Off-line Reasoning for On-line Efficiency *

Yoram Moses

Department of Applied Math and CS
The Weizmann Institute of Science
Rehovot, 76100 Israel

Moshe Tennenholtz
Robotics Lab
Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

The complexity of reasoning is a fundamental issue in AI. In many cases, the fact that an intelligent system needs to perform reasoning on-line contributes to the difficulty of this reasoning. In this paper we investigate a couple of contexts in which an initial phase of off-line preprocessing and design can improve the on-line complexity considerably. The first context is one in which an intelligent system computes whether a query is entailed by the system's knowledge base. We present the notion of an *efficient basts* for a query language, and show that off-line preprocessing can be very effective for query languages that have an efficient basis. The usefulness of this notion is illustrated by showing that a fairly expressive language has an efficient basis. The second context is closely related to the artificial social systems approach introduced in [MT90]. We present the design of a social law for a multi-agent environment as primarily an instance of off-line processing, and study this problem in a particular model. We briefly review the artificial social systems approach to design of multi-agent systems, introduced in [MT90]. Computing or coming up with a social law is viewed as a primarily off-line activity that has major impact on the effectiveness of the on-line activity of the agents. The tradeoff' between the amount of effort invested in computing the social law and the cost of the on-line activity can thus be viewed as an off-line vs. on-line tradeoff.

1 Introduction

Many activities in the framework of knowledge representation and reasoning are concerned with the following task: An intelligent agent has a given representation of a system (or a relevant aspect of the world), and a problem that relates to this representation. Its task is to solve this problem relatively efficiently. Typical examples include planning and computing whether a query is entailed by a given knowledge base. The reasoning in both of these cases can be thought of as on-line reasoning: When an input is given, our algorithm should reason about the system and about the problem instance, and should find a solution to the problem. In order to handle such problems, researchers often use general schemes of knowledge representation such as First Order Logic [MH81], logic programs [Kow74], semantic networks [Qui67], etc. Moreover, the reasoning is then carried out using general schemes of reasoning such as resolution for theorem proving, Prolog for logic programs, etc.

Consider the following question: Can our agent improve its on-line performance in a case where the model it uses (e.g., its knowledge-base or representation of the world) is fixed, and we know ahead of time that the agent is to be asked to solve many problems with respect to this model? Intuitively, it is clear that the answer should be positive; given a fixed model there should invariably be special-purpose algorithms for solving problems with respect to this particular model, instead of using general schemes of reasoning. This answer, however, is not very useful to the agent, without, our providing the agent with a way in which it can obtain such special-purpose algorithms. Our aim in this paper is to consider the problem of how an initial phase of off-line preprocessing can serve to reduce the complexity of the agent's on-line behavior. We are especially interested in systems where agents might be presented with (perhaps exponentially) many potential problems during the on-line activity. In such a case the agent should be allowed to perform rather extensive preprocessing without increasing the amortized cost per solution significantly. Specifically, we shall investigate two central contexts. The first involves an intelligent system that needs to compute whether particular formulas (queries) are entailed by its knowledge base. We present the notion of an *efficient basts* for a query language, and show that off-line preprocessing can be very effective for query languages that have an efficient basis. The second context involves dynamic multi-agent activity, and is closely related to the artificial social systems approach introduced in [MT90].

*This work was supported in part by a grant from the US-Israel Binational Science Foundation. The first author is incumbent of the Helen and Milton A. Kimmelman Career Development Chair.

Reactive approaches to problems in AI, related especially to planning, have been suggested in a number of works (see [Agr91],[AC87]). Some other works suggested to compile reactive behaviors in advance (see [Sch87]). However, the task of improving on-line behavior does not need to concentrate only on "real" reactive behavior. On-line behavior might refer more generally to the behavior of an agent where it faces various problems after the initialization of the system. The main task is to identify areas where off-line processing can be helpful and to suggest a particular type of solution for each such area. This is exactly the objective of this work.

This paper is organized as follows. Section 2 contains a high-level discussion about how off-line reasoning can be used in two general scenarios. In Section 3 we discuss a property of query languages that makes preprocessing of knowledge bases for these languages extremely effective. Section \ presents a fairly expressive query language with the property described in Section 3, and considers additional examples in which off-line preprocessing is useful in the context of knowledge bases. Section 5 relates the artificial social systems approach to the on-line vs. off-line reasoning paradigm.

