Proactive Information Exchange during Team Cooperation

Yu Zhang* Richard A. Volz* Thomas R. Ioerger* Sen Cao* John Yen†

*Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
Phone: 01-979-845-8873
Fax: 01-979-862-4813
{yuzhang, volz, ioerger, sencao}@cs.tamu.edu

†School of Information Science and Technology
The Pennsylvania State University
University Park, PA 16802
Phone: 01-814-865-6174
Fax: 01-814-865-6426
jyen@ist.psu.edu

To my Ph.D advisor, Dr. Richard A Volz, who gave me experiences in academic and professional life.

Best,

Yu

In Proceedings of International Conference on Artificial Intelligence 2002, 341-346

Proactive Information Exchange during Team Cooperation

Yu Zhang* Richard A. Volz* Thomas R. Ioerger* Sen Cao* John Yen†

*Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
Phone: 01-979-845-8873
Fax: 01-979-862-4813
{yuzhang, volz, ioerger, sencao}@cs.tamu.edu

†School of Information Science and Technology
The Pennsylvania State University
University Park, PA 16802
Phone: 01-814-865-6174
Fax: 01-814-865-6426
jyen@ist.psu.edu

Abstract We are concerned, ultimately, with multi-agent teams involving both software and human agents in which the software agents assist in training the humans, e.g., as virtual team members. The software agents must be capable of utilizing the teamwork mechanisms used by humans. In this paper, we describe how to give agents the same implicit communication capabilities, i.e., observation, that humans use. We show how agents can use observations of the environment and teammates' actions to estimate other agents' beliefs without bothering them with unnecessary messages; we also show how agents can anticipate information needs among the team and proactively exchange information, reducing the total communication volume. To achieve these goals, we add sensing capabilities to agents and present algorithms that infer teammates' mental state from the observation. Experimental data is presented that shows that the advantage of observation in proactive information exchange as well as enhanced team performance.

Keywords Intelligent agent, Multi-agent system, Teamwork, Agent communication, Observation

Introduction

We are concerned, ultimately, with multi-agent teams with both software and human agents in which the software agents assist in training the humans, e.g., as virtual team members. Our software agents must emulate, as closely as possible, the teamwork mechanisms used by humans so that we can use them for training. In this paper, we focus on giving agents the same implicit communication capabilities, i.e., observation, that humans use. Most proposed teamwork structures (e.g. Cohen and Levesque's joint intentions [1], Jennings' joint responsibilites [2], Huber and Durfee's shared plans [3], and Stone and Veloso's role definitions [4]) rely agents negotiating with each other, typically using a lot of communication. In human teams, however, agents often make decisions based upon knowledge of plans together with observations of the environment and other agents' actions. For example, Tambe and Kaminka [5] use observation to monitor failed social relationships between agents. Here, we discuss reducing communication by using knowledge of what agents can sense.

The context of our work is the CAST (Collaborative Agents for Simulating Teamwork) [6] architecture. In CAST, teamwork knowledge is represented in MALLET (Multi-Agent Logic-based Language for Encoding Teamwork) [8], and action rules (in Horn-clauses) and agent knowledge are represented in JARE (Java Automatic Reasoning Engine) [9], a Prolog-like back-chaining theorem prover. CAST uses an algorithm, DIARG (Dynamic Inter-Agent Rule Generator), for identifying opportunities for deducing communication. DIARG analyzes team member roles to identify information produced and/or needed by each agent. The team state is tracked using a Petri Net (which servers as a first-order approximation of a shared mental model). The information exchange protocol, keeps track of the teamwork status via the PN and uses the relative frequency of information need vs. information production to decide whether to proactively send information or wait for information requests. One difficulty with this approach is that significant status information must be distributed among the agents and there is no attempt to utilize agent sensing capabilities to reduce what information must be sent.

A more realistic approach (from a human perspective), is to give the agents sensing, or observing, capabilities. Agents can infer some aspects of the mental state of other agents by observing the environment. This, together with inferences about what others can see, allows an agent to decide when it does not need to send information to other agents or whom to ask when it needs information in a manner that reduces overall communication.

We first present preliminary contextual information. Then, we introduce a representation of observability and an algorithm for reasoning about it. Next we describe an algorithm of proactive information exchange. Finally, we present some empirical results. Our testbed is multi-agent wumpus world, extending that introduced in Russell and Norvig [10]. We use multiple agents with different capabilities so that teamwork becomes important.

