Multiple Roles, Multiple Teams, Dynamic Environment: Autonomous Netrek Agents*

Marcus J. Huber
Artificial Intelligence Lab
The University of Michigan
Ann Arbor, Michigan
marcush@eecs.umich.edu

Tedd Hadley

University of California, Irvine
Irvine, California
hadley@uci.edu

Abstract

We describe the architecture and performance of autonomous agents that play the complex, multi-faceted internet game called Netrek. To perform competently within Netrek, an agent must be capable of 1) reacting in real-time to arcade-like tasks and environment changes within the local surroundings, 2) reasoning about strategy and long-term tasks at a slower pace and on a global scale, and 3) coordinating both with teammates and against members of the opposing team. Agents capable of operating within the Netrek environment, therefore, require a reactive, flexible, multi-level framework. We describe in detail our current agent implementation, which is based upon the UMPRS (University of Michigan Procedural Reasoning System) architecture. As multi-agent coordination plays a significant role within the Netrek domain, we describe recent research in extending the current communication-based intra-team coordination scheme with a plan recognition scheme that enhances both intra- and inter-team coordination.

Introduction

We are currently involved in research in multi-agent coordination that utilizes totally autonomous, "robotic" agents to test our theories. We wished to perform experiments within a complex, dynamic environment, where the agents would perform complex tasks and be motivated to coordinate with and against one another. After looking at the characteristics of several experimental environments and task domains, we finally arrived upon the Netrek domain and environment. We will describe Netrek is some detail later, but in short, Netrek is a complex, real-time, multi-player, multi-team internet game normally played by humans typically distributed across the country and possibly the world.

Designing and implementing autonomous robotic agents capable of competently playing Netrek as an individual, and also capable of coordinating with members of the same team while coordinating against members of the opposing team, a natural component of the game, is a significant challenge. Interesting issues arise in many aspects of the architecture and behavior of individual agents, and coordination between multiple agents. In this paper, we describe the implementation of the agents currently being utilized in our research. The next section describes the Netrek environment in detail, including team and individual goals. The following section describes the agent architecture currently utilized, and describes how the agent makes decisions about its appropriate role and then executes the plan associated with that role. In the section following this, we briefly describe recent augmentation of the robotic agents to add plan recognition capabilities, and describe how this extension improves the agent's performance. Finally, in the last section, we summarize everything and spend some time discussing ideas we would like to the explore in the future.

Simulation Domain and Environment

Environment Characteristics

The world in which our autonomous agents live is a dynamic, real-time, fine-grained, multi-agent simulator originally constructed for the internet network game called Netrek. The environment is managed by a server process that is distinct from the players, which connect as clients from anywhere in the world. In the simulation, up to 16 players act in a grid world of 100,000 units by 100,000 units, where the agents are organized into two teams, with up to 8 agents on each team.

The environment has distributed throughout it forty labeled, stationary landmarks, called "planets", as

*This research was sponsored in part by the NSF under grant IRI-9158473, and by DARPA under contracts DAAE-07-92-C-R012 and N66001-93-D-0058.

shown on the right side of Figure 1; the right side is the long-range, "galactic" view that a player has of the environment. Each planet has resources associated with it, with each planet possibly having different resources than other planets. Each planet has a dynamically changing, limited number of a resource called "armies", which grow upon each planet. The rest of a planet's resources are static, and determine whether the planet has extra refueling, repair, or army growth capability. Each of four teams initially own ten of the forty planets, with each team's planets occupying one of the four corners of the "galaxy". Only two teams compete in any single "standard" game so that only twenty planets are actually contested. The Netrek server updates the environment rapidly, ten times per second, and this is the rate at which the agents get updates about the world. The agents are typically constrained to be able to perform only five atomic actions per second.