2 Off-line versus on-line reasoning

Consider the well known problem of determining whether queries are entailed by a knowledge base, as discussed for example in [Lev89]. We assume that we have a knowledge base KB expressed in some logical language, and a query language QL in which queries concerning KB are formulated. Given a query α, we are interested in whether KB $\models \alpha$. This problem is intractable in the generic case. Moreover, even for tractable queries, the verification process might be very inefficient. 'There are two main approaches that are discussed in the AI literature for overcoming this difficulty:

1. Replacing problem-solving by model checking [HV91]: discuss knowledge bases that represent specific models, so that a query needs only to be checked against a model, rather than computing whether it is logically entailed by a knowledge base.

2. Decreasing the expressive power of the knowledge base and of the query language in order to have more tractable queries.

The first, approach is in fact the way in which relational databases are treated in theoretical computer science. The second is concerned with finding good tradeoffs between expressiveness and complexity (as is done in the knowledge base case by [Lev89], or in the case of multi-agent activity by [TM89]).

Another potential way for decreasing complexity is the following. Assume that any specific query can be verified in time /, where / might be large but feasible (e.g., super-linear but polynomial in the size of the knowledge base KB). The question is whether we can find a subset $QL' \subset QL$ of a feasible size, verify off-line for each member or of QL' whether $KB \models \alpha$ (all of this might take a lot of time), and use this off-line processing in order to make the on-line behavior more efficient. In the next sections we illustrate how this approach can be useful. We point to a general set of queries that can be handled in this way, and discuss specific examples. Notice that this approach can be treated as a type of multiple query optimization. However, the context of our query optimization (entailment by knowledge bases instead of retrieval from relational databases), and the actual way in which it is performed (off-line preprocessing instead of clever retrieval of a set of queries after their arrival) will be different from classical multiple query optimization (see [Sel88]).

Another area in which off-line design can improve on-line reasoning is multi-agent activity. A major issue in rnulti-agent activity is concerned with the coordination of agents' activities (see [BG88] for a collection of papers on this topic and on other topics in multi-agent activity). There are several ways for coordinating activity, such as deals and negotiations (e.g., [RG85], [DS83]). Here we will concentrate on a specific methodology for coordinating activity called artificial social systems. We will now briefly review the artificial social systems approach to multi-agent activity (see [MT90] and [MT91]) and discuss its connection to the off-line vs. on-line idea.

Consider the following scenario: You are the manager of a large warehouse that treats tens of customers at a time. You have just received a shipment of fifty mobile robots for the purpose of automating your warehouse. Clearly, at any given time different robots will serve the needs of different clients (some of them may concurrently perform additional maintenance operations). Before you can put the robots to work, you are faced with a major design problem involving how to make effective use of the robots. The artificial social systems approach to this problem, introduced in [MT90, MT91, Ten91], is to allow robots to work individually but force them to obey certain social laws, conventions, etc. The basic thesis of this approach is that the right social laws can significantly simplify both a robot's task of planning to achieve its goals, and the amount of work it needs to perform while actually pursuing the goal. The choice of these laws, however, is a delicate matter. Notice that devising appropriate social laws can be considered as off-line processing, while devising a plan for achieving a specific goal in a given situation (while obeying the social order) corresponds to solving a problem on-line. We discuss this issue in greater detail in Section 5.

3 Languages with an efficient basis

In this section we concentrate on off-line reasoning in the knowledge base case. We assume that each query formulated in the query language *QL* can be verified in time *t* (generally, *t* might be a function of the size of the knowledge base and of the size of the current query). For ease of exposition we will assume that the knowledge base *KB* and every query $\alpha \in QL$ are formulas in the language $\mathcal{L}$ of propositional logic. Let us denote the set of primitive propositions in this logic by $X = \{x_1, x_2, \ldots, x_n, \ldots\}$. We use $\mathcal{L}_i$ to denote the set of formulas of *C* whose primitive propositions are a subset of $\{x_1, x_2, \ldots, x_i\}$. As usual, we define the *size,* of a formula to be the number of symbols appearing in the formula. We will associate with every query language *QL* an infinite sequence $QL_1 \subseteq QL_2 \subseteq \cdots$, where $QL_i = QL \cap \mathcal{L}_i$. We say that a query language *QL'* is of *polynomial size* if there exists a fixed polynomial *p(i),* such that $|QL'_i| < p(i)$, where $|QL'_i|$ denotes the number of elements in QL'_i

