Distributed Truth Maintenance

David Murray Bridgeland and Michael N. Huhns
Microelectronics and Computer Technology Corporation
Artificial Intelligence Laboratory
3500 West Balcones Center Drive
Austin, TX 78759-6509
bridgeland@mcc.com

Abstract

In this paper we define the concept of logical consistency of belief among a group of computational agents that are able to reason nonmonotonically. We then provide an algorithm for truth maintenance that guarantees local consistency for each agent and global consistency for data shared by the agents. Furthermore, we show the algorithm to be complete, in the sense that if a consistent state exists, the algorithm will either find it or report failure. The algorithm has been implemented in the *RAD* distributed expert system shell.

Introduction

Two trends have recently become apparent out of the widespread use of knowledge-based systems: 1) systems are being developed for larger and more complicated domains, and 2) there are attempts to use several small systems in concert when their application domains overlap. Both of these trends argue for knowledge-based systems to be developed in a distributed fashion, where modules are constructed to interact productively. The individual modules then are characteristic of intelligent agents. The interconnected agents can cooperate in solving problems, share expertise, work in parallel on common problems, be developed modularly, be fault tolerant through redundancy, represent multiple viewpoints and the knowledge of multiple experts, and be reusable. Additional motivations are presented in (Huhns, 1987) and (Gasser and Huhns, 1989). But in order for these agents to coordinate their activities and cooperate in solving mutual problems, it is essential that they be able to communicate with each other. Further, in order for them to interact intelligently and efficiently, we believe that the agents must be able to assess and maintain the integrity of the communicated information, as well as of their own knowledge.

Knowledge Base Integrity

There are many desirable properties for the knowledge base of an expert system or agent, such as completeness, conciseness, accuracy, and efficiency. For an agent that can reason nonmonotonically, there are additional properties used to describe the *integrity* of the agent's knowledge base: stability, well-foundedness, and logical consistency. A *stable* state of a knowledge base is one in which 1) each knowledge base element that has a valid justification is believed, and 2) each knowledge base element that lacks a valid justification is disbelieved. A *well-founded* knowledge base permits no set of its beliefs to be mutually dependent. A *logically-consistent* knowledge base is one that is stable at the time that consistency is determined and in which no logical contradiction exists. Depending on how beliefs, justifications, and data are represented, a consistent knowledge base may be one in which no datum is both believed and disbelieved (or neither), or in which no datum and its negation are both believed. These concepts are often extended to include other types of contradictions.

In addition, any algorithm that attempts to maintain well-founded stable states of a knowledge base, such as one of the many algorithms for truth maintenance (Doyle, 1979; de Kleer, 1986; Martins and Shapiro, 1988; McAllester, 1980; Russinoff, 1985), should be *complete*, in the sense that if a well-founded stable state exists, the algorithm will either find it or report failure. In general, we desire each agent in a multiagent environment to have a complete algorithm for maintaining the integrity of its own knowledge base.

However, the above definitions of properties for a single knowledge base are insufficient to characterize the multiple knowledge bases in such a multiagent environment. When agents that are nonmonotonic reasoners exchange beliefs and then make inferences based on the exchanged beliefs, then new concepts of knowledge-base integrity are needed. In addition, the relevant concept of global truth maintenance becomes especially problematic if agents must compute their beliefs locally, based on beliefs communicated and justified externally. The next sections extend the above definitions to the multiagent case.

The JTMS

We presume that each agent has a problem-solving component, separate from its knowledge base, that makes

inferences and supplies the results to the knowledge base. Our discussion applies to the set of beliefs that are held and maintained in this knowledge base. In particular, we focus on the systems for maintaining beliefs known as truth-maintenance systems (TMS) (Doyle, 1979).

TMSs are a common way to achieve knowledge base integrity in a single agent system, because they deal with the frame problem, they deal with atomicity, and they lead to efficient search. Furthermore, the justification networks they create can be used for nonmonotonic reasoning, problem-solving explanations to a user, explanation-based learning, and multiagent negotiation. Our research is based on a justification-based TMS, in which every datum has a set of justifications and an associated status of IN (believed) or OUT (disbelieved).

