Case-by-Case Problem Solving

Pei Wang
Temple University, Philadelphia, USA
http://www.cis.temple.edu/∼pwang/

Abstract

Case-by-case Problem Solving solves each occurrence, or case, of a problem using available knowledge and resources on the case. It is different from the traditional Algorithmic Problem Solving, which applies the same algorithm to all occurrences of all problem instances. Case-by-case Problem Solving is suitable for situations where the system has no applicable algorithm for a problem. This approach gives the system flexibility, originality, and scalability, at the cost of predictability. This paper introduces the basic notion of Case-by-case Problem Solving, as well as its most recent implementation in NARS, an AGI project.

Algorithmic Problem Solving

"Problem Solving" is the process to find a *solution* for a given *problem* by executing some *operations*. For a certain system at a certain moment, the set of executable operations usually remains constant. Therefore, the task for the system is to find a way to select proper operations and to execute them in proper order for the given problem.

In computer science and AI, the dominant approach in problem solving can be called "Algorithmic Problem Solving" (APS in the following). According to this approach, first a *problem* is specified as a function that maps any input (problem instance) of a certain type to the corresponding output. Then, an *algorithm* is designed, which accomplishes this function step by step, where each step is a well-defined operation. Finally, the algorithm is implemented in a programming language to become a computer program, which will be able to let a computer routinely transform valid input data into output data. A well-known description of this approach can be found in (Marr, 1982).

Accurately speaking, in this approach "problem solving" happens in two different levels:

1. When the *problem* refers to a *problem type*, or input-output mapping, the *solution* is the corresponding algorithm (conceptually speaking) or program (practically speaking) that accomplishes the mapping. For example, when the problem is "to sort sequences of comparable items", one solution is "quicksort".
2. When the *problem* refers to a *problem instance*, or input data, then the *solution* is the corresponding output data, according to the problem specification. For example, when the problem is "to sort [3, 2, 4, 1]", the solution is "[1, 2, 3, 4]".

So APS has two phases: at first a human solves a problem by designing an algorithm for it, then a computer applies the algorithm to solve concrete instances of the problem.

Computer science inherited APS from mathematics, and has successfully applied and enhanced it to provide a theoretical and methodological foundation for the information technology. Even so, this approach has its limitation:

- For some problems, no algorithm has been found. Even worse, for some problems it can be proved that no algorithm can be found. This is the issue of *computability* (Davis, 1958).
- For some problems, all known algorithms require too much time-space resources to solve every instances of the problem in practical situations. This is the issue of *computational complexity* (Cormen et al., 2001).

Beside the above issues that are well-known to computer science, AI has taken the additional challenge of building computer systems that require little human involvement in the whole problem solving process. To be intelligent, a computer system should have creativity and flexibility, which often means to be able to solve a problem for which it has not been given an applicable algorithm.

Some people consider this task as impossible: if everything a computer does follow some algorithm, how can it solve a problem for which no algorithm is given in advance? This opinion comes from a misconception, because a computer may be able to solve a problem without a predetermined algorithm *for that problem*, while in the whole process the system still follow algorithms defined on other problems, not the one under consideration (Wang, 2007).

Obviously, when a computer system must solve problems for which no algorithm is given in advance, then it can no longer follow the APS approach. In computer science and AI, many alternative approaches have been explored. This paper will not provide a comprehensive survey on this topic. Instead, it will concentrate on one approach, "Case-by-case Problem Solving", describe its up-to-date implementation in an AGI system, and compare it with some of the alternatives.

CPS: the Basic Idea

Case-by-case Problem Solving (CPS) is a notion introduced in contrast with Algorithmic Problem Solving (APS). This notion was formed during the development of NARS, an AGI project, and the basic idea has been described in previous publications (Wang, 1996; Wang, 2004), though not bearing this name. Here the notion is briefly summarized and explained.