Preliminaries

In this section, we introduce some concepts that will be used in subsequent sections.

Team Knowledge. Team knowledge is encoded in a team knowledge representation language, MALLET. Goals are achieved by assigning a team of agents to a task and then invoking that task. Each task has one or more plans by which it might be accomplished. Plans describe processes. Processes consist of invocations of primitive operations, subplans, or various constructs such as sequential, parallel, conditional, or iteration. The syntax of processes can be defined recursively. For example,

```
(plan killwumpus()
(role carrier ?ca)
(role fighter ?fi (constraint (closest-to-wumpus)))
(process
  (seq (do ?ca (findwumpus))
       (do ?fi (movetowumpus))
       (do ?fi (shootwumpus)))))
```

Time. To simplify our initial investigation, we index time into discrete steps, $\tau = \{t_j\}$, not necessarily evenly spaced, and force a synchrony of actions among the agents. At each time step, each agent performs one step in its plan.

Action. We consider an action to be the execution of an operation. An action Ω is a 2-tuple $<\Phi, \Psi>$, where, Φ is precondition of Ω, and Ψ is effect of Ω. Each precondition and effect is a conjunction of predicates. All predicates in a precondition must be believed before the action can be performed. Conjuncts in the effects are known to be true after an action is performed. Actions may be individual, (performed by only a single team member), or they may be performed jointly by multiple team members. The specification of an action has the following form:

```
(operator <op-name> (<var>*)
  [pre-cond <cond>*]
  [effect <cond>*])
```

Environment. In general, the environment is a function of time τ and the actions performed by the agents. At any time t, the environment is one element of a set $\mathcal{E} = \{e^t\}$ of instantaneous states. Formally, the effect that an agent's action has on environment can be represented using a state transformer function, $\tau : e^t \times \Omega^t \rightarrow e^{t+1}$ (Fagin, et.al,[11]). Thus $\tau(e^t, \Omega^t)$ denotes the environment state that would result from performing action Ω^t in environment state e^t. To simplify things, we use a set of predicates in JARE to represent portions of the state, e.g., (location carrier ?x ?y). A JARE query returns the values of ?x and ?y for which the predicate is true. The portions of the environment observable to an agent are represented by such predicates.

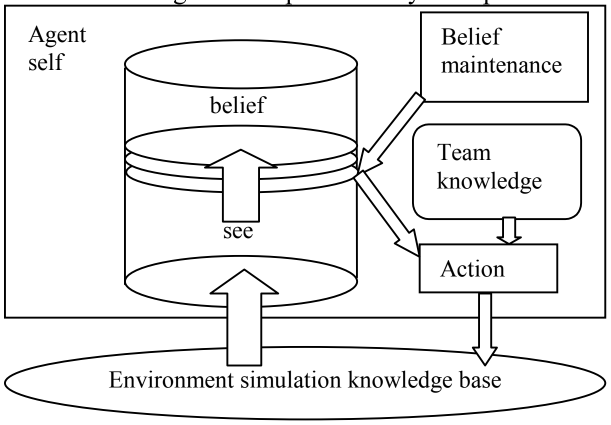

Figure 1 Agent function cycle

Agent. At each time step, each agent has function cycle: observe, belief maintenance, and act (Figure 1). Observation is represented as a set of rules (see next section). At any time t, each agent has a local state, which encapsulates all the information to which the agent has access at time t. The agent's knowledge base is composed of the set of local states at different times. Since the number of steps could be large, we keep only a limited history (see next section). Every agent maintains its knowledge base, which contains the agent's observability and the agent's belief about other agents' observability.

Reasoning about Observability

To define observability, we need to state both *what* is observed and the *conditions* under which it can be seen. The *what* may be any object property (of either an agent or environment object) or an action performed by an agent. Conditions are expressed as predicates. Both may contain parameters, which enables a JARE query to return, e.g., the location of an observed object. The observation process typically has a dependency upon time τ. We handle this by including the time of an observation with the fact that is locally stored. We make the following assumptions:

Seeing is believing. While philosophers may entertain doubts because of the possibility of illusion, common sense is that, other things being equal, one should believe what one sees (Bell and Huang [12]). Thus, we assume that an agent believes an observed predicate persists until it observes another predicate that negates that predicate.