In the example considered below, an initial state of a distributed knowledge base is given and presumed consistent. Our goal is to construct a consistent extension of this state or determine that no such extension exists. The distributed TMS (DTMS) algorithm presented for this task is most often invoked to restore consistency when a consistent state is disrupted by altering the justification for a datum.

Consistent Beliefs among Agents

Consider a network of many agents, each with a partially-independent system of beliefs. The agents interact by exchanging data, either unsolicited or in response to a query. Each agent has two kinds of data in its knowledge base:

Shared Data Beliefs that the agent has shared with another agent sometime in the past.

Private Data Beliefs that the agent has never shared with another agent.

A private datum might become a shared datum by being told to another agent, or by being the answer to some other agent's query. Once shared with other agents, a datum can never again be private. Each shared datum is shared by a subset of the agents in the network—precisely those that have either sent or received assertions about the datum.

We extend the concept of knowledge-base consistency stated above by defining four degrees of consistency and well-foundedness that are possible in a multiagent system.

Inconsistency: one or more agents are individually inconsistent, i.e., at least one agent has a private datum without a valid justification and labeled IN, or a private datum with a valid justification and labeled OUT.

Local Consistency: each agent is locally consistent, i.e., no private OUT datum has a valid justification, and each private IN datum has a valid justification. However, there may be global inconsistency among agents: there may be a shared datum that one agent believes to be IN and another believes to be OUT.

Local-and-Shared Consistency: each agent is locally consistent and each agent is mutually consistent about any data shared with another agent, i.e., each shared datum is either IN in all the agents that share it or OUT in those agents. There is, however, no global consistency.

Global Consistency: the agents are both individually and mutually consistent, i.e., their beliefs could be merged into one large knowledge base without the status of any datum necessarily changing.

In the absence of interagent communication, and presuming the local environment of each agent is consistent, then Local Consistency should hold. The introduction of interagent communication, however, tends to drive the system towards Inconsistency, because the agents might receive data that conflict with their current beliefs. The mechanism for truth maintenance we describe below enables each agent then to strive for Local-and-Shared Consistency. The presumption here is that the shared data are the most important, because they affect the problem solving of another agent, and so special effort should be made to maintain their consistency.

Although our goal is to maintain Local-and-Shared Consistency, we at times allow the system to fall short of this goal in order to permit agents to have different viewpoints. In this case, one agent may hold a belief that is known to be contrary to the belief of a second agent. The agents do not attempt to resolve this dispute if resolution would result in their being individually inconsistent. A consequence of this is that these agents should then not believe any data originating from each other, unless that agent can prove that its belief for that data is independent of the disputed data.

Ill-Foundedness: individual agents have beliefs that are internally ill-founded.

Local Well-Foundedness: individual agents have beliefs that are internally well-founded; however, there may be shared data that are IN but have no valid justifications in any agent.

Local-and-Shared Well-Foundedness: individual agents have beliefs that are internally well-founded, and every IN shared datum has a valid justification in some agent; however, there may be ill-founded circularities of beliefs among groups of agents.

Global Well-Foundedness: every datum has a globally valid justification and no set of data, whether local to an agent or distributed among a group of agents, is mutually dependent.

A Multiagent TMS

In the classical TMS, a datum can be either IN or OUT. For the DTMS, we refine the IN status to two substa-

tuses: INTERNAL and EXTERNAL. An INTERNAL datum is one that is believed to be true, and that has a valid justification. An EXTERNAL datum is believed to be true, but need not have a valid justification. Intuitively, the justification of an EXTERNAL datum is "so-and-so told me." Hence, only a shared datum can be EXTERNAL. For Local-and-Shared Well-Foundedness, a shared datum must be INTERNAL to at least one of the agents that shares it and either INTERNAL or EXTERNAL to the rest of the agents.

In the only complete justification-based TMS labeling algorithm (Russinoff, 1985), Russinoff takes a generate and test approach, first unlabeling a collection of data, then attempting to relabel that collection. On failure to relabel, a superset of the last unlabeled collection is unlabeled. We take a similar approach in the DTMS. Since new data in one agent can change not only the status of that agent's beliefs, but also that of other agents, our unlabeling and subsequent labeling will sometimes involve multiple agents.

