# Problem Solving without Search

**Rodrigo M. M. Ventura** and **Carlos A. Pinto-Ferreira**
Instituto de Sistemas e Robótica
Instituto Superior Técnico
Rua Rovisco Pais, 1
1096 Lisboa Codex
Portugal
emails: {yoda,cpf}@isr.ist.utl.pt

## Abstract

In this paper the approach to problem solving using societies of agents is discussed. Albeit being a very promising topic in Artificial Intelligence, it seems to lack strong theoretical and methodological foundations and easy-to-use implemented systems in order to prove its usefulness and adequacy. This is the reason a methodology for problem solving based on societies of agents is presented. The approach to some problems using the implemented system — RUBA — and the corresponding results follows. The source code and documentation of RUBA is available at the project's web page: `http://www.isr.ist.utl.pt/~yoda/ruba`.

## 1 Introduction

A very frustrating aspect in the study and implementation of classical problem solving algorithms is the recognition that most of their "intelligence" was *a priori* incorporated by their developers. As a result, little room is left for surprise as their outcomes are, in most cases, *predictable* (in fact, an unexpected result corresponds, almost always to some bug in the program...). Another uninteresting characteristic of such systems is their *reiteration* behavior, as they usually return the same output, over and over again, given the same set of initial conditions. The third negative aspect deserving consideration is their *opacity* as their recipe-like code is only understandable when an adequate translation — from *the how* to *the why* — is performed. And the last, their modification is very difficult given the fact that, even a simple change demands new coding as these implemented algorithms are, in most cases, completely *inflexible* and not reusable.

On the other hand, problem solving methods based on *societies of agents* are intended to exhibit characteristics which are dual of the mentioned ones: solution *emergence*, instead of predictability — as there is no algorithm running — outcome *variety*, instead of reiteration — as the complexity of the interaction among agents seldom allows the same answer twice — *readability*, instead of opacity — if agent behavior is described by a rule-base mechanism, and finally, easy *incremental* updating, as it is always possible to create and run new agents to take care of the new aspects of a problem.

The basic idea underlying this new problem solving paradigm is that, instead of considering the solution of a problem as the result of a process of searching in a state space, it departs from the premise that *a problem is nothing but a set of restrictions to be (partially) met*. Some of them are structural and consequently should be strictly respected — for instance, it is not possible to put different objects in the very same place — and some are not structural and, as a consequence, can be relaxed. When it is possible to create a society of agents, each one taking care of its subset of restrictions, the "global" problem is suitable of being handled by agent oriented problem solving methods and, instead of search, it is solved by a process of negotiation. What is particularly interesting in this approach is the fact that different agents can represent different (and possibly) conflicting interests which should be taken care of (imagine, for example, the process of constructing a timetable for a school). Now assume, for instance, that after the interaction among the agents it is found out that the solution is not satisfactory. As this corresponds to the fact that there is a particular interest which are not being considered yet, it is simply necessary to introduce one (or more) agents responsible for the set of these new constraints, and restart the interaction (for instance, if the solution of the timetable problem is considered poor because of being resource consuming, an agent in charge of the resource administration should be integrated in the system).

Finally, it is important to point out that this method is not a panacea: on the one hand, it is not suited for optimization problems — as it usually allows the finding out of pragmatical solutions for real-world problems — and, on the other hand, it is not

adequate for solving puzzle-like problems where variables are inextricably linked: interestingly, these two classes of problems are poorly handled by human beings.

## 2 Nature of AI problems and solutions

Consider a division of AI problems in two classes, according to their solving methods. The first one includes the problems usually addressed by classical AI search methods. These problems must have a unambiguous specification and the solutions sought are few (if not unique, in optimization problems). Therefore, it makes sense to use state-space search based techniques. The order of complexity of these problems can be quite high, but the solution is in itself simple (in the sense of being defined by a small number of variables). The search for the best solution to the traveling salesman problem illustrates this kind of problems and solutions. What makes this problem hard to solve is the search for an optimal solution, rather than a reasonable one (as humans do). Combinatorial explosion, which can make things quite difficult is a commonplace on this kind of problems.