Players also have a smaller, more precise, "tactical" view of the world, shown on the left side of Figure 1, and this is where they focus much of their attention during interaction with other players. During play, individual players "dogfight" individual agents of the other team on and those agents that are victorious are then capable of picking up armies from the team's own planets (which produce armies) and transporting the armies to a planet owned by the opposing team. If more armies are transported to an opponent's planet than that planet has armies, the planet changes ownership. The global objective of each team in the simulation is to capture all of the other team's planets. The agents that lose dogfights are placed back near the "homeworld" of the player's team (Romulus for the Romulans, Earth for the Federation, etc.).

Players in the game are faced with a large number of decisions, most of which must be made very quickly in order to be able to act in time to be effective. Besides the tactical "arcade"-level decision making of dogfighting, players must also decide where and when to dogfight to be most effective. These decisions are dependent upon the larger, strategic context of the game, such as where concentrations of players are, what planets are being contested, which opposing players are carrying armies, what other teammates are doing, etc.

Agent Architecture

In this section, we describe the implemented agent architecture. This architecture addresses many issues found in "real" domains such as mobile robotics, where the environment is complex, dynamic, and uncertain. As mentioned earlier, all agents in the environment are permitted to perform actions five times a second typically, where the actions consist of various movement, combat, navigation, and communication actions. A player in Netrek cannot afford to sit idle for a long period of time reasoning about what it should do. The environment may change dramatically while the player is reasoning, most likely ruining the reasoning already performed. Instead, Netrek agents are better off remaining active at all times, acting and reacting continuously in the short term, directing and altering its actions as best it can to achieve the long-term goals that it has.

Mobile robotics research has, in particular, recognized and addressed architectures for performing significant levels of reasoning while dealing with dynamic environments. Some architectures break the architecture into several distinct modules, with a "reasoning" component that reasons over long-term goals, a "scheduling" component that orders goals and plans, and a "reactive" component that performed limited, short-term reasoning based upon the scheduled tasks (Connell 1992; Gat 1992). Other architectures are broken into a number of coordinating modules designed for particular functionality and behavior characteristics with some modules performing reasoning and other modules handling reactive short-term behavior (Simmons 1990; Arkin, Riseman, & Hanson 1987). Another architecture, the Procedural Reasoning System (PRS) manages reactivity while exhibiting goal-driven behavior by continually interleaving execution of its long-term plans with evaluation of the current situation, which can invoke reactive plans when necessary (Ingrand, Georgeff, & Rao 1992). All of these architectures have previously demonstrated their ability to work effectively in domains with characteristics similar to Netrek.

The architecture that we chose to base our Netrek agents upon is a C++ implementation of PRS done by the University of Michigan, called UMPRS (Huber *et al.* 1993), which we describe in some detail below. The decision to use UMPRS was made both because UMPRS has characteristics that promised to match well with the domain and tasks and was readily available. We are interested in investigating the use of alternative architectures and we plan on exploring this in the future.

UMPRS - The University of Michigan Procedural Reasoning System

UMPRS is a general-purpose reasoning system, integrating traditional goal-directed reasoning and reactive behavior. UMPRS continuously tests its decisions (both high- and low-level) against its changing knowledge about the world, and can redirect the choices of actions dynamically while remaining purposeful to