The support status of a shared datum is jointly maintained by several agents. Hence, a single agent is generally not free to change the status of a shared datum on its own accord. It must coordinate with the other agents so that they are all consistent on the status of the datum. Central to the DTMS then is the single agent operation of **label-wrt**. **label-wrt** is a variation of classical TMS labeling in which the statuses of some data—though unlabeled—are fixed by external requirements.

More precisely, **label-wrt** is given a network of data. Some of the data have statuses of IN, OUT, INTERNAL, or EXTERNAL. Other data are unlabeled. For each shared datum, there is a *desired* label of either OUT, INTERNAL, or EXTERNAL. **label-wrt** either finds a consistent well-founded labeling of the network that satisfies the shared data requirements, or it reports failure. Space prohibits a presentation of an algorithm to implement **label-wrt**. Our approach is a variation of Russinoff's well-founded and complete labeling algorithm (Russinoff, 1985).

Algorithm Schema

The DTMS is a double generate and test. Relabeling is invoked by the addition or removal of a justification. When invoked, the DTMS does the following three things:

1. Unlabel some data, including the newly justified datum and presumably its consequences. This unlabeled data set might be confined to a single agent or it might span several agents. If a shared datum is unlabeled in some agent, it must be unlabeled in all the agents that share it.
2. Choose labelings for all the unlabeled shared data, as defined above.
3. Label each of the affected agents with respect to the requirements imposed by the shared data, invoking **label-wrt** as described above. If any of the affected agents fails to label, then backtrack. Either choose different labelings for the shared data (step 2), or unlabel a different set of data (step 1).

This schema will be refined later, but some nice properties emerge at this abstract level:

- Any labeling found by the DTMS will have Local-and-Shared Consistency and Well-Foundedness.
- If the two generate steps are exhaustive, the DTMS is complete: it will find a labeling should one exist.

Note that these properties are true both of the DTMS algorithm described in this paper, and any other algorithm that conforms to this schema.

Unlabeling

When the DTMS algorithm is invoked, it starts by unlabeling a collection of data. This collection may be confined to a single agent or it may span many agents. However, it must meet the following constraints:

1. It must include the datum that originally acquired the new justification.
2. A shared datum that is unlabeled in one agent must be unlabeled in all the agents that share it.
3. On failure to label the collection, it must generate a new collection of unlabeled data. To guarantee completeness, the generation must be exhaustive: it must eventually generate a collection sufficiently large that failure to label it means the whole network cannot be labeled.

Using only these constraints, unlabeling is underconstrained: many algorithms satisfy. For example, on any status change one could unlabel all data in all agents. This global unlabeling satisfies all the constraints and is also quite simple, but also is too inefficient for practical use. The global unlabeling does reveal two DTMS principles that motivate the more complex algorithm presented later:

Principle 1 *Belief changes should be resolved with as few agents as possible.*

Principle 2 *Belief changes should be resolved by changing as few beliefs as possible.*

Most belief changes can be resolved by changing things only "downstream" of the new justification, i.e., those data that directly or indirectly depend on the datum newly justified. It is sometimes necessary to move "upstream" as well, and relabel data that directly or indirectly support the status of the newly justified datum. Consider the knowledge base for a single agent shown in Figure 1 (Russinoff, 1985). Here, datum Q acquires the new justification shown in dotted lines. If only P and Q are reassigned, the system is forced to report an unsatisfiable circularity. In order to restore stability, the status of the data upstream from P must

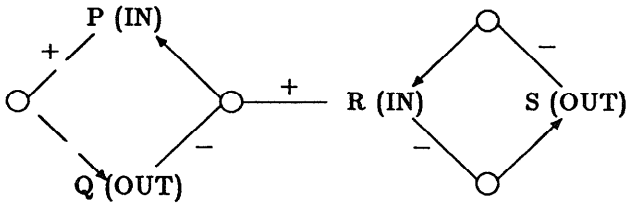

Figure 1: Relabeling upstream data to resolve an odd loop

be changed: if the system makes S OUT and R IN, both P and Q can be OUT.