Observation condition. An observation condition, OC, is a conjunctive list of predicates. An agent can observe property p at time t if both p and OC are true at t. An action occurrence is represented as a predicate (including time) in the environment, e.g.. (shootwumpus fi T). An agent can observe an action Ω if Ω and OC are true at the time of the observation. We assume OC includes the constraints of the action See itself as well as the constraints of the things (p, or Ω) to be observed (e.g., preconditions of Ω). We also assume that every agent notices all observable predicates and all predicates are unambiguous.

Negation representation. . It is important to be able to reason about cases it is unknown whether the predicate is true or false. Thus, we need to be able to say that the agent can observe *whether or not* a predicate is true. For each predicate P in environment, we create a new predicate notP for $\neg$P. Everywhere throughout the domain rules, we replace $\neg$P with notP. So, if the theorem prover returns neither P nor notP as true, this means the value is unknown.

Observation

The syntax of observation is thus given as:

```
<observation> ::= ( see <viewing>) |
                  (believe-can-see <believer><viewing>)
<viewing> ::= <observer><observed> <time> <cond>*
<believer> ::=<agent>
<observer> ::= <agent>
<observed> ::= <property | action>
```

```
<property> ::=(<property-name> <object> <var>*)
<action> ::=(<action-name> <doer> <var>*)
<object> ::=<agent> | <non-agent>
<doer> ::=<agent>
```

More formally, we use See(a_s, α, t) to refer the action for agent a_s to observe α at time t, and CanSee(a_s, α, t, c) as the capability for agent a_s to observe α at time t under the observation condition c. The axiom below specifies the occurrence of an agent's actual sensing action:

$$\forall a_s, \alpha, t, c,\ \ CanSee(a_s, \alpha, t, c) \wedge Hold(c, t) \rightarrow See(a_s, \alpha, t)$$

An agent's inference about other agent's observations is based on the following axiom:

$$\forall a_b, a_s, \alpha, t, c, t_b$$
$$Bel(a_b, CanSee(a_s, \alpha, t, c), t_b) \wedge Bel(a_b, c, t_b) \rightarrow Bel (a_b, See(a_s, \alpha, t), t_b)$$

which means at time t_b, the agent a_b believes a_s can observe α at time t under condition c. Finally, the assumption of "seeing is believing" is formalized as the axiom below:

$$\forall a_s, \alpha, t$$
$$See(a_s, \alpha, t) \rightarrow [Hold(\alpha, t) \rightarrow Bel(a_s, \alpha, t)] \wedge [\neg Hold(\alpha, t) \rightarrow Bel(a_s, \neg\alpha, t)]$$

For the convenience of our discussion, we also take believe-can-see(a, b, α, t) as an abbreviation of Bel(a, CanSee(b, α, t, c_a), t). A more detailed discussion about the formalism on agent observability and proactive information delivery can be found in [8].

Observations are presented as rules in JARE. Some example observation rules in carrier's knowledge base might be as shown below. Here, the symbols ca, fi1, fi2 are used for carrier, fighter1, fighter2, respectively:

```
(see ca (hasarrow ?o) ?t
        (location ca ?xs ?ys ?t) (location ?o ?x ?y ?t)
        (= ?xs ?x) (agent ?o) )
(believe-can-see ca fi1 (location ?o ?x ?y) ?t
        (location fi1 ?x1 ?y1 ?t) (location ?o ?x ?y ?t)
        (= ?x1 ?x) )
(believe-can-see ca fi1 (hasarrow ?o) ?t
        (location fi1 ?x1 ?y1 ?t) (location ?o ?x ?y ?t)
        (adjacent ?x ?x1 ?y ?y1) (agent ?o) )
```

The semantics of observation are maintained by the algorithm given in Figure 2. This algorithm builds beliefs in believer's knowledge base by checking the following:

- Observing a property
  1) When the observation (see self (<prop-name> <object> <var>*) <time> <cond>*) is made at time t, a JARE query ((<prop-name> <object> <var>*) t <cond>*) is made of the environment knowledge base. The response is a list of tuples (null if conditions are not satisfied) of variable bindings for which the query is true. Then, self's knowledge base is updated with the fact ((<prop-name> <object> <var>*) t) for each tuple of variable bindings from the returned.
  2) In the case of (believe-can-see self Ag(≠self) <property> T <cond>*), a query is made with respect to KB_{self}. If the condition is satisfied at T, self believes Ag can see the property. However, self may or may not have knowledge of <property>, e.g., a carrier may believe a fighter can smell stench of wumpus if the fighter is adjacent to wumpus, but the carrier doesn't itself smell the stench.

```
/*Let self be the agent invoking the algorithms. We denote the
knowledge base for agent a by $KB_a$, and for the environment by
$KB_{env}$. The updateKB(agent, action, $KB_{self}$) algorithm is
independently executed by each agent, denoted self below, after
the completion of each step in plan in which the agent is involved.
The function update(…) used by updateKB(…) maintains time
history and coherence of KB. */

updateKB(self, action, $KB_{self}$){
    updateEnv(action, self); //update simulation environment
                             // by effects of latest action
    for each conjunct in the effects of action
        update($KB_{self}$, conjunct, T);
    updateSelfObs(self, $KB_{self}$); //by observation at T
    updateSeflBel(self, action, $KB_{self}$);
}
updateSelfObs(self, $KB_{self}$){
    for each rule in $Kb_{self}$ of the form (see self (prop object ?var*) ?t
    cond), if ((prop object ?var*) T cond) is true in $KB_{Env}$ for some
    bindings of variables,
        update($KB_{self}$, (name object ?var*), T)
        for each such binding of values to the variables;
    for each rule in $KB_{self}$ of the form (see self (action ?doer ?var*)
    ?t cond) for which cond is true in $KB_{env}$ for some binding of the
    variables, if the environment KB has been updated with (action
    doer) since the last time self performed its observation update
    (at ?t = T1),
        for each conjunct of precondition of action
            update($KB_{self}$, (conjunct), T1);
        for each conjunct of effects of action
            update($KB_{self}$, (conjunct), T1);
}
updateSelfBel(self, action, $KB_{self}$){
    for each rule of the form (believe-can-see self ?Ag (prop object
    ?var*) T cond) for which cond is true in $KB_{self}$ for some binding
    of arguments to variables (?Ag≠self),
        for each such binding of arguments to the variables
            update($KB_{self}$, (believe-can-see ?Ag (prop object ?var*),
            T);
    for each rule of the form (believe-can-see self ?Ag (action doer
    ?var* ) T cond) for which cond is true in $KB_{self}$ for some
    binding of arguments to variables,
        for each conjunct of the precondition of action
            update($KB_{self}$, (believe-can-infer Ag conjunct), T);
        for each conjunct of the effects of action
            update($KB_{self}$, (believe-can-infer Ag conjunct), T);
}
```

Figure 2 A belief maintenance algorithm

- Observing an action
  1) In the case of (see self <action-name> Agd(≠self) <var>* T <cond>*), the query is made with respect to KB_{env}. Self can make two inferences. First, self infers that the preconditions of the action were true just prior to time T, and that the effects of the action are true at time T. Second, self infers that Agd knew the preconditions just prior to time T and that Agd can infer the effects of the action.
  2) In the case of (believe-can-see self Ag(≠self) <action-name> Agd(≠self) <var>* T <cond>*), the query is evaluated with respect to KB_{self}. Self add's tuples to KB_{self} indicating that Ag can infer that the preconditions of the action were true just prior to time T, and that the effects of the action are true at time T[1]. Similarly, tuples for Ag inferring the effects of the action are added to KB_{self}.

The algorithm begins with updateEnv by self's last action. We don't show how updateEnv works in detail, as that is not the principal point of this paper. Basically, the environment simulation updates the knowledge base KB_{Env} after receiving any action from the agents. Because self can infer the effects of its own actions, the algorithm saves these effects as new knowledge. updateSelfObs evaluates direct property observation rules with information obtained from KB_{Env} and updates KB_{self} with the results of the observation. updateSelfBel updates self's beliefs on what it can infer other by observing agents and their actions.

To avoid having the size of the knowledge base explode, we define a duration, d. Information older than d is deleted, except that the most recent assertion of a fact is kept, even if it is older than the duration. The function update manages this finite history. While at the moment, we only utilize the most recent information added to the knowledge base, we anticipate more complex forms of reasoning in the future, and hence keep this finite history.