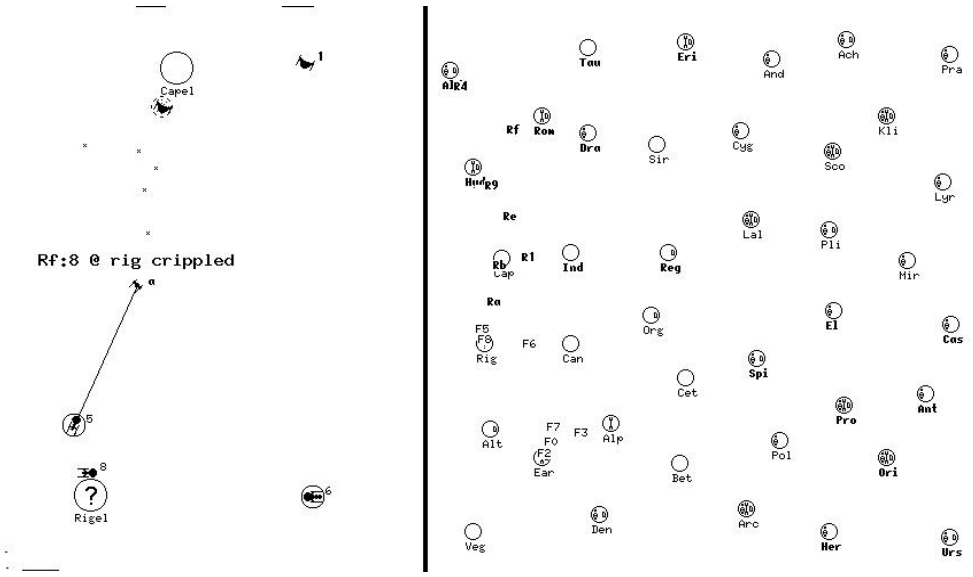

Figure 1: The standard, asymmetric Netrek tactical and galactic displays showing several players from both teams. This is a scene (from the perspective of Romulan player 'a') from a human Netrek game with eight players per side. Interesting features in the scene include player '5' phasoring player 'a', torpedoes from player '5' flying toward player 'b', and player 'b' (near the planet labeled "Capel") just beginning to cloak (becoming invisible on the tactical display. The planet labeled "Rigel" with the "?" symbol in the middle of planet means that the planet was just taken by the Federation team. The different ship icons indicate different ship types and/or teams.

the extent of unexpected changes to the environment. UMPRS is composed of five major components: a database representing the current world state; a library of plans; a set of goals to be achieved; an intention structure that maintains the runtime state of the set of currently active goals; and the interpreter, the active component that reasons about the other components to determine what to do. The UMPRS execution cycle is very similar to that originally discussed for PRS in (Ingrand, Georgeff, & Rao 1992). When a new fact about the world is recorded in the world model through sensing, communication, or internal conclusions, the UMPRS interpreter will look for KAs that are applicable to the new situation. Likewise, if a new goal (or goals) is added, UMPRS looks for applicable KAs to satisfy the new goal(s). In either case, UMPRS selects one of the applicable KAs, weighs its importance against that which it is actively working on, and if found to be of more importance, places it on the intention structure, and then executes actions in the active "intention". Alternatively, the system may continue to execute the intention currently on the intention structure. A diagram of the UMPRS architecture is shown in Figure 2.

The UMPRS world model holds the facts that represent the current state of the world. Information that might be kept there includes state variables, sensory information, conclusions from deduction or inferencing, modeling information, etc.

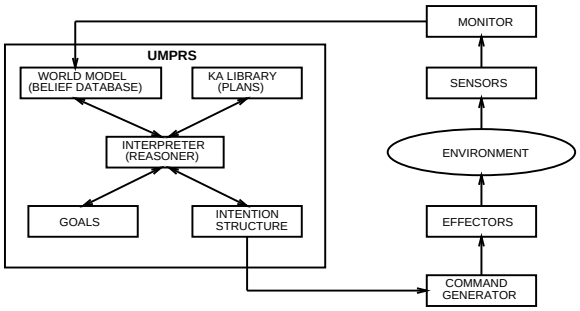

Figure 2: The UMPRS system architecture.

The UMPRS plan library contains one or more KAs which define procedural methods for accomplishing goals. A KA's applicability is limited to a particular PURPOSE, or goal, and may be further constrained to a certain CONTEXT. The procedure to follow in order to accomplish the goal is given in the KA's BODY. The PRIORITY section can be used to influence selection of certain procedures over others through the meta-level reasoning mechanism of UMPRS. An optional section, called the FAILURE section, can be used to specify a procedure to execute when the KA fails for some reason. Each action in a KA's BODY can specify a goal or

condition to achieve. In addition, a KA action can be a low–level primitive function to execute directly or an action that retrieves data from or modifies the world model. Furthermore, iteration and branching are supported for conditional execution of actions.