We can spend time $t(|\alpha|)$ to compute any given query a. However, if we expect to encounter exponentially many queries of a certain size, it will be very costly to compute each one of them from scratch. In such a case it would be desirable to identify a small set of queries which, once computed, make computing other queries considerably simpler. If we consider polynormally many queries a reasonably small number, this leads to the following definitions:

Definition 3.1: A set $B \subseteq QL$ of queries will be called a *basis* for *QL* if every query in *QL* is equivalent to a conjunction of elements of *B*. *A* basis is called an *efficient basis* if its size (as a sublanguage of *QL)* is polynomial.

Given these definitions, we can now show:

Theorem 3.1: *Let QL be a query language, and let QV be an efficient basis for QL. Moreover, let KB be a knowledge base and let* $n \geq 1$ *be an integer for which* $KB \subseteq \mathcal{L}_n$ *holds. Finally, lei t be an upper bound on the time it takes to compute whether* $KB \models \alpha'$ *for an arbitrary* $\alpha' \in QL'_n$. *Then there exists an off-line computation of complexity 0(t • poly(n)), after which on-line testing whether* $KB \models \alpha$ *can be performed in time* $O(size(\alpha) \cdot \log n)$ *for every* $\alpha \in QL_n$.

The proof of this result, as well as all other results reported on in this paper, will appear in the long version of the paper. We remark that the size of the knowledge base *KB* in Theorem 3.1 plays a role only in affecting the parameter *t.* Once the preprocessing is done, the knowledge base can be ignored, and the complexity of computing entailment of a query is linear in the size of the query and in logn.

Notice that, assuming the size of a query is negligible relative to the size of *KB,* this result shows that in the abovcmentioned case we are able to get on-line reasoning which is much more efficient than what can be achieved without appropriate off-line computations.

4 Efficient On-Line Reasoning

In the previous section we showed that off-line preprocessing can be very effective for query languages that have an efficient basis. One wonders, however, whether this family contains any natural and/or useful query languages that can be used in practice. We now present such a query language. Recall that a CNF formula is a conjunction of clauses each of which contains a disjunction of literals. (A literal is a primitive proposition or the negation of one.) A k-CNF formula is a CNF formula where each clause contains no more than *k* literals. It is not hard to show:

Proposition 4.1: *For every* $k > 0$, *the k-CNF query language has an efficient basis.*

Recall that every formula of propositional logic is equivalent to a CNF formula. In particular, every formula is equivalent to a k;-CNF formula for a sufficiently large *k.* Moreover, formulas that serve as queries to a knowledge base are likely to be expressible as k-CNF formulas for a rather small *k.* The language k-CNF is thus a fairly expressive query language in general, for which off-line preprocessing is a useful procedure.

We remark that Proposition 4.1 can be extended somewhat beyond the purely propositional case. In particular, it applies to universal formulas of the predicate calculus that have the form $\forall x_1, \ldots, x_n \ \varphi(x_1, \ldots, x_n)$, where φ may contain function symbols and relations, but is syntactically of the form of a k-CNF formula. (Here we allow as literals not only primitive propositions, but any term of the predicate calculus.) The result and the proof are the same as in the propositional case. The key point remains having a polynomial basis.

We now consider a concrete class of knowledge bases for which the preprocessing stage for a basis for k-CNF described above can be performed using feasible resources, and can yield considerable amortized savings in the on-line computations. Consider the case in which the knowledge base *KB* consists of a formula in disjunctive normal form (DNF). In this case we can show:

Proposition 4.2: *Testing whether a k-CNF formula φ is entatled by a DNF knowledge base KB is linear in* $|\varphi| \cdot |KB|$.

Notice that considering very large knowledge bases and the need for close to real-time response during the on-line activity, the above proposition points to the fact that the on-line reasoning in this case might still be rather inefficient. However, Theorem 3.1 guarantees that with appropriate off-line processing (before any query arrives) testing whether a k-CNF formula φ is entailed by a DNF knowledge-base is linear in $|\varphi| \cdot log(n)$. This is significantly better than what can be achieved without off-line processing.