NARS is an intelligent system designed according to the theory that "intelligence" means "adaptation and working with insufficient knowledge and resources". Descriptions of the whole project can be found in (Wang, 1995; Wang, 2006), and this paper only focuses on a certain aspect of the system.

NARS accepts three types of task from the environment: *knowledge* to be absorbed, *questions* to be answered, and *goals* to be achieved. Each piece of new knowledge is turned into a belief of the system, and is used in forward inference to derive or revise other beliefs; Each new question and goal, which is what we usually call a "problem", is matched with existing beliefs for possible direct solutions, as well as used in backward inference to produce derived questions and goals, based on relevant beliefs.

One concrete implication of the above theory of intelligence is that an intelligent system, like NARS, often needs to deal with problems for which the system has no applicable algorithm, as a special case of "insufficient knowledge". As analyzed in the previous section, this can be caused by various reasons, such as:

- The problem is not computable;
- Though the problem may be computable, no algorithm has been found yet;
- Though the problem can be solved by an algorithm, it is unknown to the system at the moment;
- Though the system knows some algorithmic solutions to the problem, it cannot afford the resource required by any of them.

No matter what the reason is, in this situation the system cannot follow APS. To work in this situation, there are two possible approaches:

1. Find an algorithm first, then use it to process the problem instances;
2. Directly solve the problem instances without following a predetermined algorithm.

While most of the relevant works in AI follow the first approach, in NARS the second approach is explored. Here a key observation is that the "problem" an intelligent system meets is usually a "problem instance", rather than a "problem type". The "sorting problem" ordinary people meet in their daily life is usually to sort concrete sequences, one at a time, not "to find a method to routinely sort any sequence", as defined by mathematicians and computer scientists. Therefore, even when a problem type cannot be "solved" by an algorithm, some (even if not all) of its instances may still be solved, by taking the special properties of each of them into consideration. In this way, "problem solving" is carried out in a *case by case* manner, and that is where the name CPS comes.

Some people may suspect CPS as APS rebranded, by treating what is previously taken as a *problem instance* as a *problem type* — though the system has no algorithm to sort all sequences, it might have an algorithm to sort [3, 2, 4, 1]. This is not what CPS means, because in a system like NARS, not only that each *instance* of the same problem type may be processed differently, but also that each *occurrence* of the same problem instance may be processed differently. This is not as strange as it sounds if we consider human problem solving, where the same problem (instance) often gets different treatment when it occurs in different contexts.

How about to insist that "The system is still following an algorithm for each *occurrence* of the problem, though different occurrences of the same problem may be handled by different algorithms"? After all, the actual solving process of the problem (occurrence) consists of nothing but a sequence of operations, right? Isn't it just an algorithm? Such a usage of the notion of "algorithm", though possible, would make it useless in analyzing the system, because such an "algorithm" can only be recorded *after* the problem-solving process, and is not repeatable. No matter what word is used, the system's processing of the next occurrence of the problem is no longer accurately predictable, unless everything in the environment and the system are fully specified. In this situation the system does not serve as a fixed function mapping the problem instances to corresponding solutions.

The above analysis suggests that non-algorithmic CPS is not only logically possible, but also has the human mind as an existing proof. However, it does not tell us *how* to carry out this kind of process in a computer. After all, a computer system has to follow some algorithms (though not specific to the domain problems) to control its activities.

Theoretically speaking, if the precondition and consequence of each operation are accurately known, the system should be able to solve a concrete problem by exhaustively evaluating all possible operation sequences, and choosing the best solution according to their overall results. However, it is obvious that for any non-trivial problem such an exhaustive search will not be affordable. This is especially true for NARS, with its assumption on the insufficiency of knowledge and resources — "insufficient knowledge" means that the precondition and consequence of each operation are not accurately known, and "insufficient resources" means that the system does not have the time and space to consider all known possibilities. Under this assumption, by definition, the system cannot always find the best solution that guarantees the optimal result among all alternatives.