UMPRS's goal list constitutes conditions or tasks that the system is supposed to achieve. These goals are continually evaluated to determine the most important task to pursue. A plan is selected to achieve the goal when the context constraints of an applicable plan is satisfied by the current situation. Subsequent execution of the plan may lead to establishment of subgoals, or perhaps even more top-level goals.

The UMPRS intention structure maintains information related to the runtime state of progress made toward the system's top-level goals. The system will typically have more than one of these top-level goals, and each of these goals may, during execution, invoke subgoals to achieve lower-level goals. A goal that is being pursued may be interrupted by a higher priority goal and then later resumed (if possible). When a goal gets suspended, due to a higher level goal becoming applicable, the current state of execution of the current goal is stored.

Netrek Agent Design

The agents were originally programmed entirely in the *C* high-level programming language several years ago. Recently, this *C* code was in part replaced by UMPRS KAs and UMPRS primitive functions in order to more explicitly represent the agents' goals, procedures, and situational contexts, and thereby increase the flexibility of the agent tasking. One of UMPRS's primary advantages is its natural ease at representing procedural knowledge, so that converting from *C* code to KAs was particularly straightforward.

The agent's top-level KAs deal with the agent's interface to the simulator and other such mechanics. The KAs that represent the beliefs, desires, and intentions of the agent with regards to the Netrek domain and tasks are found in the "middle" of the plan hierarchy. At this point, their are a number of KAs associated with various agent roles. The more important roles that the agents can take include: **engage**, dogfight with the closest opponent; **assault**, a complex role involving either bombing a planet (to reduce the number of armies on an opponent's planet), or dropping armies on an opponents planet; **escort**, help a team agent when it tries to drop armies on an opponent's planet; **ogg**, suicide attack an opponent; **protect**, defend a team-owned planet from an assault (bomb or capture); and **get armies**, move to a planet that has team armies and pick them up.

Below, in Figure 3, we show the KA that the robot agents use to decide upon their current role and perform that role's task(s). This KA's purpose (goal) is to determine the next command to perform. This KA is always applicable (the KA has an empty CONTEXT), and is a subgoal for top-level KAs. The KA's procedural BODY first checks to see if the agent has decided to exit (e.g., the game is over) and skips all of the significant actions of the BODY. If the agent is still active, the agent executes a series of primitive actions that perform miscellaneous low-level functionality (indicated by the UMPRS action EXECUTE, which specifies primitive actions), such as firing phasers and torpedoes, dodging attacks, and gathering current information about players and planets. These primitive actions in essence perform much of the agent's "reactive" behavior.

One statement of the procedural BODY, however, specifies a subgoal (the `ACHIEVE goto_role` line) to perform the agent's current role. There are a number of KAs for this subgoal, one or more associated with each of the roles listed above. Each of these "role" KAs perform the "goal-directed" behavior of the agents according to the situation. Several of the KAs used are shown in Figure 4. These KAs handle the various situational contexts that may arise. The first KA is selected by UMPRS if the agent is ready to take a planet (e.g., it is currently carrying armies). The second KA is selected if the agent is not ready to take a planet (e.g., no armies, one of the conditions embedded in the predicate `not_ready_to_take`), and the third KA is selected if the agent is not ready to even assault because it has not selected a target planet (embedded within the predicate `not_ready_to_assault`.) When the agent is ready to assault a planet (the first KA), it performs the assault if it is orbiting the target planet, otherwise it navigates to the target planet first.

Agent Performance

Playing against the agents revealed many interesting characteristics. The robotic agents are formidable dogfighters, reacting extremely quickly and effectively to attacks upon them while simultaneously effectively inflicting damage. The robotic agents dodge torpedoes exceedingly well, changing speed and course as needed. Few human players are competent enough dogfighters to be able to regularly defeat the robotic agents. Winning dogfights is a vital segment of the game as it enables the victor to pick up armies and drop them on the opposing team's planets.