The second class of problems comprises real-life problems, the ones that humans usually solve in relatively short time, everyday. Unlike the ones outlined above, these problems have a more complicated specification and the optimality of the solution is of small concern. The specification has usually a certain degree of flexibility. But most important of all, a *satisfycing* solution is desired rather than an optimal one, which is intractable in this context. The issue of *satisfycing* versus *optimizing* has been introduced by Simon [Simon, 1959]. It is well known that human reasoning is much more related with the search for a solution that meets their needs, rather than looking for an optimal one. No traveling salesman is known to be solving NP-complete problems.

## 3 Methodology

The purpose of this paper is to present a methodology to allow AI applications to be scalable in complexity. The term complexity is used here not to denote computational complexity but as specification complexity (in variety of restrictions). The proposed approach is to shift from the classical AI approaches to the problem itself to be solved. Before proceeding, two assumptions are made: first, that the increase in complexity does not always invalidate the decomposition of the problem in small and tractable parts (the *divide and conquer* principle). Note that no assumption is made on the nature of these parts. Next, it is assumed that the linkage of these parts is weak, *i.e.*, the increase in complexity does not affect (much) the nature of the interactions of the parts. For instance, consider the classical scheduling problem. A natural division in parts defines the events to be scheduled. Although each part may contain a considerable degree of complexity (restrictions both on time and place), the basic principle that two events cannot occur at the same time at the same place is maintained. And this principle binds these parts from the start.

Let us now focus on these parts, which in this methodology are represented by agents. The problem specification is thus distributed over several agents. Furthermore, the control strategy (the centralized entity that drives the problem solving process from the start to the end) is also distributed over all agents. Note that this idea does not invalidate any hierarchical relationships among them. One agent can still be responsible for a central control strategy, totally or partially.

It seems clear that a single agent by itself doesn't look much interesting from the operational point of view, in what problem solving is concerned. Instead, the result of a set of interacting agents (societies) are able to enlarge the usefulness of the idea. This follows from the fact that the agent concept is defined from an external point of view, namely the interaction with the environment. No reference is made to the inner workings of an agent, allowing interesting combinations of quite heterogeneous agent societies. One key concept underlying agent societies is *distribution*. This paradigm is a strong one. It has founded an AI field (DAI) and inspired several other areas, like neural networks (see [Lippmann, 1987] for instance). The Minsky's classic "Society of Mind" [Minsky, 1988] makes extensive use of this concept.

The question that immediately follows the concept of agent societies is how exactly agents do interact among them. If they are all pursuing the same global problem (despite the fact that each one tries to solve its particular local problem), and if there are restrictions inherent to the problem, between the parts agents represent, then a *negotiation* model becomes necessary. In this model the agents exchange messages in order to negotiate one or more issues relevant to the problem to be solved (see [Zoltkin and Rosenschein, 1996] for a thoughtful study on agent negotiation). Each agent has both to secure his own positions and to be flexible enough in order to concede some of his points. Note that in a strategy like this one, no one is particularly responsible for the outcome of the negotiation. There is also no guarantee that this outcome is optimal in any sense, not even sub-optimal. But the relevant aspect of this idea is that the solution *emerges* from the agent society interactions. This emergence is one of the main issues of this paper. Brooks' pioneering work ([Brooks, 1989]) has demonstrated the relevance of this issue in the context of robotics. The solution is not given through a closed form as an algorithm, but what is proposed is

to program the agents and let them solve the problem. Note the distinction between building an algorithm to solve a problem, where the solving task was indeed accomplished by the algorithm designer, and this notion of solution emergence. In this case, the agents are responsible for solving the problem together. No agent is embodied with the knowledge for solving it (otherwise it would solve the problem by itself). It is the group concept which allows the solution to emerge.