Proactive Information Exchange

The purpose of proactive information exchange is to reduce the communication overhead as well as improve the efficiency or performance of a team. We extend two types of information exchanges protocols: activeAsk and proactiveTell [6]. These protocols are used by each agent to generate inter-agent communication. DIARG analyzes operator preconditions and effects and builds an information flow describing potential communication needs. An information flow is defined as a three tuple <info, providers, needers>, where info is the predicate name together with zero or more arguments, providers is a list of agents who might know such information, and needers is a list of agents who might need to know the information. We differentiate information into two types [6]:

- Frequently changing but infrequently needed information. We call these type I1;
- Infrequently changing but frequently needed information. We call these type I2;

For any piece of information I, we define two functions, f_C and f_N. $f_C(I)$ returns the frequency with which I changes. $f_N(I)$ returns the frequency with which I is used by agents. If $f_C(I) > f_N(I)$, then I is I1 type information. If $f_C(I) \leq f_N(I)$, then I is I2 type information.

For activeAsk, an agent requests the information from other agents who may know it (determined from the information flow); for proactiveTell, the agent tells the information to other agents who need. An agent always assumes others know nothing until it can observe or reason that they do know a relevant item. Information sensed and beliefs about others' sensing capabilities becomes the basis for this reasoning. First the agent determines what another agent needs from the information flows. Second, the observation rules are used to determine whether or not one agent knows that another agent can sense the needed information. Figure 3 shows the algorithm.

```
/*Independently executed by each agent when the agent needs the
value for an item of Information, I. */
activeAsk(self, I, $KB_{self}$, T){
candidateList=null;
    if (I is of type I1 and (I at t) v (notI at t) is not true in $KB_{self}$ for
    any t ≤ T) { //self doesn't know whether or not I is true
        If there exists a x ≥ 0 such that
        ((believe-can-see Ag (I) T-x) v (believe-can-see Ag (notI) T-
        x) v (believe-can-infer Ag (I) T-x) v (believe-can-infer Ag
        (notI) T-x)) is true in $KB_{self}$, {
            Let xs be the smallest such value of x;
              for each agent Ag≠self
                  if ((believe-can-see Ag (I) T-xs) v (believe-can-see
                  Ag (notI) T-xs) v (believe-can-infer Ag (I) T-xs) v
                  (believe-can-infer Ag (notI) T-xs)) is true in $KB_{self}$, {
                      add Ag to candidateList;}
        }else return null;
    }
    else return null;
    randomly select Ag from candidateList;
    Ask Ag for I;
    return value received from I;
}
/* Independently executed by each agent after the agent executes
updateKB().*/
proactiveTell($Kb_{self}$, T) {
    for each conjunct I for which (I, T) is true in $Kb_{self}$, consider the
    information flow <I, neederList, providerList>
    ∀ Agn ∈ neederlist
        if ((believe-can-see Agn (I) T) v (believe-can-infer Agn (I) T)
        is not true in $Kb_{self}$)
            tell Agn (I);[2]
}
```

Figure 3 Proactive information exchange protocols

[1] Note, however, that self does not necessarily know what these values are. This is useful, though, in case self needs to make a proactive ask.

[2] Note that there is no need to say anything about previous time points, as those would have been handled when they were first entered. Further, there is no need to explicitly consider notI; if true, it will be entered as a fact on its own.

Experimental Results

We use a synthetic multi-agent wumpus world as our test domain. The experiments are performed on five randomly generated worlds with 20 by 20 cells. Each world has 20 wumpuses, 8 pits, and 20 piles of gold. The team goal is to kill wumpuses and get the gold without being killed. Four agents, 1 carrier and 3 fighters compose a team. The carrier finds wumpuses and picks up gold. Fighters shoot wumpuses. Every agent can sense a stench (from adjacent wumpuses), a breeze (from adjacent pits), and glitter (from the same position) of gold.

The world simulation maintains object properties and actions, such as objects' locations, agents' arrows, etc. All agents use the same KB_{Env} to reason about the environment and for determining their priority of actions.