The robotic agents perform poorly in several aspects of the game, however. Foremost among these is the agents' predictability, lack of guile, and steadfastness

```
KA {
NAME: "Get next command"
PURPOSE: ACHIEVE R_NextCommand;
CONTEXT:
BODY:
   EXECUTE initR_NextCommand $done;
   OR {
      TEST (== $done 1);
      EXECUTE print "initR_NextCommand DONE!\n";
   }
   {
      TEST (!= $done 1);
      EXECUTE do_alert;
      EXECUTE phaser_plasmas;
      EXECUTE warfare 1;
      EXECUTE warfare 2;
      EXECUTE handle_misc_problems;
      EXECUTE init_dodge;
      EXECUTE do_defense;
      ACHIEVE do_role (get_role);
      EXECUTE update_players;
      EXECUTE update_planets;
      EXECUTE decide;
      EXECUTE exitR_NextCommand;
   };
}
```

Figure 3: The top-level KA for the autonomous Netrek agent.

```
KA {
NAME: "Assault a planet"
PURPOSE: ACHIEVE assault;
CONTEXT: (|| (! (not_ready_to_assault))
             (! (not_ready_to_take)))
BODY:
  EXECUTE set_statestate 4; // S_DEFENSE
  EXECUTE print "PRS: Assaulting planet.n";
  OR {
    TEST (orbiting "ASSAULT");
    EXECUTE print "PRS: Orbiting planet.\n";
    EXECUTE assault_planet;
  }{
    TEST (== (get_stateassault_req) 1);
    EXECUTE print "PRS: Going to take planet.\n";
    EXECUTE notify_take 1;
    EXECUTE goto_assault_planet;
  }{
    EXECUTE print "PRS: Going to assault planet.\n";
    EXECUTE goto_assault_planet;
  };
}

//--------------------------------
KA {
NAME:    "Drop assaulting planet"
PURPOSE: ACHIEVE assault;
CONTEXT: (not_ready_to_take);
BODY:
  EXECUTE print "PRS assault: Not ready to take.\n";
  EXECUTE unassault_c "no armies";
}

//--------------------------------
KA {
NAME:    "Not ready to bomb a planet"
PURPOSE: ACHIEVE assault;
CONTEXT: (not_ready_to_assault);
BODY:
  EXECUTE print "PRS assault: Not ready to assault.\n";
}
```

Figure 4: The KAs for assaulting (bombing or taking) a planet.

to fulfill their missions. Human players quickly figure out the robotic agents' individual weaknesses and take advantage of them. One aspect of future work on the agents will be to add flexibility in the pursuit of their goals so that they are not as predictable.

The robotic agents are also inferior to their human counterparts in coordinating their activities. While play in a game, it is easy to see that players working together toward a common goal greatly improve the chances of success. A player attempting to take a planet with the help of seven teammates that are actively protecting the player has a much better chance of doing so compared to having no help from teammates. In the next section we show how coordination of the agents was improved greatly in a number of different areas by extending their reasoning capabilities to include the ablity to infer the goals and plans of other players (this is called *plan recognition*)

Extensions and Experiments

The procedure of adding plan recognition capabilities to the Netrek agents was greatly facilitated through the use of the ASPRN (Automated Synthesis of Plan Recognition Networks) system (Huber, Durfee, & Wellman 1994). ASPRN quickly and automatically constructs Plan Recognition Networks (PRNs) from an agent's native, executable plan library. PRNs are specially constructed probabilistic networks that support plan recognition inferencing. The agent's decision-making routines were then extended to consider some of the information supplied by the PRNs. Future work may include a more thorough and complete utilization of the rich information provided by PRNs.

We ran a number of experiments to explore the relative advantages and disadvantages of plan-recognition-based coordination of teams of agents. Our first experiments, however, simply pitted the UMPRS version of the agents against the original C agents. We use the C agents as a performance baseline against which to measure the impact of various architectural and environmental factors upon agent and coordination performance. We then went on to compare UMPRS plan-recognition-equipped agents against the original C agents.