Underlying any formulation of problem solving, there is a concept of state-space search. But this approach based on agent negotiation has no explicit state-space search scheme. Yet, there is a process of negotiation between the agents, which can be seen as another paradigm of search. Picture a situation where a set of agents is negotiating an issue of the global problem. Although each individual agent has no concern with the global state of the system, from the outside the agent society is running an implicit state-space search. Note that there is no explicit (or centralized) representation of the global state of the system. This state is distributed over all agents. And since no particular agent behaves with explicit regard to the evolution of the global state, this search procedure produces solution emergence.

When considering state space searching, combinatorial explosion arises from the irrelevant combinations considered by a centralized search algorithm. If the problem complexity scales up, the search procedure usually becomes intractable. This fact results from the implicit assumption (when defining the state space) that all parts of the problem are mutually dependent. Assume that the state $S$ can be represented as the concatenation of $k$ sub-states: $S = (p_1, p_2, \ldots, p_k)$. If the sub-state space for each $p_i$ is $\mathcal{P}_i$, the total state space of $S$ becomes the Cartesian product of these sub-spaces: $\mathcal{S} = \mathcal{P}_1 \times \ldots \times \mathcal{P}_k$. A search algorithm based on this state space $\mathcal{S}$ contains all possible combinations of the sub-states, forcing the search algorithm to look for a solution in this space. From the search algorithm point of view, all combinations are equally relevant from the start. From the DAI point of view, the situation is the opposite. When distributing the problem among the agents, the more independent these parts are, more independently agents will be able to work on a solution, thus maintaining the resolution complexity not higher than each agent's one. Going back to the above state representation $S = (p_1, p_2, \ldots, p_k)$, consider that each part $p_i$ can be assigned to each agent $A_i$. Then, in the best case scenario, *i.e.,* all parts $p_i$ are independent, each agent has just to work with the sub-state space $\mathcal{P}_i$. As the parts become more and more dependent, negotiation opens the state space to partial Cartesian products. For instance, negotiation between agents $A_l$ and $A_m$ generate a $\mathcal{P}_l \times \mathcal{P}_m$ sub-space. As agents negotiate between them, sub-spaces are expanded into larger ones, but the implicit search remains constrained to these sub-spaces, including just the relevant agents for each negotiation. One can conclude that while search methods assume that the problem parts are dependent, the DAI methods assume the opposite, and start from there, using dependency as needed, thus avoiding combinatorial explosions.

In order to put the above ideas into practice, a development environment was created. This system is called RUBA, and will be briefly presented in section 5. The idea behind RUBA was to provide the programmer with the framework described in this section.

One final point which deserves a note concerns programming language levels. What makes the difference between several programming languages is its proximity to the human level of representation and reasoning. Programming languages are designed to be read by humans, rather than by machines, since they don't care about semantics of the words used in source code [Abelson *et al.*, 1985]. The adequacy of a language is function of the application the programmer has in mind. But most of all, the gap between the conceptual level and the language level is desirable to be as small as possible. Of course agent societies can come close to human's conceptual level if one pictures agents as small human emulators. The idea is to imagine a society of human-like beings, and emulate their behavior by agents. Agents are not supposed to emulate fully-featured human beings, but only to emulate how human beings would behave (in a simplified fashion) in a particular society solving a particular problem.

It is reasonable to ask in what way these ideas contribute to solve complicated real-life problems outlined at the beginning of this section. Two ideas to answer the question are proposed. The first one is the distribution of the problem over several agents. Each agent represents a part of the problem to be solved. The fine tuning of an agent, in order to adapt itself to a particular aspect of the problem is localized to the agent, simplifying the process of adapting the system to problem specification updates. It is the agent's autonomy that makes this localization possible.

