© Copyright by Abhilash Babu Patel, 2004

A SWAPPING MECHANISM FOR DYNAMIC TASK ASSIGNMENT IN MULTI-AGENT SYSTEMS

BY

ABHILASH BABU PATEL

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

To my family

Acknowledgments

The work is supported in part by the Defense Advanced Research Projects Agency (the DARPA IPTO TASK Program, contract number F30602-00-2-0586).

I would like to thank my advisor, Prof. Gul Agha, for fostering a flexible research environment which has allowed me explore my interests. He has provided me with the intellectual guidance and support to pursue the highest academic standards.

I have also benefited greatly from discussions with members of the Open Systems Laboratory. In particular, I would like to thank Amr Ahmed, Myeong-Wuk Jang, Predrag Tosic, Soham Mazumdar, and Tom Brown.

The idea of using swaps is due to a suggestion by Amr Ahmed, who is also primarily responsible for the design of the framework described in Chapter 6. I have taken the liberty of reusing Amr's diagrams to describe the framework. The UAV demonstation described in Chapter 6 was developed in context of the multi-agents work at UIUC Open Systems Laboratory, and I would like to acknowledge the significant contributions made by Amr Ahmed, Tom Brown, MyungJoo Ham, and Hannaneh Hajishirzi.

This thesis would not have been possible without the never-ending patience and encouragement from my family. My heartfelt thanks to them.

Table of Contents

List of Figures . . . . . **vii**

List of Abbreviations . . . . . **viii**

Chapter 1 Introduction **1**
1.1 Background 1
1.2 Dynamic Task Assignment Optimization 2
1.3 Motivation 3
1.4 Major Contributions 4
1.5 Thesis Outline 5

Chapter 2 Problem Formulation **6**
2.1 Classic Assignment Optimization 6
2.2 Dynamic Assignment Optimization 7

Chapter 3 Swapping Mechanism for Cooperative Agents . **9**
3.1 Solution Strategy 9
3.2 Applicability 11
3.3 Implementation Details 11
3.4 Correctness 12
3.5 Complexity 12
3.6 Distributed Decision Computation 13
3.6.1 Distributed Swap Opportunity Computation 13
3.6.2 Parallel Rounds 14
3.7 Multiple Agent Requirements for Tasks 17
3.8 Randomized Decision Computation 18
3.9 Rate of Change 18
3.9.1 Thrashing 19
3.9.2 Distance Example 19

Chapter 4 Swapping Mechanism for Selfish Agents **20**
4.1 Game Theoretic Problem Formulation 20
4.2 Decision Criterion 21
4.2.1 Mutual Benefit Maximization 21
4.2.2 Incentive Payments 21
4.3 Solution Concepts 22
4.4 Swapping with Coalitions 23

Chapter 5 Related Work 24
5.1 Contract Net Protocol: Swap Contracts 24
5.2 Interchange Heuristic for the Quadratic Assignment Problem 25
5.3 Dynamic Programming 26
5.4 Multi-Stage Linear Programming 27
5.5 Markov Decision Processes 27

Chapter 6 Simulation Studies 28
6.1 UAV Robot Simulation Objectives 28
6.2 The Auction Algorithm 29
6.3 Implementation Architecture 30
6.4 Experiments and Results 33

Chapter 7 Conclusions 36

References 38

List of Figures

3.1 Assignment before and after a swap 10
3.2 Parallel round cyclical preference problem 15
3.3 Multi Requirement Task modelling resolution. Here task i has a requirement of 2 agents. 17

4.1 Coalition based multi-agent systems can use cooperative swapping mechanisms for intra-coalition optimization and can use selfish swapping mechanisms for extra-coalition optimization. 23

6.1 Hardware/Software Shared Code 31
6.2 UAV Agent Architecture 32
6.3 Auction State Diagram 34
6.4 Swapping Mechanism Metrics 35

List of Abbreviations

MAS Multi-Agent System

UAV Unmanned Aerial Vehicle

LP Linear Program

QAP Quadratic Assignment Problem

Chapter 1

Introduction

1.1 Background

Multi-agent technology has generated a lot of excitement in recent years because of its promise to create intelligent software that interacts in distributed heterogeneous environments. An agent is an autonomous entity, such as a software program or a robot, whose interactions are used to perform tasks for the end user. A user can perform complex tasks with the aid of multiple agents by dividing the work or allowing each agent to leverage its specialized skills. These agents can interact amongst themselves and agents of other users to coordinate the business of the two users.

Agents can be modelled as either cooperative or selfish. Cooperative agents can share a common goal and work together to achieve those goals. An ant colony is an example of cooperative behavior: each ant works together with other ants for the prosperity of the colony. Selfish agents pursue their own interests when interacting with other agents. A free market economy is an example of selfish behavior: each person tries to maximize their personal wealth and is unconcerned with the wealth of other people.

The choice of behavior modelling in Multi-Agent Systems [MAS] depends on the interactions necessary. For some tasks a user's agents may only need to work amongst themselves, which can be modelled as cooperative behavior. While other tasks may require agents to interact with the agents of other users in order to complete their tasks. Multiple competing agencies may have control over a subset of the total agent population. In such a scenario, each user will want to maximize it's personal objectives so the agents will behave selfishly.