The results presented so far illustrate the fact that off-line reasoning can greatly improve the on-line performance of useful AI applications involving knowledge bases. However, a designer that decides to use such off-line reasoning must be careful. A possible drawback of such reasoning might appear when we consider knowledge bases that need to be updated frequently. In such cases the contribution of off-line processing depends on the amount of updates and on the cost of updating the preprocessing performed earlier. Suppose that we have two separate knowledge bases KB_1 and KB_2 that use the same language, that the query language for both of them is k-CNF, and that appropriate off-line reasoning was performed for each knowledge base separately. If we want to combine these knowledge bases, there is no general way for combining the respective off-line data on which much effort was spent. The designer will have to investigate whether it is worthwhile to compute all the off-line queries again, or whether in the specific case it is relatively easy to combine the off-line results. We now show a particular form of systems where the above problem can be handled efficiently.

One motivation for discussing knowledge bases that consist of propositions and not of specific models is the need to represent uncertainty (this issue is thoroughly discussed in [Lev89]). Therefore, it is often reasonable to consider knowledge bases in contexts where updates *increase* the degree of uncertainty in the knowledge base. For example, a knowledge base might contain a hypothesis about the relationships between $x_1, \ldots, x_n$, and there might be another knowledge base that represents another alternative for these relationships. Combining these alternatives corresponds to taking a disjunction between propositional formulas. In such cases the appropriate off-line computations can be easily combined. If two scientists worked on different knowledge bases (hypotheses) using off-line computations, and would like to combine their hypotheses (to see what is entailed if it might be the case that only one of the hypotheses is true), then they can combine their off-line computations easily in order to answer the on-line queries efficiently. Formally, this can be formulated as follows:

Proposition 4.3: *Let KB_1 and KB_2 be knowledge bases, let QL be a query language, and let $QL' \subset QL$ be an efficient basis for QL. Finally, let n satisfy $(KB_1 \vee KB_2) \subset \mathcal{L}_n$. Then computing the relevant off-line data for QL with respect to $(KB_1 \vee KB_2)$ given the data with respect to KB_1 and the data with respect to KB_2 can be done in time linear in $|QL'_n|$.*

5 Social Laws as Off-line Design

The notion of artificial social systems has been suggested as a paradigm for the design of shared multi-agent environments. Essentially, when a number of loosely-coupled agents are to function in a shared environment, care must be taken to ensure that the agents do not interfere with one another. Conflicts should be resolved, or better yet to be avoided whenever possible. As suggested in [MT90] and [MT91], an effective way for agents to usefully coexist in a shared environment is by having them obey certain general rules, and allowing them to act independently in the context of these rules. We think of these rules as conventions or a social law.

The design of a social law can be thought of as an instance of off-line preprocessing whose role is to improve the agents' ability to better attain their goals on-line. For example, we now consider the case in which agents are modelled by finite state automata, each able to perform a certain set of actions.

A system of *dependent automata* (DA) is a tuple $\langle N, \{M_i\}_{i \in N}, T \rangle$ where $N = \{1, \ldots, n\}$ is a set of n agents where each agent i is represented by a non-deterministic finite-state machine M_i, and T is a state transition function for the system. Each M_i may be in one of a finite number of different (physical) local states from a set S_i (we assume that the S_i's are disjoint). A tuple of states $(s_1, \ldots, s_n)$, where $s_i \in S_i$ for all i, is called a *configuration* of the system. We denote the set of system configurations by C. We assume that at any point in time the system is in a particular configuration. At every step, agent i performs an action taken from a set A of possible actions (notice that the agent can still *choose* which action it will perform in a given state, since this is not necessarily determined by its state). The set of possible actions an agent can take is in general a function of the local state the agent is in. A tuple of actions $(a_1, \ldots, a_n)$ consisting of the actions the different agents perform at a given point (where agent i is assumed to execute a_i) constitutes the agents' *joint action* there. The next state of every agent is a function of the system's current configuration and the joint action performed by the agents. Formally this is captured by the transition function $T : C \times A^n \rightarrow C$. At any given point, a goal for an agent is identified with one of its states. We assume that an agent can perform computations to plan how to attain its goal, and to determine what actions to take at any given point.