On the other hand, for a system working in this situation, the "insufficiency" assumption does not mean that all solutions are equally good. According to the opinion that intelligence is a form of adaptation, an intelligent system should pick the best solution that, according to its experience, is most likely to achieve a desired result, among the alternatives the system can consider with available resources.

As a realization of the above idea, the problem-solving process in NARS can be informally and briefly described as the following.

First, since NARS is designed in the framework of reasoning system, in it goals, operations, and beliefs are all represented as sentences in a formal language. A *goal* describes what the system want to achieve (i.e., to make it true); an *operation* can be directly achieved by executing some code; and a *belief* summarizes the system's experience on the relations among items in the system, including goals, operations, and other beliefs.

Assuming insufficient knowledge, in NARS a belief can only specify partial preconditions or consequences of an operation, with a truth-value to indicate the evidential support for it according to the system's experience. Each inference rule, when used for *forward* inference, takes a couple of existing beliefs as premise, and derives a conclusion, with a truth-value determined according to the evidence provided by the premises. With the coming of new evidence, new beliefs are derived, and existing beliefs are revised. Therefore, the system's overall opinion about the preconditions and consequences of each operation changes over time.

Similarly, a goal is a *statement*, not a *state*, so is a *incomplete* specification of a certain aspect of the (internal or external) environment. There are inference rules used for *backward* inference, to produce derived goals, recursively from existing goals and beliefs. At any moment, there are usually many (input or derived) goals in the system, which are not necessarily consistent in what they specify as desired. If according to its experience the system expects the execution of a certain operation will achieve a certain goal, there is no guarantee that the expectation will be confirmed by future experience. Furthermore, the operation may have some undesired impact on other goals. Therefore, in the system each statement has a desire-value associated to summarize its overall relations with the goals considered, which, plus some other factors, will decide whether the system will take the statement as a goal.

Under the assumption of insufficient resources, the system cannot afford to explore all possibilities by interacting *every* task with *every* (relevant) belief. Instead, it can only let *selected* tasks interact with *selected* beliefs. The selections are based on the system's evaluation on the *priority* (which summarizes factors like *urgency*, *importance*, *relevance*, *usefulness*, etc.) of the tasks and beliefs, according to the system's experience. Since the system constantly gets new experience while communicating with the environment and working on the tasks, the evaluation results change from time to time.

The above mechanism inevitably leads to CPS. To NARS, each task corresponds to a new *case*, that is, the occurrence of a problem instance in a internal context (defined by the available knowledge and resources at the moment). What the system does to a task is determined by what the system knows about it (the existing relevant beliefs), how much resources the system can spend on it (the number of beliefs that will be selected), and the priority distribution among the beliefs (the access order of the selected beliefs). Since the above factors are constantly changing, the processing of a given task becomes unpredictable and non-repeatable according to the task alone, and the problem-solving process cannot be abstracted as APS.

CPS with Procedural Knowledge

In this section, more technical details are provided on the CPS process in NARS. Given the paper length restriction, here the focus is in the recent progress on CPS with *procedural* knowledge. For CPS with *declarative* knowledge, see (Wang, 1996; Wang, 2004).

NARS uses a formal language Narsese, which is term-oriented, that is, a statement in it typically has the form of *subject-copula-predicate*. While "There is a R relation among objects a, b, c" is usually represented in predicate logic as $R(a, b, c)$, in Narsese it becomes $((\times\ a\ b\ c) \rightarrow R)$, which states that the tuple [a, b, c] (the subject term) is a special case of the relation R (the predicate term), and '$\rightarrow$' (the copula) is the *inheritance* relation.

A "statement on statements" can be represented as a *higher-order statement*. For example, "An R_1 relation among objects a, b, c implies an R_2 relation among b, a, c" is represented as $(((\times\ a\ b\ c) \rightarrow R_1)\ \Rightarrow\ ((\times\ b\ a\ c) \rightarrow R_2))$, where the two terms are statements, and '$\Rightarrow$' is the *implication* relation, another type of copula.