While meeting the needs of the user, agents must make decisions such as which task to perform, how to utilize resources, and how to perform individual tasks. In MAS each agent's decision affects the outcomes of the

other agent's decisions. For example, an agent may decide to perform task A, but that means that other agents should not also perform task A. If another agent decides to perform task A then task A will be performed twice, which may be unnecessary. Thus, a single agent cannot create an optimal decision based on a myopic viewpoint because the agent does not unilaterally control the outcome of its decision. A system designer must create a mechanism to induce each agent to make a decision that aggregates into a global optimum.

When agents interact cooperatively, an agent can sacrifice it's personal objectives for the greater good of the end user. Since cooperative agent's are solving a larger problem in a distributed manner, their decisions can be based on a distributed problem solving approach. For cooperative agents, complex tasks can be performed using standard optimization techniques that utilize the distributed agent processes [12].

This is not the case for agents that interact selfishly. Complex tasks with selfish agents cannot simply be performed using distributed problem solving techniques, since each agent will be unwilling to sacrifice it's personal objectives for the greater good. Thus, agents acting selfishly can make globally suboptimal decisions [24]. In order to create a MAS that makes globally optimal decisions, each agent must be induced to follow predictably rational decisions which aggregate to optimal solutions.

1.2 Dynamic Task Assignment Optimization

Multi-agent systems can be used to create a group of agents that are capable of servicing tasks. It is desirable for the user to service all tasks as quickly as possible using a set of agents. Agents must autonomously decide which tasks to service. However, each agent's decision to service a particular task affects the choices other agents can make. The task assignment decisions made by each agent should aggregate into an optimal assignment in order to minimize the time required to complete all the tasks.

A set of agents needs to be optimally assigned to a set of tasks. This generalized problem, known as the task assignment optimization problem, has several known solutions. These solutions assign tasks to agents given static task and agent properties. However in the real world, the agent environment and task specifications may be constantly changing. In a dynamic environment, agents must be assigned to tasks adaptively, so that each assignment remains optimal until the task has been serviced.

The dynamic assignment problem is a fundamental problem in routing

and scheduling. Consider example 1.1 about the police patrol routing problem :

Example 1.1. *The law enforcement for a district has a set of police officers. Each day officers patrol areas of the neighborhood to reduce crime and aid patrons. The patrolling tasks can be modelled as a static assignment problem, since the officers can be assigned to an area at the beginning of the day and continue to patrol that area for the rest of the day.*

We can make the scenario more realistic when we add the possibility of incoming crime reports which must be investigated by police officers. The severity of the crime reports can warrant a varying number of officers to investigate the crimes. Officers may also require additional support as the crime is investigated. In dangerous situations, officers may also be injured during a task, which may require new tasks and additional support. Thus a static assignment cannot be performed, since the nature of the tasks and the abilities of the officers can change over time. The assignments must be performed in an adaptive way so that all crimes are investigated adequately within the shortest amount of time.

The example illustrates the need for dynamic task assignment in real-world problems. The police officer scenario is similar to the kinds of dynamic environments faced by MAS. One day computational agents (such as robots) can even assist in law enforcement which would require the agent to autonomously form task assignments in the illustrated dynamic environment.

1.3 Motivation

Traditional solutions to dynamic assignment optimization have included periodically recomputing static assignments, and searching for optimums in large enumerations of possible future states. These methods can be computationally intensive and are not generally applicable to scenarios with unpredictable future states. Little attention has been given to explicitly extending the classical static assignment problem into a dynamic setting. Algorithms needs to be provided for adaptively allocating tasks among multiple agents, since a major role of agents is to autonomously solve complex tasks by dividing the task into subproblems. The swap-based task allocation protocols present in the literature do not specifically address the use and analysis of swaps in dynamic environments.

The assignment optimization problem is NP-hard due to the fact that the

problem can be reduced to an integer linear program which is known to be NP-Complete. Thus, an initial assignment is necessarily an approximation. As time progresses there may be opportunities to create better assignments which could lead to an optimal assignment over the course of time.

1.4 Major Contributions

We intend to provide a simple yet effective approach to adaptively assigning agents to tasks in dynamic environments where the nature of the tasks and agents are continuously changing. Our approach uses a swapping mechanism to allow each agent to make assignment improvements as the environment conditions change. With cooperative agents, agents can negotiate with other agents to switch assignments when there are opportunities to increase the global optimality. By using game theoretic mechanism design, selfish agents can also be induced to swap assignments whenever there is a global objective maximization opportunity. We include an analysis of the swapping mechanism for both cooperative and selfish agents. We also provide some intuition for the convergence and maintenance of optimality when the rate of change in the environment is less than the rate of swap decision making.

Even if there is little change in the environment, the solution will also improve upon the inherent sub-optimality of traditional static assignment optimization solutions. Once an initial suboptimal assignment is created the assignments are swapped whenever there is an opportunity to increase the global objectives. This process continues until there are no more swaps that increase the user's objectives. This allows for an anytime algorithm for both static and dynamic conditions.