In such a model, the success of one agent's actions may depend in a crucial way on the actions the other agents take. Many of the issues that arise in complex multi-agent systems can already be represented in DA systems.

An agent's plan that guarantees the attainment of a particular goal in a DA system amounts to a strategy by which, regardless of what the other agents do, our agent, will attain its goal. Computing such plans can be rather complex. Moreover, a plan that needs to be able to respond to any possible behavior by the other agents may be very inefficient in the number of steps it takes. Indeed, such a plan may often fail to exist! A DA system is said to be *social* if it is computationally feasible for an agent to devise, on-line, efficient plans that, guarantee to attain any of its possible goals. From the point of view of artificial social systems, a number of computational questions are natural at this stage. These computational problems relate to finding a set of restrictions (called the social law) on the actions performed by different agents at different states of the original DA. These restrictions will be determined off-line before the initiation of activity and will induce a system where agents are able to (efficiently) achieve their goals during the on-line activity. For example, given a DA system *S* we may be interested in restricting the agents' actions by a social law Σ to yield a system S^Σ so that either:

1. in S^Σ every agent has a plan to achieve each of its goals;
2. in S^Σ every agent has an efficient plan to achieve each of its goals;
3. the problem of computing plans in S^Σ is tractable; or
4. the system S^Σ is social.

Various assumptions about the structure of the DA system, for example regarding the number of local states agents have, or the number of actions an agent can perform in every state, may affect the abovementioned computational problems. These and similar problems will apply to more complex types of systems as well. We now turn to study a particular problem in the context of DA systems. Let us call a plan *simple* if it consists of a short sequence of actions, where by short we mean that its length is at most polynomial (in the size of the DA system).[1] We say that a social law Σ for a DA system *S* is *enabling* if in the system S^Σ resulting from the application of the social law Σ in *S,* every agent has a simple plan for attaining each of its goals from each of its local states. We can now show:

[1] Our results hold also when we assume that a simple plan is a tree of polynomial size where the nodes correspond to tests on the current state and the edges correspond to appropriate actions.

Proposition 5.1: *Let $n \geq 2$ be a constant. Given a DA system S with n agents, the problem of finding an enabling social law for such a system, if one exists, is NP-complete.*

Notice that although the number of agents is constant in Proposition 5.1, the size of the system (or, rather its representation) may vary. It depends on the number of local states the agents can be in, the number of actions they can perform, and on the number of possible transitions in the system. These parameters may vary considerably even in the case of just two agents.

Proposition 5.1 answers a question of the second type in the above list. While NP-completeness results are usually interpreted as evidence that a problem is hard, we interpret this result in a slightly different manner here. Since social laws will be found off-line before the initiation of activity, we can spend much time in determining a social law. The proof of the proposition shows that the design process of an enabling social law can be supported by an efficient verification of whether we found an enabling social law or where we failed. Thus, in generic cases, it, should be feasible to (off-line) construct an enabling social law incrementally.

A further restriction can provide us with answers to questions of the third and fourth type in the above list. Given a DA system, we say that a pair (.s,/) of local states for an agent are *neighbors* if there exist two configurations c_s, c_t of the system and a joint action *a* such that (i) in c_s the agent is in state *s* and in c_t it is in /, and (ii) performing *a* in c_s yields the state c_t. Given a DA system ,9, a social law Σ for this system is called *determinizing if* for every pair of neighbor states of the agent, in the resulting system S^Σ the agent has a simple plan for getting from *s* to *t.* Specifically, it has a sequence consisting of a constant number of actions which, when started in s is guaranteed to reach t. Notice that a determinizing social law is necessarily an enabling social law, but not vice-versa. A determinizing social law solves the third computational problem stated above, and since enabling social laws solve the second, we obtain that a determinizing social law yields a social system (and hence solves the fourth computational problem). We have:

Proposition 5.2: *Let $n \geq 2$ be a constant. Given a DA system S with n agents, the problem of finding a determinizing social law for such a system, if one exists, is NP-complete.*

As before, the fact the problem is NP-complete can be interpreted here as relatively good news. Given a candidate for a determinizing social law we will be able to efficiently verify whether it is appropriate or where it fails. Thus, in generic cases, it should be feasible to (off-line) construct a determinizing social law incrementally. We remark that if the local states of the agents have a sufficiently simple form, then the problem of finding a determinizing social law for a system of a constant number of agents can be solved in polynomial time (see [Ten91]).