An *event* is a statement with temporal information. For example, "An R_1 relation among objects a, b, c is usually followed by an R_2 relation among b, a, c" is represented as $(((\times\ a\ b\ c) \rightarrow R_1)\ /\!\Rightarrow\ ((\times\ b\ a\ c) \rightarrow R_2))$, where '$/\!\Rightarrow$' is implication plus the temporal information that the event as subject happens before the event as predicate.

With insufficient knowledge, in NARS no statement is absolutely true. A *judgment* is a statement with a truth-value attached, indicating the evidential support the statement gets from the experience of the system. A truth-value consists of two factors: a *frequency* factor in [0, 1], measuring the proportion of positive evidence among all available evidence, and a *confidence* factor in (0, 1), measuring the proportion of current evidence among future evidence, after the coming of new evidence of a unit amount. For example, if statement $((\times\ a\ b\ c) \rightarrow R)$ has been tested 4 times, and in 3 of them it is true, while in 1 of them it is false, the truth-value of the statement is $f = 3/4 = 0.75,\ c = 4/5 = 0.80$, and the judgment is written as "$((\times\ a\ b\ c) \rightarrow R)\ <0.75; 0.80>$".

A *goal* is an event the system wants to achieve, that is, the system is willing to do something so that the truth-value of that statement will approach $<1.00; 1.00>$ as closely as possible. The attached desire-value of each goal is the truth-value of the system's belief that the achieving of the goal really leads to desired situations.

An *operation* is an event that the system can directly realize by executing some program (which are usually not written in Narsese). In other words, it is a statement with a "procedural interpretation", as in logic programming. For example, if the term R corresponds to the name of an operation, and a, b, and c are arguments of the operation, then $((\times\ a\ b\ c) \rightarrow R)$ represent the event that R is applied on a, b, and c, which is a special case for the three to be related.

The system's knowledge about an operation is mainly represented as beliefs on what the operation implies, as well as what it is implied by. Each belief provides partial information about the precondition or consequence of the operation, and the overall meaning of the operation, to the system, is the collection of all such beliefs. To simplify the description,

in the following a term, like S, will be used to represent a statement, such as $((\times\ a\ b\ c) \rightarrow R)$.

In NARS, a judgment $(S_1 \not\Rightarrow S_2) <f; c>$ can be used to uniformly represent many different types of knowledge.

- If S_1 is directly about an operation, but S_2 is not, then the judgment represents a belief on an *effect* or *consequence* of the operation;
- If S_2 is directly about an operation, but S_1 is not, then the judgment represents a belief on a *cause* or *precondition* of the operation.

Such a judgment can be used by various rules. In forward inference, it and a judgment on S_1 can derive a judgment on S_2 by the *deduction* rule, as a prediction; it and a judgment on S_2 can derive a judgment on S_1 by the *abduction* rule, as an explanation. This judgment itself can be derived from the system's observation of event S_1 followed by event S_2, by the *induction* rule, as a generalization. In backward inference, this judgment and a goal (or a question) on S_2 can derive a goal (or a question) on S_1. Different rules use different truth-value functions to calculate the truth-value of the conclusion from those of the premises. The details of these rules, with their truth-value functions, can be found in (Wang, 2006).

For more complicated situations, both the S_1 and S_2 in above judgment can be compound statements consisting of other statements. For example, very common the condition part of an implication statement is a "sequential conjunction", as in $((S_1,\ S_2)\ /\Rightarrow\ S_3)$, which means the event sequence "S_1, then S_2" is usually followed by event S_3. When S_2 is an operation, such a statement represents its (partial) precondition and consequence. When S_3 is a goal, $(S_1,\ S_2)$ indicates a plan to achieve it.

The inference rules of NARS carry out various cognitive functionalities in a uniform. Beside the above mentioned prediction, explanation, and generalization, the system can also do planning (finding a sequence of operations that lead to a given goal), skill learning (forming stable operation sequence with useful overall function), decision making (choosing among alternatives), etc., though in this paper their details cannot be explained. Working examples of these functions in NARS can be found at the project website *http://code.google.com/p/open-nars/*.