Agents may also have additional sensing capabilities, defined by observation rules in its knowledge base. For example, the carrier's sensing rules may be:

```
(see ca (location ?o ?x ?y) ?tc //see property location
   (location ca ?xc ?yc ?tc) (location ?o ?x ?y ?t)
   (inradius ?xc ?yc ?x ?y rca) (= ?tc ?t))
(see ca (hasarrow ?o) ?tc //see property has arrow
   (see ca (location ?o ?x ?y) ?t) (= ?tc ?t))
(see ca (shootwumpus ?o) ?tc //see action shoot
   (see ca (location ?o ?x ?y) ?t) (= ?tc ?t))
(believe-can-see ca fi (location ?o ?x ?y) ?ti //belief
   (location fi ?xi ?yi ?ti) (location ?o ?x ?y ?t)
   (inradius ?xi ?yi ?x ?y rfi) (= ?ti ?t))
```

The carrier can see all objects (and their locations) within a radius of rca. It can also sense whether or not fighters have arrows, and see any fighter shoot a wumpus if the fighter is within range. Further, the carrier knows that fighter fi can see objects within rfi of fi.

Each team is allowed to operate a fixed number of 150 steps. Agents use proactiveTell to impart information they just learned if they believe other agents will need it. Agents use activeAsk to request two categories information: 1) an unknown conjunct that is part of a precondition of plan or operator, (e.g., wumpus location) 2) an unknown conjunct that is part of a constraint, such as selecting the agent closest to the wumpus.

We use two teams, designated A and B. Except for the observability rules, conditions of both teams are exactly the same. All agents move at each time step. In the absence of any target information (wumpus or gold), they move randomly. If they are aware of a target location requiring action on their part (shoot wumpus or pick up gold), they move toward the target. In all cases, they avoid unsafe locations. We report two experiments.

For the first experiment, the teams are composed as:

- Team A: The carrier can observe objects within a radius of 5 grid cells, and each fighter can see objects within a radius of 3 grid cells.
- Team B: None of the agents have any sensing capabilities beyond the basic capabilities described at the beginning of the section.

We use measures performance, reflecting the number of wumpuses killed, amount of communication used and gold recovered. In order to make comparisons easier, we have chosen to have decreasing values indicate improving performance, e.g., smaller numbers of communication messages are better. To maintain this uniformity with some parameters of interest, we use as our measure the quantity not achieved by the team rather than the number achieved, e.g., the number of wumpuses left alive rather than the number killed. The results are shown in Table 1.

Table 1 shows that, as expected, Team A found and killed more wumpuses than Team B. Basically, the further the agents can see (we have run other experiments), the more wumpuses are killed. Team A used somewhat more messages than Team B. However, one must consider that the more wumpuses found, the more messages that are likely to be sent. Hence, it makes more sense to compare the average number of messages per wumpus killed. In these terms, the performance of Team A, is much better than that of Team B. Hence, our algorithms for managing the sensing capabilities of agents have been effective.

TeamA	T1	T2	T3	T4	T5	T6	T7
Test1	5	10	0	32	60	4.00	2.03
Test2	4	7	0	35	58	3.62	2.18
Test3	4	8	0	38	66	4.12	2.31
Test4	5	8	0	32	58	3.86	2.03
Test5	6	10	0	24	45	3.21	1.71
TeamB	T1	T2	T3	T4	T5	T6	T7
Test1	16	14	0	27	54	13.50	6.75
Test2	16	16	0	27	54	13.50	6.75
Test3	16	14	0	27	54	13.50	6.75
Test4	14	15	0	30	60	10.00	5.00
Test5	15	14	0	30	60	10.00	5.00

T1: number of wumpuses left alive
T2: amount of gold left unfound
T3: number of agent killed
T4: total number of activeAsks used
T5: total number of proactiveTells used
T6: average number of activeAsks per wumpus killed
T7: average number of proactiveTells per wumpus killed

Table 1 Experiment Results

From the first experiment, it appears that the average communication per wumpus killed is improved with sensing. We designed Experiment 2 to show how team size affects this metric. We use the same sensing capabilities for Teams A and B. However, we increased the number of fighters from 3 to 4 and 5, respectively, in two tests that we run.

Figure 4 shows the trend of average proactiveTell per wumpus killed as a function of increasing team size. Team A's per unit communication is almost independent of size. Team B has an obvious increase in average communication with increasing the team size. In Team B, a fighter kills a wumpus only after getting the wumpus location from a carrier. So, increasing the number of fighters will do little in terms of the number wumpus of killed, but the communication load goes up because more fighters must be proactively told the location of wumpuses.