Principle 3 *Belief changes should be resolved downstream if possible; upstream relabeling should be minimized.*

The above principles motivate the algorithm **unlabel**. It attempts to minimize both the involvement of other agents and the unlabeling of upstream data, but prefers the former to the latter. **Unlabel** is invoked on a list containing either the newly justified datum, when **unlabel** is first invoked, or the unlabeled data that could not be labeled on a previous invocation. **Unlabel** attempts to find yet to be unlabeled private data downstream of those already labeled. If there are none, it looks for shared data downstream, unlabeling those in all the agents that share them, and also unlabeling private data downstream of the shared data. Finally, if there are no shared data downstream that are yet to be unlabeled, it unlabels data just upstream of all the downstream data, and all private data downstream of that. If there is nothing yet to be unlabeled upstream, **unlabel** fails and, in fact, the data admit no Local-and-Shared Consistent and Well-Founded labeling.

Consider the DTMS network in Figure 2. There are two agents, Agent 1 and Agent 2, and they share the datum T. As in Figure 1, the initial labeling shown in the diagram is perturbed by the addition of the new dotted justification. Agent 1 initially unlabels just the changed datum and private data downstream, P and Q, but there is no consistent relabeling. Hence, Agent 1 unlabels all shared data downstream of P and Q, and all private data downstream from there: P, Q, both Ts, and U. Again labeling fails. Since there is no further shared data downstream, Agent 1 and Agent 2 unlabel upstream and privately downstream from there: P, Q, Ts, U, R, and S. Now labeling succeeds (with S and U IN and everything else OUT). Had labeling failed, **Unlabel** would not be able to unlabel more data, and would report that the network is inconsistent.

Distributed System Issues To be implemented, the **unlabel** algorithm needs to be distributed. This is straightforward if each agent keeps track of which data are currently unlabeled and reports to other agents only whether yet to be unlabeled data became unlabeled. Upstream and downstream messages mention

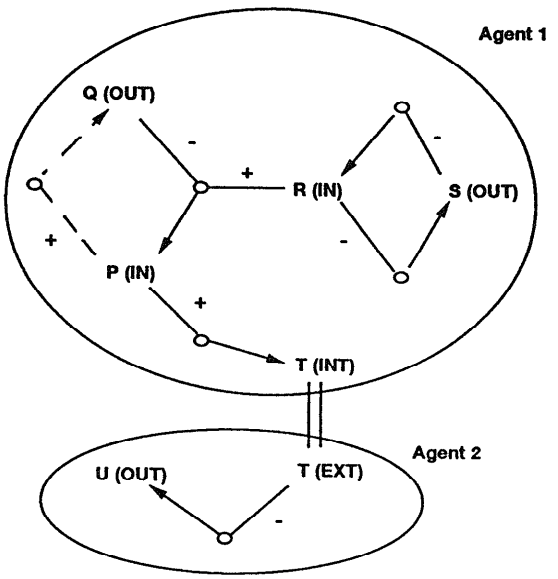

Figure 2: A DTMS network before relabeling

only which shared datum is affected, and the corresponding acknowledgements report only whether new data were unlabeled.

When a group of agents are labeling, their beliefs are in a state not suitable for reasoning. Hence, queries from other agents must be queued until labeling is complete. However, if two agents share a datum, and are involved in separate unlabeling tasks (i.e., initiated by different changes to beliefs), deadlock could occur. Fortunately, the separate unlabeling tasks can be combined into a single one, with a corresponding combination of the subsequent labeling.

Labeling

Once an area of data is unlabeled, the DTMS must pick candidate labels for the shared data, such that each datum is either OUT in all the agents that share it, or INTERNAL in at least one agent and INTERNAL or EXTERNAL in the rest. Any exhaustive means of picking will guarantee completeness. The following Prolog code shows one means:

```
label-shared([], Shared-labels, Shared-labels).

label-shared([Agent|Agents], So-far, Final) <--
  label-one-shared(Agent, So-far, New),
  label-shared(Agents, New, Final).
```

label-shared relates its first argument—a list of agents—and its third argument—a list of shared data and their labels. The second argument is used to pass commitments about labels for shared data to recursive calls to **label-shared**. The relation calls **label-one-shared**, attempting to assign labels to a single agent

that are consistent with those already assigned to others. If it finds such an assignment, it recursively attempts to find labels for the other agents. On failure, it bactracks and looks for alternatives to the previous single-agent labeling.