In NARS, all beliefs (existing judgments) and tasks (new knowledge, questions, and goals) are clustered into *concepts*, according to the terms appearing in them. The system runs by repeating the following working cycle:

1. Select tasks in a task buffer to insert into the corresponding concepts, which may trigger the creation of new concepts and beliefs, as well as direct processing on the tasks.
2. Select a concept from the memory, then select a task and a belief from the concept.
3. Feed the task and the belief to the inference engine to produce derived tasks.
4. Add the derived tasks into the task buffer, and send report to the environment if a task provides a best-so-far answer to an input question, or indicates the realization of an input goal.
5. Return the processed belief, task, and concept back to memory.

The selections in the first two steps are all probabilistic, with the probability for an item (concept, task, or belief) to be selected proportional to its priority value. In the last step, the priority of the involved items are adjusted according to the immediate feedback obtained from the inference result.

Now we can see that for a given task, its processing path and result are determined by the beliefs interacting with it, as well as the order of the interactions (that is, inference steps), which in turn depends on the items in the memory (concepts, tasks, and beliefs), as well as the priority distributions among the items. All these factors change constantly as the system communicates with the environment and works on the tasks. As a result, there is no algorithm specifying the inference step sequence for a task. Instead, this sequence is formed at run time, determined by many preceding events in the system. In this way, task processing (that is, problem solving) in NARS becomes "case by case".

Comparison and Discussion

CPS and APS are different approaches of problem solving in computer. In APS, it is the programmer who solves the problem (as a class), and the computer just applies the solution to each instance of the problem. In CPS, it is the computer that directly solves the problem (as a case), depending on its available knowledge and resources. A CPS system still follow algorithms, but these algorithms are not solutions of domain-specific problems. Instead, they are domain-independent solutions of "meta-problems" like the handling of input/output, the carrying out of the inference steps, the allocating of resources, etc.

These two approaches are suitable for different situations. Given the scope of the problems a system faces, APS is preferred when there are sufficient knowledge (to get a problem-specific algorithm) and resources (to execute the algorithm), while CPS is an option when no problem-specific algorithm is available and affordable. CPS gives the system creativity, flexibility, and robustness, though it lacks the predictability, repeatability, and reliability of APS.

CPS processes are difficult to analyze, because the traditional theories on computability and computational complexity become inapplicable at the problem-solving level (though it may be applied in other levels), as the solvable problems and the solution costs all become context-sensitive and practically unpredictable, unless the system's experience in the past and near future (when the problem is being solved) is fully known, and the system can be simulated step-by-step with all details.

Some claims on the limitations of AI are based on the "non-algorithmic" nature of intelligence and cognition, as in (Dreyfus, 1979; Penrose, 1989). When facing CPS systems, all such claims become invalid, because the problem-solving processes in these systems are already non-algorithmic. This topic has been discussed with more details in (Wang, 2007), and will not be repeated here.

A large part of AI research is driven by the challenge of problems for which no efficient algorithm is available. The

typical response is to find such an algorithm first, then to use it in APS. CPS is different from these techniques in its basic idea, though still related to them here or there.

One of the earliest AI technique is *heuristic search* (Newell and Simon, 1976). Since all possible solutions come from permutations of a constant set of basic operations, problem solving in theory can be described as searching for a path from the initial state to a goal state in a state space. Because exhausting all possibilities usually demands unaffordable resources, the key becomes the selection of paths to be explored. NARS is similar to heuristic search in that (1) it compares alternative paths using numerical functions, since in NARS the truth-values, desire-values, and priority-values all have impact on the order by which the alternatives are explored, and (2) the system usually gets satisfying solutions, rather than optimal solutions. Their major differences are that at each step NARS does not evaluate a static list of alternatives according to a fixed heuristic, but recognizes and builds the alternatives by reasoning, and allocates resources among them, so to explore them in a *controlled concurrency* (Wang, 1996), which is similar to *parallel terraced scan* (Hofstadter and FARG, 1995). Furthermore, in NARS the heuristic information is provided mainly by the domain knowledge, which is not built into the system, but learned and derived, so is flexible and context-sensitive, while a heuristic algorithm has a fixed step sequence.