We also tested several different elements which may affect the average communication per killed wumpus value. These elements include team starting locations (clustered in one part of the world, or randomly scattered), number of wumpuses, and number of carriers. We tested these elements individually, keeping the elements not being tested the same as in the first experiment. In all cases, the average number of proactiveTells per wumpus killed for Team A remained significantly better than for Team B. Indeed, the average proactive communication remained roughly constant for Team B, while none of the variations significantly reduced Team A's communication.

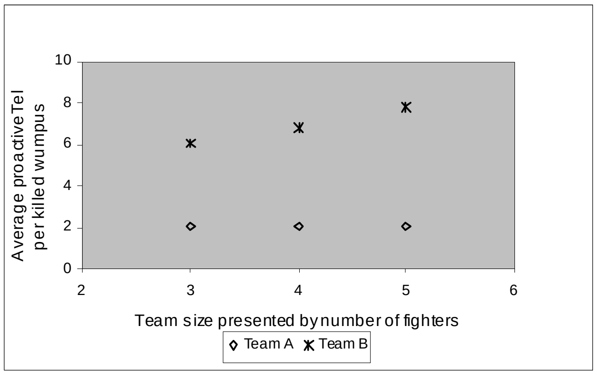

Figure 4 The comparison of average communication with different team size

Discussion and Conclusions

In this paper, we have presented the notion of agent sensing as a mechanism for improving performance and reducing inter-agent communication. We have presented algorithms for achieving this in our agent architecture, and we have shown experimental results that demonstrate the effectiveness of our agent sensing algorithms.

In the future, we plan to manage time flow in a less restrictive manner and develop additional mechanisms for reducing inter-agent communication. For our present proactive information algorithm to work, each agent has to know the information needs and production of each other agent. This is done now by the use of information flows. However, this is too rigid. We would like to make the recognition of needed information more dynamic. We plan to work on a way to recognize the goals of other agents and track the sequence of sub-goals on which an they are working dynamically. Using this information together with the action an agent has most recently performed we will dynamically estimate the most likely information needs of other agents over a finite time horizon. Then, we can send only information unknown to other agents that will be needed in the near future.

Acknowledgements

This work was supported in part by DoD MURI grant F49620-00-I-326 administered through AFOSR.

References

1. Cohen, P.R., H. Levesque, and I. Smith, *On Team Formation*, in *Contemporary Action Theory*, J.H.a.R. Tuomela, Editor. 1997, Synthese.
2. Jennings, N., *Controlling cooperative problem solving in industrial multi-agent systems using joint intentions.* Artificial Intelligence, 1995: p. 75.
3. Huber, M. and E. Durfee. *Deciding when to commit to action during observation-based coordination*. in *First International Conference on MultiAgent Systems*. 1995. San Francisco, CA: MIT Press.
4. Stone, P. and M. Veloso, *Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork.* Artificial Intelligence, 1999. **110**(2): p. 241-273.
5. Kaminka, G.A. and M. Tambe, *Robust Agent Teams via Socially-Attentive Monitoring.* Journal of Artificial Intelligence Research, 2000. **12**: p. 105-147.
6. Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., and Volz, R. A., *CAST: Collaborative Agents for Simulating Teamwork.* In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI2001).* 2001. Seattle, WA.
7. Yen, J., Fan, X., and Volz, R. A., *On Proactive Delivery of Needed Information to Teammates.* (Submitted to the *Workshop on Teamwork and Coalition Formation at AAMAS'02*).
8. Yin, J., Miller, M. S., Ioerger, T. R., Yen, J., and Volz, R. A., *A Knowledge-Based Approach for Designing Intelligent Team Training Systems.* In *Proceedings of the Fourth International Conference on Autonomous Agents*. 2000. Barcelona, Spain.
9. Ioerger, T.R., *JARE MENU*. 2001.
10. Russell, S. and P. Norvig, *Artificial Intelligence A Modern Approach*. 1995, NJ: Prentice Hall.
11. Fagin, R., et al., *Reasoning About Knowledge*. 1995, Cambridge, MA: The MIT Press.
12. Bell, J. and Z. Huang. *Seeing is believing*. In *Common Sense 98'*. 1998.