, $gn(prob, time(prob), result(prob))$. We define a representation utility by averaging the gain over all problems [8]. We denote the set of problems by $\mathcal{P}$, assume a fixed probability distribution on $\mathcal{P}$, and denote the probability of encountering *prob* by $p(prob)$. If we select a problem at random, the expected gain is as follows:

$$G = \sum_{prob \in \mathcal{P}} p(prob) \cdot gn(prob, time(prob), result(prob)).$$

We use G as a utility function for evaluating representation, which unifies the three utility dimensions: number of solved problems, speed, and solution quality.

Time bounds: If we never interrupt a solver, its search time may be infinite. In practice, we eventually have to stop the search, which means that we always set some bound on the allowed search time. If we use a bound B, the search time and result are as follows:

$$\begin{aligned} time' &= \min(B, time(prob)) \\ result' &= \begin{cases} result(prob), & \text{if } B \geq time(prob) \\ \mathsf{intr}, & \text{if } B < time(prob) \end{cases} \end{aligned}$$

Thus, the choice of a bound may affect the search time and result, which implies that it affects the gain. We denote the function that maps problems and bounds into gains by gn':

$$gn'(prob, B) = gn(prob, time', result').$$

If gain satisfies Constraint 4, and we use a quality measure $quality(prob, result)$, then we can define gn' in terms of the solution quality. If we use a time bound B, the solution quality is as follows:

$$quality'(prob, B) = \begin{cases} quality(prob, result(prob)), \\ \quad \text{if } B \geq time(prob) \\ quality(prob, \mathsf{intr}), \\ \quad \text{if } B < time(prob) \end{cases}$$

We then express gn' through the function gn_q defined in Section 4:

$$gn'(prob, B) = gn_q(prob, time', quality'(prob, B)).$$

The choice of a time bound often depends on a specific problem; for example, users of artificial-intelligence systems usually set smaller bounds for smaller-scale problems. We have suggested some heuristics for choosing time bounds, along with learning algorithms for improving these heuristics, in an article on selection of problem-solving methods [6].

If we view the selected time bound as a function of a given problem, $B(prob)$, the expected gain for a randomly selected problem is as follows:

$$G = \sum_{prob \in \mathcal{P}} p(prob) \cdot gn'(prob, B(prob)).$$

Multiple representations: We next consider the use of multiple representations; that is, we analyze a system that includes a library of solver algorithms and related data structures, along with a manual or automated mechanism for selecting among them [6, 11, 15].

We denote the number of available representations by k, and consider the respective gain functions $gn_1, ..., gn_k$ and bound-selection functions $B_1, ..., B_k$. When solving a problem *prob* with representation i, we set the time bound $B_i(prob)$ and use gn_i to determine the gain. For every i, we define the function gn'_i in the same way as we have defined gn'; the gain of solving *prob* with representation i is $gn'_i(prob, B_i(prob))$.

For each given problem *prob*, either the user or an automated control module chooses an appropriate representation $i(prob)$, and then sets the respective bound $B_{i(prob)}(prob)$. If we select a problem at random, the expected gain is as follows:

$$G = \sum_{prob \in \mathcal{P}} p(prob) \cdot gn'_{i(prob)}(prob, B_{i(prob)}(prob)).$$

Thus, the utility G depends on the gain function, probability distribution, and procedure for selecting representations and time bounds.

6 Concluding remarks

We have proposed a definition of representation in artificial-intelligence problem solving, and a framework for evaluating representations. This framework shows that the relative quality of representations depends on the user's judgment of relative solution quality, probabilities of encountering different problems, and heuristics for selecting appropriate time bounds. Thus, it confirms the well-known observation that no representation is universally better than the others, and the choice of representation should depend on specific requirements and problem types.

We have applied the described framework to develop a mechanism for evaluation and selection of representations [6], which is part of a system for automated representation improvement [5].

The framework is based on several simplifying assumptions, and we plan to build a more general framework as part of the future work. In particular, we have assumed that all solver algorithms are *sound*; that is, they always produce correct solutions. Furthermore, we have assumed that solvers do not use *any-time* behavior; that is, a solver finds a solution and terminates rather than outputting successively better solutions. We have also assumed that solvers do not make random choices; however, we can readily extend the framework to randomized solvers by viewing every possible behavior as a separate problem-solving episode.

References

[1] Saul Amarel. An approach to automatic theory formation. In Heinz M. Von Foerster, editor, *Principles of Self-Organization: Transactions*. Pergamon Press, New York, NY, 1961.

[2] Saul Amarel. Problem solving procedures for efficient syntactic analysis. In *Proceedings of the* ACM *Twentieth National Conference*, 1965.

[3] Saul Amarel. On representations of problems of reasoning about actions. In Donald Michie, editor, *Machine Intelligence 3*, pages 131–171. American Elsevier Publishers, New York, NY, 1968.

[4] Eugene Fink. Systematic approach to the design of representation-changing algorithms. In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation*, pages 54–61, 1995.

[5] Eugene Fink. *Changes of Problem Representation: Theory and Experiments*. Springer, Berlin, Germany, 2003.

[6] Eugene Fink. Automatic evaluation and selection of problem-solving methods: Theory and experiments. *Journal of Experimental and Theoretical Artificial Intelligence*, 16(2):73–105, 2004.

[7] Robert C. Holte. *An Analytical Framework for Learning Systems*. PhD thesis, Artificial Intelligence Laboratory, University of Texas at Austin, 1988. Technical Report AI-88-72.

[8] Sven Koenig. *Goal-Directed Acting with Incomplete Information*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1997. Technical Report CMU-CS-97-199.

[9] Richard E. Korf. Toward a model of representation changes. *Artificial Intelligence*, 14:41–78, 1980.

[10] Jill H. Larkin and Herbert A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65–99, 1987.

[11] Steven Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints: An International Journal*, 1(1–2):7–43, 1996.

[12] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice Hall, Upper Saddle River, NJ, 1972.

[13] Herbert A. Simon, Kenneth Kotovsky, and John R. Hayes. Why are some problems hard? Evidence from the Tower of Hanoi. *Cognitive Psychology*, 17:248–294, 1985.

[14] Jeffrey Van Baalen. *Toward a Theory of Representation Design*. PhD thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1989. Technical Report 1128.

[15] David E. Wilkins and Karen L. Myers. A multiagent planning architecture. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 154–162, 1998.