During each experiment, we accumulated a subset of statistics which are typically accumulated during human INL (International Netrek League) matches (Ahn 1996). These statistics are analyzed after human games to determine relative performance and provide insight into autonomous agent performance as well. These statistic consisted of:

- **tpt** - total planets taken, the number of planets totally converted to the team's ownership.

- **tpd** - total planets destroyed, the number of planets made independent (i.e., reduced to 0 armies).
- **tek** - total enemy killed, the number of dogfights won by team agents.
- **def** - deaths by enemy fire, the number of times team agents died.
- **tab** - total armies bombed, the number of armies bombed off enemy planets.
- **tac** - total armies carried, the number of armies picked up by the team's agents.
- **cak** - carried armies killed, the number of armies that died while being carried by agents.
- **eao** - enemy armies ogged, the number of armies killed as a result of enemy fire.

Some additional information was also captured, such as total experiment time (referred to as "time" in experiment results shown later) and bombing latency (the accumulated time between when a planet grew armies and when the planet was bombed, and referred to as "latency" in the results).

Explicit vs. Implicit Representation Results

We briefly describe our first experiment's results as they simply represent a baseline comparison of the UMPRS agents (without plan recognition or communication capabilities) when they competed against the equivalent *C* agents. We hypothesized that the directly converted UMPRS agents, with their explicit representation of goals and plans, could only do as well but not better than the original agents because of the inherent overhead of UMPRS. We also hypothesized that the UMPRS agents will be easier to retask and be more flexible in the face of changing situations once the agents are extended to fully utilized UMPRS's functionality. Our experiments shed some light on the former hypothesis, but we leave to future work the redesign and extension of the agents to leverage UMPRS.

As shown in Table 1, of forty experiments, the original *C* agents won 25, or 62.5 percent, of them. Experiment statistics show that the original *C* agents performed slightly better than the UMPRS agents almost across the board, indicating a slight performance decrease due to the overhead associated with the use of UMPRS and its explicit goal and plan representation and execution scheme. Clearly, the computational overhead associated with UMPRS's more general and flexible architecture outweighed its benefits in a direct translation from *C* functions. However, we expect that we will realize significant benefits in both reduced implementation time as well as increased agent performance when we extend the agents' behavior to deal with more of the task and environmental complexity.

Team	Stat.	total	avg.	std.dev
UMPRS bots	tpt	75	1.88	7.02
	tpd	90	2.25	8.38
	tek	775	19.38	54.97
	def	877	21.93	42.85
	tab	8007	200.18	424.93
	tac	728	18.2	52.47
	cak	49	1.23	7.92
	eao	55	1.38	5.88
	time		15663.0	
	latency		23453.0	
C bots	tpt	107	2.68	7.30
	tpd	115	2.88	8.04
	tek	864	21.6	41.31
	def	794	19.85	56.46
	tab	7985	199.62	427.59
	tac	943	23.58	52.80
	cak	55	1.38	5.88
	eao	44	1.1	6.97
	time		14227.9	
	latency		21188.1	

Table 1: Experiment statistics of UM-PRS agents vs. original *C* agents.

Non-communicating agent results

The next two sets of experiments were conducted to determine how effective our agents that coordinate using their new plan recognition capabilities would be against the original agents. In the first experiments, we placed the agents in a world where their communication channels are disabled for some reason (e.g., being jammed). In the second set of experiments, we pitted the plan recognizing agents against *C* agents that *could* communicate and coordinate, although in a limited manner.

The results of the first series of experiments, with the plan-recognizing agents ("PRbots") competing against non-communicating *C* agents ("Stdbots"), is shown in Table 2. In this series, the plan recognizing robots won 36 games. Almost all of the statistics in the table point to dominance by the PRbots. The PRbots captured and destroyed more planets, won more dogfights, carried more armies, and killed more enemy-carried armies. Bombing latency reflects that the PRbots coordinated much better in their bombing and distributed themselves better when multiple planets needed bombing (more on this later). The higher "tab" statistic for the Stdbots seems at first glance to indicate that the Stdbots did a better job bombing overall,

but this must be mitigated by the fact that, as the experiments progressed, the Stdbots had fewer planets producing armies while the PRbots respectively had more planets.