This algorithm could be implemented on a fully connected multiagent system by having each agent responsible for generating labels that are consistent with others already generated, as in **label-one-shared**, and implementing the recursive nature of **label-shared** with a message passed depth-first from agent to agent. This message needs to contain a list of agents already visited, so that none are revisited, and a list of the already labeled shared data. Also, before an agent passes a message to another, it needs to record its state for future backtracking. The shared-data labeling fits into the larger labeling process as follows:

```
label(Agents) <--
  label-shared(Agents, [], Shared),
  label-private(Agents, Shared).

label-private([],Shared).

label-private([Agent|Agents],Shared) <--
  local-shared(Agent, Shared, Local-Shared),
  label-wrt(Agent, Local-Shared),
  label-private(Agents, Shared).
```

The private labeling follows the shared data labeling. The private data are labeled one agent at a time. First the relation **local-Shared** extracts the shared labels relevant to a single agent from the list of all the shared labels. Then **label-wrt** attempts to label the private data consistently with the shared data. Any failure to label causes backtracking. This algorithm will find a Local-and-Shared Consistent and Well-Founded labeling of an unlabeled area in a collection of agents.

Unfortunately, this algorithm has poor computational performance. If there is no consistent labeling of the agents, the DTMS will generate all shared data labelings and attempt to label each privately. The performance can be improved by interleaving the labeling of the shared data and the private data. Failure in a single agent to find a private labeling consistent with the labels of the shared data will then cause earlier backtracking:

```
label(Agents) <-- label-internal(Agents, []).

label-internal([], Shared).

label-internal([Agent|Agents], So-far) <--
  label-one-shared(Agent, So-far, New),
  local-shared(Agent, New, Local-shared),
  label-wrt(Agent, Local-shared),
  label-internal(Agents, New).
```

Label-internal could be implemented by a message passed depth-first from agent to agent. This message needs to contain a list of the agents visited so far, and a record of the labels given so far to the shared data.

Consider again Figure 2. R, S, P, Q, U, and both Ts have now been unlabeled. Agent 1 chooses labels for T and attempts to label his private data in a consistent manner. If Agent 1 chooses INTERNAL as T's label, he finds there is no labeling of his private data to make T internally justified. A next attempt with EXTERNAL is consistent (with S IN and everything else OUT), and Agent 1 passes his label of T to Agent 2. Agent 2 must then label T INTERNAL, but finds there is no way to label U. Agent 2 then backtracks and Agent 1 tries a final attempt to label T, this time as OUT. This succeeds with S IN and everything else OUT, and Agent 2 can also label T OUT by labeling U IN.

Optimizations

This DTMS algorithm admits several local optimizations:

1. An agent can forgo the labeling of its unlabeled shared data by **label-one-shared** and instead label everything that is unlabeled with **label-wrt**. This requires a more sophisticated **label-wrt** that can generate INTERNAL and EXTERNAL labels for the shared data, as well as IN and OUT labels for the private data.
2. An agent can keep a record of label attempts, caching for each combination of shared data labels whether it succeeded or failed to find a private labeling. A call to **label-wrt** will first consult the cache, thus avoiding redundant work. Ordering the shared data and then indexing the combinations in a discrimination net seems to be a good implementation for this cache.
3. In the above algorithm, only one agent is active at a time. However, there is something productive that the other agents can do: fill in their label caches by attempting to find private labelings for shared data combinations not yet examined. In fact, this effort could be guided by other agents. If agent 1 shares data with agents 2, 3, and 4, when agent 1 passes a label-internal message to agent 2, it could advise agents 3 and 4 about its decisions on the labels of shared data. Then other agents could work only on that portion of their caches that are consistent with agent 1's decision.