A related technique is *production system*, or *rule-based system*, where each state change is caused by the applying of a rule, and different solutions correspond to different rule-application sequences. NARS looks like such a system, since it also describes a problem in a formal language, and modifies the description by rules during inference. However, in traditional rule-based system there is a static long-term memory (containing *rules*) and a changeable working memory (containing *facts*) (Newell, 1990). In a problem solving process, only the latter is changed, and after the process, the working memory is reset. Therefore, the system still does APS, since it provides a (fixed) mapping from input (problem instances) to output (solutions), even though here the "algorithm" is not explicitly coded as a program, but is implicitly distributed among the rules and the control mechanism (which is responsible for selecting a rule to fire in each working cycle). On the contrary, in NARS the content of memory is modified by every problem-solving process, so the processes have strong mutual influence, and it is impossible to analyze one of them without the others.

The "case" in CPS should not be confused with the same term in *Case-Based Reasoning* (Leake, 1996), which solves problems by revising solutions of similar problems — that is still APS, with the algorithms distributed among the cases and the control mechanism (which is responsible for selecting similar cases and putting together a new solution).

More complicated forms of APS can be found in works on *randomized algorithms* (Cormen et al., 2001), *anytime algorithms* (Dean and Boddy, 1988), and *metareasoning* (Russell and Wefald, 1991). Though driving by different considerations and suggesting different approaches, each of these technique solves a problem with a family of algorithms, rather than a single one. For a given problem instance, some outside factor (beyond the input data) decides which algorithm in the family will be selected and applied. In randomized algorithms, the selection is made randomly; in anytime algorithms, it is determined by the executing time restriction; and in metareasoning, it is the result of an explicit deliberation. If we treat these factors as an additional argument of the problem, used as the index of the algorithm selected from the family, all these situations are reduced to APS. On the contrary, CPS cannot be reduced into APS in this way, since it is not even a selection among preexisting algorithms. If a problem (instance) is repeatedly solved in NARS, the solution does not form a probability distribution (as in a randomized algorithms). Even if the same amount of time is allocated to a problem, in NARS the results can still be different, while according to anytime algorithm and metareasoning, the results should be the same. Even so, NARS still share some properties with these approaches. For example, given more resources, the system usually provides better solutions, as an anytime algorithm (Wang, 1996).

In NARS a problem-solving process is non-algorithmic, because the process is not built into the system, but formed by the learning and adaptation mechanism of the system. To learn problem-solving skills from environment feedback is not a new idea at all (Turing, 1950). There is a whole *reinforcement learning* field aimed at the optimization of the reward from the environment to the system's operations (Kaelbling et al., 1996). Also, *genetic programming* provides a powerful way to generate algorithms for many problems (Koza, 1992). Though these techniques (and some others) are very different in details, they are still within the APS framework given at the beginning of the paper: first, find an algorithm for a problem class, then, apply the algorithm to each problem instance. What these techniques aim is to replace the human designer in the first phase of APS. This is different from the aim of CPS, which merge these two phases into one, by directly solving problem instances in a non-algorithmic manner. There are indeed many situations where algorithms are desired, and therefore some kind of algorithm-learning technique will be preferred. However, there are also situations where the system cannot satisfy the demand of these algorithm-learning techniques on knowledge and resources, so CPS will be more proper. CPS does not reject all forms of skill learning, as far as it does not reduce the problem-solving process into APS. As described in the previous section, NARS can learn procedures.

In summary, CPS is designed for a situation where no existing technique can be applied, rather than as an alternative to an existing technique in the field for which it is designed. CPS is similar to the existing techniques in many aspects, but cannot be reduced to any of them.