To illustrate this methodology, consider the problem of timetabling. This is a typical real-life problem this methodology is aimed at. The formulation is extremely complex, considering all practical constraints that all intervening parts have (teachers, students, space allocation, etc.) to take care of. Search-based methods have difficulty in complying with all this complexity, but human beings, with far less brute force computational resources have practical ways of accomplishing the task. The usual scenario for manual schedule building is a meeting between all interested parts (teachers, school clerks, etc.) with a blackboard containing a schedule proposal (for instance, the last year's one). The ones that have problems with their schedules (some lecture they can not give), start nego-

tiating with others, trying to exchange lecture hours. After some time, and some concessions in order to accommodate each others lectures, a final solution is reached. The state space is the same, but the fact that the search is distributed, without any central control strategy, allows the problem to be solved in a reasonable amount of time. The methodologies discussed in the above paragraphs can be naturally applied to this problem, assigning an agent to each interested part, and using a negotiation strategy similar to the one humans use.

## 4 Framework

In this section a framework capable of implementing the ideas outlined above is presented.

First the mechanisms that underlie agent interaction are focused. In this agent-oriented problem solving scenario, there are at least two kinds of entities agents communicate with. The first ones corresponds to the other agents and the second respects to external entities responsible for providing the agents with all information regarding the problem, as well as retrieving the results. From an agent's point of view, there is no point in making the communication different. It can use the same mechanism.

An agent society is defined by a rich interaction among the participating agents. They not only exchange information in one-to-one basis, but also require a mechanism to broadcast information to all agents. A *blackboard* was chosen to allow this interaction, mainly because it provides flexible means of communication [Erman *et al.*, 1980]. This blackboard is able to ensure all the information exchange mechanisms stated above. The communication is based on atomic messages which are posted on the blackboard, and can be seen by all agents. The messages include the information that specifies to whom is the message sent. The destination of a message can be a specific agent, a group of agents, or all agents that have access to the blackboard. Note that there is no need for inter-agent communication mechanisms. Agents can exchange private messages with any other agent (that shares a common blackboard) without knowing of each others existence, prior to this interaction.

Note that a blackboard may not be unique within an environment. Groups of agents can share different blackboards. The scope of a blackboard is limited to the agents which are able to exchange messages with. On the other hand, agents themselves can be divided into several sub-societies. The agent concept unleashes imagination to think of quite complex structures based on agent societies.

In the previous section it was stated that agents exist in an environment. In this case, the environment these agents live in is primarily one of more blackboards. It is however possible to build more complex environments. Agents can be embodied with mechanisms that allow them to interact with complex environments. However, the blackboards can still be the basic communication mechanism.

## 5 Implementation

In this section an overview of an implementation of a system devoted to create agent societies, under the perspective presented in this paper, will be presented. The system is called RUBA[1] and comprises a set of tools built in a LISP (with CLOS) environment.

The programming model of RUBA is quite simple. It comprises two types of (CLOS) objects: *agents* and *blackboards*. A *blackboard* is simply a data structure which holds messages. Messages can be posted or removed at will by agents. An *agent* contains three elements:

- A *set of variables* that forms the current agent's state. These variables are normal programming variables, and can contain for instance agent's knowledge or data related with the problem in hand;
- A *set of rules*, which define the agents behavior. Each one is a production rule `IF-THEN-ELSE`. These rules can be further organized into subsets, in order to work as a state machine. One (or more) variable(s) can be assigned to represent a state in this sense (it will be called *mstate* thereafter, to avoid confusion). Then, only rule sub-sets marked with that mstate are considered. The `THEN` or `ELSE` part can optionally indicate to what mstate the agent must switch, depending on the logic value of the `IF` part. This embedded state machine has no fundamental role in RUBA, but it simplifies agent programming and improves the rules readability;
- A *set of links to blackboards* the agent has contact with;

Both the rules' antecedent and consequent are either calls to RUBA primitives, or LISP expressions. The RUBA primitives are lambda expressions (functions) which are executed in the agent's context (agent's variables and blackboards). The expressions that appear in the rules use these primitives, although direct access to LISP functions is possible (but not desirable, for readability reasons). These primitives hide the implementation aspects from the rule construction. These rules must be as (human) readable as possible. This way, agent rules can be written with great readability, simplifying the process of translating the human emulation behavior into RUBA rules.