In this set of experiments, the PRbots clearly outmatched the Stdbots. One important result of these experiments was the establishment that the plan recognizing agents were apparently able to recognize the goals/plans of the other agents early enough to give the coordinating agents an opportunity to be in the right place and time to help or hinder, as the situation required. Had the same observations been made just before the observed agent completed the critical portion of its task (e.g., dropping armies on a planet) the observing agent would have been unable coordinate with the observed agent. This introduces the concept of "observation distribution", a measure of where observable actions occur during execution of a plan (e.g., early or late in the plan). This is an extremely important issue in plan recognition and one that deserves much more attention.

Another important aspect was the negligible naturally occurring overhead associated with performing plan recognition observations and inferencing (typically 0.02 real-time seconds). Clearly this is a nearly ideal situation, and in future work we will simulate more complex and higher cost perceptual processing and inferencing in order to gain a better understanding of where plan recognition becomes too much of a burden to be of utility.

Observing the experiments showed that the PRbots bombed in a much more coordinated fashion, using their plan recognition capabilities to determine that some other teammate was already bombing (or going to bomb) an enemy planet and choosing another planet to bomb or, if there were no more planets to bomb, switching to some other role. The Stdbots, on the other hand, quite often bombed planets *en masse* and oftentimes moved and bombed as a group. The PRbots, with their better bombing efficiency, had more opportunity to perform other roles such as protecting armies and defending planets from enemies carrying armies.

In the second series of experiments, we have started to explore how plan-recognition-based coordination compares with communication-based coordination of varying expressiveness. In these experiments, C agents utilized a limited communication protocol to inform teammates when going to attempt to capture a planet[1]; the receiving C agents would use this information in determining their own course of action and would assist the capturing agent if they could.

Team	Stat.	total	avg.	std.dev
PRBots	tpt	122	3.05	5.86
	tpd	143	3.575	5.80
	tek	954	23.85	52.85
	def	752	18.8	49.04
	tab	8057	201.425	460.60
	tac	1134	28.35	44.32
	cak	37	0.925	6.33
	eao	80	2	8.60
	time		16469.3	
	latency		30649.9	
Stdbots	tpt	47	1.175	5.53
	tpd	46	1.15	5.20
	tek	738	18.45	48.22
	def	974	24.35	52.72
	tab	8136	203.4	476.03
	tac	484	12.1	50.79
	cak	80	2	8.60
	eao	36	0.9	6.34
	time		15338	
	latency		47219.9	

Table 2: Experiment statistics of PR agents vs. non-communicating "standard" agents.

The results of forty experiments of the plan recognizing agents competing against C agents with limited communications capabilities is shown in Table 3. Of these experiments, the plan recognizing robots won 35 while the standard robots won just five. The statistics in Table 3 again show dominance by the plan recognizing agents. Bombing latency again shows the PRbots' improved bombing coordination, with total bombing being nearly equivalent. Again, this increased efficiency permitted the PRbots to be more flexible, permitting them to more dynamically switch to other, more useful, roles. There are two significant items to note in the results of this experiment. First, the "eao"/"cak" ratio was reduced from a nearly 2:1 ratio in the non-communicating experiments to a 1:1 ratio in the communicating experiments. This indicates that the Stdbots were much more successful in protecting (coordinating with) teammates when the teammates were taking planets. Second, the game time statistics for when the Stdbots won is significantly lower than the game time statistics for when the PRbots won. This suggests that the Stdbots only won when they could very quickly exploit some particular advantage before the PRbots could compensate. Conversely, as games became extended, the PRbots had an increased chance of compensating for any advantage held by the Stdbots and eventually established and maintained their own

[1] This was the default C agent configuration and communication protocol implemented. Future work will examine relative performance as the expressiveness of the protocol increases.