Conclusion

For problem-solving, the "case-by-case" approach (CPS), which is used in NARS and advocated in this paper, is different from the algorithmic approach (APS) by taking the following positions:

- Do not define a "problem" as a class and use the same method to solve all of its instances. Instead, treat each

"problem instance" as a "problem" on its own, and solve it in a case-by-case manner, according to the current (knowledge/resource) situation in the system.

- Do not draw a sharp line between solutions and non-solutions for a given problem, and treat all solutions as equally good. Instead, allow solutions to be partial, and compare candidate solutions to decide which one is better.
- Do not insist on "one problem, one solution". Instead, allow the system to generate zero, one, or a sequence of solutions, each of which is better than the previous ones.
- Do not depend on a predetermined algorithm to solve a problem. Instead, cut a problem-solving process into steps. Though each step follows an algorithm, the overall process is formed by linking steps together at run time in a context-sensitive manner.
- Do not predetermine the method by which a problem is processed in each step. Instead, let the selected problem and available knowledge decide how the problem is processed in that step.
- Do not attempt to use all relevant beliefs to solve a problem. Instead, in each step only consider one of them, selected according to their priority values.
- Do not solve problems one after another. Instead, process problems (and subproblems) in parallel, but at different speed, according to their priority values.
- Do not throw away the intermediate results at the end of a problem-solving process. Instead, keep them for future problems.
- Do not isolate each problem-solving process in its own working space. Instead, let all problems interact with the same memory.
- Do not attempt to keep all beliefs forever. Instead, remove items with the lowest priority when the memory is full.
- Do not process each problem with a fixed resources supply. Instead, let the processes compete for resources.
- Do not keep a fixed resources distribution. Instead, adjust the priority distribution according to the experience of the system and the current context, so as to give important and relevant items more resources.

Some of the above issues are discussed in this paper, while the others have been addressed in related publications (Wang, 1996; Wang, 2004; Wang, 2006).

This new approach for problem-solving is proposed as a supplement to the traditional (algorithmic) approach, for situations where the system has insufficient knowledge and resources to apply or to build an algorithm for the problem it faces.

The practice of project NARS shows that such an approach can be implemented, and has properties not available in traditional systems. Since the capability of such a system is not only determined by its design, but also by its experience, it is hard to evaluate the potential of this approach in solving practical problems. However, at least we can say that this approach is exploring a territory beyond the scope of classic theory of computation, and it is more similar to the actual thinking process of the human mind.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, McGraw-Hill Book Company, 2nd edition.

Davis, M. (1958). *Computability and Unsolvability*. Mcgraw-Hill, New York.

Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of AAAI-88*, pages 49–54.

Dreyfus, H. L. (1979). *What Computers Can't Do: Revised Edition*. Harper and Row, New York.

Hofstadter, D. R. and FARG (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, New York.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts.

Leake, D., editor (1996). *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press, Menlo Park, California.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman & Co., San Francisco.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts.

Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126.

Penrose, R. (1989). *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford University Press.

Russell, S. and Wefald, E. H. (1991). Principles of metareasoning. *Artificial Intelligence*, 49:361–395.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, LIX:433–460.

Wang, P. (1995). *Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence*. PhD thesis, Indiana University.

Wang, P. (1996). Problem-solving under insufficient resources. In *Working Notes of the AAAI Fall Symposium on Flexible Computation*, pages 148–155, Cambridge, Massachusetts.

Wang, P. (2004). Problem solving with insufficient resources. *International Journal of Uncertainty, Fuzziness and and Knowledge-based Systems*, 12(5):673–700.

Wang, P. (2006). *Rigid Flexibility: The Logic of Intelligence*. Springer, Dordrecht.

Wang, P. (2007). Three fundamental misconceptions of artificial intelligence. *Journal of Experimental & Theoretical Artificial Intelligence*, 19(3):249–268.