[1] The name RUBA is the acronym of "**R**ule **B**ased **A**gents". This name doesn't capture the main features of the system, but remained so for historical reasons.

## 6 Results

Four examples were initially explored with the RUBA system. The first one is an experiment on negotiation, where the agents can trade objects, in order to get a desired one. The negotiation model is quite simple. There are three kinds of messages exchanged between the agents:

1. *Complaints*, stating the need for the acquisition of a certain object;
2. *Proposals*, containing a proposal of a trade;
3. *Trades*, stating the acceptance of a proposal.

Each agent behaves as a state machine, inspired on how humans would negotiate given this operators.

The second example is the classic n-queens problem. The goal is to deploy $n$ chess queens on a normal chess board, in a way that no queen attacks any other. It's a complex combinatorial problem[2]. Each agent represents a single queen, that can be in one of two states: in or out of the board. Initially, each agent (queen) is placed outside the board. Then, each one tries to move to a vacancy on the board, *i.e.*, a place where no other queen is attacked (and therefore, no other queen attacks it). If no vacancy is found, a special message is sent to the common blackboard (with the name "shake") in order to cause a small perturbation in the board[3]. The interesting result was that such a simple strategy allowed the system to find a solution in a small amount of time. But the main point is not how long it took to reach a solution, but rather the *true emergence* of this solution! Some results for board sizes ranging from 4 to 15 can be found in figure 1. This graphic shows how the system scales well given hard combinatorial problems. There is neither an explicit search algorithm running nor an evaluation function. In fact, this approach is quite close to the human way of solving the problem. No one pictures in his mind a complex search tree. Instead, the reasonable human approach is to move each piece, once at a time, in the hope that a solution is found.

Next, a simple resource allocation problem was approached. Each agent represents a *resource unit* with the ability of exchanging *products* with other agents. Instead, the agents were given a simple strategy in order to reduce the waste on resource utilization. As it was stated earlier in this paper, the objective is not to optimize, but rather to find a satisfycing solution. The point is that the problem was *distributed* over the agent society, where the solution *emerges*, albeit

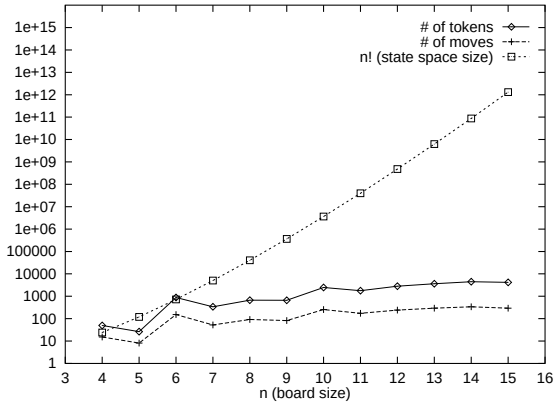


Figure 1: Some results from the n-queen problem, for a range of board sizes (n). The state space size is the number of combinations of $n$ queens in an $n \times n$ chess board, constrained to the fact that there can only be one queen in each row and one in each column. This value gives an idea of the combinatorial complexity of the problem. The other two graph represent averages over 20 runs of the RUBA system. The number of moves denotes the number of times any queen moves, both entries to the board and moves within the board, until a solution is found. Each time an agent is (randomly) activated, it is called a *token*. The third graph represents the number of tokens given.

the *naïve* strategy embodied in each agent. It would be trivial to devise a classic greedy algorithm in order to solve the problem. The question is, how much effort would be necessary if the problem formulation were more complex. And how easy it would be to cope with the new complex algorithm. This agent's approach brings two fundamental points: first, the problem representation remains simple, because it's *distributed* among the agents, and second, there is no danger of combinatorial explosion, since there is neither explicit search nor iterative algorithms. Resolution complexity is therefore less sensitive to complexity of the problem statement.