A particular case study of the design of a social law is presented in [ST92]. There, Shoham and Tennenholtz investigate traffic laws for mobile robots that operate on an n by n grid. They present nontrivial laws that allow the robots to carry out respective tasks without collision at a rate that is within a constant of the rate it would take each of them if it had the whole space to itself. This is an example of how appropriate off-line design of social laws guarantees very effective on-line behavior.

6 Conclusions

This paper suggests the use of off-line processing before the initiation of a system in order to improve the on-line behavior of artificial systems. We investigated this approach in the framework of entailment of queries by knowledge bases, and in the context of multi-agent activity. We presented the notion of an efficient basis for a query language, and showed that off-line preprocessing can be very effective for query languages that have an efficient basis. We also showed that the language of k-CNF formulas, a useful and rather expressive language, has an efficient basis. The second context in which off-line processing is very helpful is multi-agent activity. Finding an efficient and computationally tractable social law is the appropriate form of off-line processing in this case. The social law is a set of restrictions on agents[1] activities that enables agents to tend to work individually but in a mutually compatible manner. This social law needs to be designed carefully and the design process might be a relatively long trial and error procedure, but when we arrive at the appropriate social law then the on-line activity becomes effective and efficient. We formulated the approach in the framework of dependent automata. In that framework, testing whether a given social law is appropriate or where it fails can be efficiently computed. This can greatly speed up the process of generating an appropriate social law.

References

[AC87] P. Agre and D. Chapman. Pengi: An Implementation of a Theory of Activity. In Proc. of AAAI-87, pages 268 272, 1987.

[Agr91] P. Agre. The Dynamic Structure of Everyday Life. Cambridge University Press, Cambridge, UK, 1991.

[BG88] A. H. Bond and L. Gasser. Readings in Distributed Artificial Intelligence. Ablex Publishing Corporation, 1988.

[DS83] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. Artificial Intelligence, 20(1):63 109, 1983.

[HV91] J. Y. Halpern and M. Y. Vardi. Model checking vs. theorem proving: a manifesto. In Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, pages 325-334, 1991.

[Kow74] R. Kowalski. Predicate logic as a programming language. In IFIP Conference, Stockholm, pages 569 574, 1974.

[Lev89] H. J. Levesque. Logic and the Complexity of Reasoning. Technical Report KRR-TR-89-2, University of Toronto, 1989.

[MH81] J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In E. L. Webber and N. J. Nilsson, editors, Readings in Artificial Intelligence. Tioga Publishing Company, 1981.

[MT90] Y. Moses and M. Tennenholtz. Artificial Social Systems Part I: Basic Principles. Technical Report CS90-12, Weizmann Institute, 1990.

[MT91] Y. Moses and M. Tennenholtz. On Formal Aspects of Artificial Social Systems. Technical Report CS91-01, Weizmann Institute, 1991.

[Qui67] M. Quillian. Word concepts: a theory and simulation of some basic semantic capabilities. Behav. Sci., 12:410 430, 1967.

[RG85] ,1. S. Rosenschein and M. R. Genesereth. Deals Among Rational Agents. In Proc. 9th International Joint Conference on Artificial Intelligence, pages 91-99, 1985.

[Sch87] M.J. Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. In Proc. of AAAI-87, pages 1039-1046, 1987.

[Sel88] T.K Sellis. Multiple-query optimization. ACM Transactions on Database Systems, 13(1):23—52, 1988.

[ST92] Y. Shoham and M. Tennenholtz. On Traffic Laws for Mobile Robots. Proc. of AIPS-92, 1992.

[Ten91] M. Tennenholtz. Efficient Representation and Reasoning in Multi-Agent Systems. PhD thesis, Weizmann Institute, Israel, 1991.

[TM89] M. Tennenholtz and Yoram Moses. On Cooperation in a Multi-Entity Model. In Proc. 11th International Joint Conference on Artificial Intelligence, 1989.