Team	Stat.	total	avg.	std.dev
PRbots	tpt	129	3.23	7.38
	tpd	134	3.35	6.80
	tek	937	23.43	59.82
	def	766	19.15	44.24
	tab	8005	200.13	531.73
	tac	1178	29.45	60.41
	cak	62	1.55	12.53
	eao	69	1.48	12.21
	time		15147.8	
	latency		26234.4	
Stdbots	tpt	41	1.03	7.46
	tpd	56	1.4	8.47
	tek	776	19.4	53.62
	def	981	24.53	69.76
	tab	8167	204.18	620.64
	tac	511	12.8	71.54
	cak	64	1.6	12.21
	eao	65	1.63	14.71
	time		12478	
	latency		17742.63	

Table 3: Experiment statistics of PR agents vs. communicating "standard" agents.

advantage for the victory.

Discussion Coordination through communication by the Stdbots led to increased Stdbot success. The Stdbots' communication language and protocol, as defined for the C agents, are not powerful enough, however, to overcome the flexibility provided the PRbots by PRNs. The ability to utilize a restrictive communication protocol manifested itself in the Stdbot's increased success in safely delivering armies to the PRbot planets. The Stdbots' communication language and protocol was too restrictive, however, to impact significantly enough upon other aspects of the experiments to completely reduce the dominance by the PRbots. The relatively rich modeling of the complete task structure maintained by the PRbots provided them with a broader scope of coordination information than that provided by the Stdbots' communicated information, even though the information provided to the PRbots by the PRNs was uncertain and perhaps incomplete. Of note here is that the overhead associated with the Stdbot's communication was negligible, posing virtually no computational load and having virtually instantaneous transmission. Issues and tradeoffs related to non-negligible communications with regard to relative performance against plan recognition is yet another research area to be explored.

Summary

We have described in some detail the design and characteristics of our autonomous Netrek playing agents. The agents have proven their ability to pursue complex goals within a very complex, dynamic environment. Although they do not currently perform at a level to compete with skilled human players, they nonetheless represent a significant stepping stone toward that performance level; the addition of relatively modest plan recognition capabilities has already demonstrated significantly improved performance. The UMPRS implementation promises to offer significant advantages to agent development in both the short and long term due to the increased flexibility and expressiveness of the agent's goals and plans.

Although primarily used so far to explore plan recognition issues, we are looking forward to future research utilizing these agents in other research areas as well. One area of interest is exploration of variations in the agent's architecture through more elaborate use of UMPRS and through introduction of alternative architectures. The agents and domain are also extremely conducive to exploration of many multi-agent coordination issues and approaches, including organizational structures and richer communication-based coordination paradigms.

References

Ahn, D. 1996. International Netrek League WWW Home Page. World Wide Web. http://indigo2.medeng.bgsm.edu/ ahn/INL/index.html.

Arkin, R. C.; Riseman, E. M.; and Hanson, A. R. 1987. ArRA: An architecture for vision-based robot navigation. In *Proceedings of the DARPA Image Understanding Workshop*, 17–431.

Connell, J. H. 1992. SSS: A hybrid architecture applied to robot navigation. In *Proceedings IEEE International Conference on Robotics and Automation.*

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 809–815.

Huber, M. J.; Lee, J.; Kenny, P.; and Durfee, E. H. 1993. *UM-PRS Programmer and User Guide.* The University of Michigan, 1101 Beal Avenue, Ann Arbor MI 48109.

Huber, M. J.; Durfee, E. H.; and Wellman, M. P. 1994. The automated mapping of plans for plan recognition. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 344–351.

Ingrand, F.; Georgeff, M.; and Rao, A. 1992. An architecture for real-time reasoning and system control. *IEEE Expert* 7(6):34–44.

Simmons, R. 1990. An architecture for coordinating planning, sensing and action. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.*