The fourth example explored with RUBA was a simplified jobshop scheduling problem. As in the last example, a simple strategy was used for each agent. In this case it resembles a cards game, where the agents (players) represent machines and the cards represent the products. The solution is built on the blackboard as time goes by, as agents (machines) try to exchange cards (products) with the other agents, according to each one's interests. No agent has any knowledge concerning other agent's interests, keeping therefore the problem *distributed* among all agents. Most of the remarks made in the last example can be applied to this one.

[2]Constraining the search space to the ones with only one queen in each row and in each column, there are $n!$ combinations.

[3]In fact, the agent that receives the "shake" message tries to move to other vacancy, after getting out of the board, and removes the message from the blackboard.

## 7 Conclusions and related work

In this paper an agent based methodology for problem solving was discussed. The problem statement is *distributed* over several agents that constitute an agent society. The role of each agent is to represent an entity which is relevant to the problem in hand, but instead of designing a global strategy, each agent is embodied with an adequate individual behavior. The idea is to emulate how a human being would behave under such circumstances. When several agents are put together, each one following its own strategy, the outcome is unpredictable. Furthermore, if a solution for the problem is ever found, it will *emerge*. This is an important point. However, one cannot expect that a solution will be optimal (or sub-optimal) in any way, not even that it will be found at all. Optimality was not a concern here. As it was stated at the beginning of the paper, as the nature of problems that this methodology aims at are real-life problems. In this kind of problems, it is often good enough that any solution is found at all. Optimality issues are usually intractable at this level.

The proposed approach is to provide the agents with the means that allow them to seek for a solution for themselves, not knowing that a global solution is being sought. The fundamental difference is that in this approach, the ones that work out a solution are the agents themselves. There is no pre-defined algorithm, from which theorems can be derived regarding the solution reachability or the solution optimality. In this last case, the one that actually solved the problem was the algorithm developer. In an agent society scenario, the responsibility for seeking a solution is completely deployed on the agents.

There are several streams of work similar to this one. Some interesting results on solving the n-puzzle and more generally on methodological issues can be found in [Drogoul and Ferber, 1992] and [Drogoul and Dubreuil, 1993]. The former paper claims to have the first published results on solving n-puzzles with sizes over 100, using distributed agent methodologies. A through analysis on negotiation mechanisms can be found in [Zoltkin and Rosenschein, 1996]. Some issues on teamwork and cooperation can be found in [Tambe, 1997], using operators that spawn over groups of agents, representing commitment to joint plans. A survey on multiagent systems can be found at [Stone and Veloso, 1997], focusing at the learning aspects.

## 8 Acknowledgments

Authors would like to thank Nuno Gonçalves and Miguel Garção for their useful contribution to the design and implementation of RUBA.

## References

[Abelson *et al.*, 1985] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press and McGraw-Hill, 1985.

[Brooks, 1989] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journ. of Robotics and Automation*, RA-2(1):14–23, March 1989.

[Drogoul and Dubreuil, 1993] A. Drogoul and C. Dubreuil. A distributed approach to n-puzzle solving. In *Proceedings of the Distributed Artificial Intelligence Workshop*, 1993.

[Drogoul and Ferber, 1992] A. Drogoul and J. Ferber. Multi-agent simulation as a tool for modeling societies: Application to social differentiation in ant colonies. In *Actes du Workshop MAAMAW'92*, 1992.

[Erman *et al.*, 1980] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computer Surveys*, 12(2):213–253, 1980.

[Lippmann, 1987] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP magazine*, April 1987.

[Minsky, 1988] M. Minsky. *The Society of Mind*. Touchstone, 1988.

[Simon, 1959] H. A. Simon. Theories of decision making in economics and behavioural science. *American Economic Review*, 49:253–83, June 1959.

[Stone and Veloso, 1997] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. (under review), 1997.

[Tambe, 1997] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7, 1997.

[Zoltkin and Rosenschein, 1996] G. Zoltkin and J. Rosenschein. Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5, 1996.