Discussion

There have been many other attempts to develop systems of cooperating agents or knowledge sources. Early attempts, based on the blackboard model, involved agents with independent knowledge bases. The independence was achieved by restricting agent interactions to modifications of a global data structure—a blackboard—and by minimizing overlap in the agents'

knowledge. Later systems allowed richer agent interactions and overlapping knowledge, but the agents were required to have consistent knowledge and to reason monotonically. This led to representational problems, because different experts in the same domain often have different perspectives and *conflicting* knowledge, making it difficult to construct a coherent problem-solving system for that domain. Earlier solutions were to allow inconsistent knowledge bases; this enabled the conflicting knowledge to be represented, but it did not confront the problem of how to resolve the conflicts.

Other researchers have explored negotiation as a means to mediate among conflicting agents. These systems have involved either monotonic reasoners, such as (Sycara, 1989), or nonmonotonic, but memoryless, reasoners, such as (Zlotkin and Rosenschein, 1989), i.e., reasoners that simply discard old solutions and re-solve in the face of conflicts.

Another approach is to consider the research efforts in multiple-context truth-maintenance systems (de Kleer, 1986; Martins and Shapiro, 1988) from a distributed viewpoint. These systems manipulate belief spaces, or contexts, in order to remove inconsistent ones. One might imagine each belief space represented by a different agent, who then maintains it. However, in this model, the belief spaces themselves do not interact and, in fact, the belief revision system treats each space separately.

A notable exception to this is the work of (Mason and Johnson, 1989), who developed a distributed assumption-based TMS. In this system, agents interact by exchanging data, with their associated assumption sets, and NOGOODS, i.e., bad assumption sets. The agents maintain separate belief spaces and may disagree about an exchanged datum. The agents therefore have Local-and-Shared Well-Foundedness, but only Local Consistency.

The system presented herein, although an improvement in that it achieves Local-and-Shared Consistency, nevertheless suffers from several deficiencies:

- First, by not supporting some form of explicit negation or reasons for disbelief in a datum, the system allows an agent with less information to dominate an agent with more. For example, if two agents each have an identical justification for belief in a shared datum, and one agent learns a fact that invalidates its justification, the other agent's still-valid justification will be sufficient for both to continue believing in the datum.
- Second, our algorithm can involve significant computational overhead if the agents have shared a large amount of data, if the data have many connections to the rest of the agents' belief networks, and if the status of their beliefs changes frequently.
- Third, we believe unsatisfiable circularities are more likely in a distributed system.

We are currently investigating the likelihood and severity of these deficiencies in real-world application domains. We are also developing a mechanism for negotiation that uses the beliefs supplied by our DTMS.

The above algorithm has been implemented in the *RAD* distributed expert system shell, which includes a framework within which computational agents can be integrated. RAD is a first step toward cooperative distributed problem solving among multiple agents. It provides the low-level communication and reasoning primitives necessary for beneficial agent interactions, but it does not yet guarantee successful and efficient cooperation. The next steps will require increased intelligence and capabilities for each agent, resulting in more sophisticated agent interactions occurring at a higher level.

References

Johan de Kleer. An Assumption-Based TMS, Extending the ATMS, and Problem Solving with the ATMS. *Artificial Intelligence*, 28(2):127–224, March 1986.

Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12(3):231–272, 1979.

Les Gasser and Michael N. Huhns. *Distributed Artificial Intelligence, Volume II.* Pitman Publishing, London, 1989.

Michael N. Huhns. *Distributed Artificial Intelligence.* Pitman Publishing, London, 1987.

Joao P. Martins and Stuart C. Shapiro. A Model for Belief Revision. *Artificial Intelligence*, 35(1):25–79, May 1988.

Cindy L. Mason and R. R. Johnson. Datms: A Framework for Distributed Assumption Based Reasoning. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 293–317. Pitman Publishing, London, 1989.

David A. McAllester. An Outlook on Truth Maintenance. AI Memo No. 551, Artificial Intelligence Laboratory, MIT, Cambridge, MA, August 1980.

David M. Russinoff. An Algorithm for Truth Maintenance. MCC Technical Report No. ACA-AI-062-85, Microelectronics and Computer Technology Corporation, Austin, TX, April 1985.

Katia Sycara. Multiagent Compromise Via Negotiation. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 119–137. Pitman Publishing, London, 1989.

Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and Task Sharing among Autonomous Agents in Cooperative Domains. In *Proceedings IJCAI-89*, pages 912–917, Detroit, MI, August 1989.