The solution proposed is applicable to a wide range of scenarios where the nature of the agent utility function is quasi-linear and the assignment objective function is a linear sum. The tasks must also be swap-able without loss of partial completion of the task. Given these restrictions, the swapping mechanism can provide a useful tool for designing dynamic multi-agent systems.

This approach has been used in the Unmanned Aerial Vehicle [UAV] robot coordination simulation, where mobile UAVs needed to coordinate with other teammates to service moving targets by travelling to each target. Our static auction-based assignment algorithm often resulted in poor assignments when targets moved away from assigned UAVs and toward other UAVs. By incorporating the swapping mechanism, we were able to optimize the mission performance with little computational overhead. The swapping

mechanism also reduced the variance of the mission performance. Our solution has been experimentally shown to be a useful tool in dynamically assigning cooperative agents to tasks.

1.5 Thesis Outline

Chapter 2 formulates the task assignment problem in both the traditional and dynamic setting. Chapter 3 provides a proposed solution and analysis using cooperative agents. Chapter 4 extends the proposed solution and analysis for selfish agents. Chapter 5 surveys previous work in the field. Chapter 6 explains an empirical study that supports our proposed solution ideas.

Chapter 2

Problem Formulation

2.1 Classic Assignment Optimization

The classic asymmetric assignment problem deals with the question of how to assign n agents to m tasks (where $n \leq m$) in the best possible way, where the "best way" is modelled by an objective function.

Each agent derives some utility from being assigned to a particular task, and no other entity in the system derives utility. A utility function defines an agent's preferential ordering for different tasks. The characteristics of the utility function are derived from the particular problem domain. We restrict our discussion to utilities that are quasi-linear in money, consequentially an agent's net utility can be determined by the utility of the assigned task minus the necessary payments made.

Our formulation does not assume selfish or cooperative behavior for agents, and the formulation is designed to maximize the global objective function and not necessarily the individual agent utilities.

The assignment problem can be modelled as a bipartite graph matching. Let V be the set of n agents and W be the set of m tasks. An edge set E is called a matching of V to W, if every vertex of V is incident to at most one vertex from W.

Let u_j denote the utility of a task j and let $c_{ij} \in C$ denote of the cost for agent i to service task j. Let x_{ij} denote a binary variable which indicates whether agent i is assigned to task j. The benefit or value of assigning an agent i to a task j is denoted $b_{ij} = u_j - c_{ij}$.

In order to gain the most benefit from the assignment we must maximize the objective function

$$max \sum_{e_{ij} \in E} b_{ij}$$

The problem can now be formulated as a 0-1 integer linear program:

$$
\begin{aligned}
&max \sum_{i=1}^{n} b_{ij} x_{ij} \\
&\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., m \\
&\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n \\
&x_{ij} \in 0, 1 \quad i = 1, ...n \quad j = 1, ...m
\end{aligned}
$$

Once formulated as an integer linear program several algorithms exist to find an approximately optimal assignment.

2.2 Dynamic Assignment Optimization

We extend the classical assignment problem into a dynamic setting. Variables that were constant in the classical setting become functions with respect to time. Hence, the objective function must be maximized throughout time.

The benefit to agent i for servicing a task j could vary through time. Let $u_j(t)$ denote the utility of a task j at time t and let $c_{ij}(t) \in C(t)$ denote the cost for an agent i to service task j at time t. The benefit or value of assigning an agent i to a task j at time t is denoted $b_{ij}(t) = u_j(t) - c_{ij}(t)$.

Let $X_{ij}(t)$ be a function that maps a time t to an assignment of $x_{ij} \in 0, 1$ s.t. for time $= t$:

$$
\begin{aligned}
&\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., m \\
&\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n \\
&x_{ij} \in 0, 1 \quad i = 1, ...n \quad j = 1, ...m
\end{aligned}
$$

The objective function must be defined $\forall t$.

$$max \sum_{i=1}^{n} b_{ij} x_{ij}$$

This problem formulation becomes a multi-stage integer linear program, which has an extremely large search space. Furthermore, the functions $u_j(t)$ and $c_{ij}(t)$ may not be known a priori, so a search for an approximate solution

to dynamic assignment is infeasible.

Often the dynamic assignment problem is solved using an integer linear programming solution for each time interval. Thus, no predictions of the future are needed. However the time required to compute each static assignment may be prohibitively expensive.

Chapter 3

Swapping Mechanism for Cooperative Agents

The dynamic assignment problem can be considered for two types of agents: cooperative and selfish. When dealing with cooperative agents, agents are willing to sacrifice personal objectives to increase social welfare, which allows for more flexibility when designing a dynamic assignment algorithm. Cooperative agents are a useful model when all agents belong to the same authority, since the agents need only consider the aggregate welfare of the common authority.

3.1 Solution Strategy

The proposed solution to dynamic assignment is a swapping mechanism where agents interchange tasks with each other whenever the global objective function can be increased. Instead of recomputing the static assignment at each time interval, agents determine if a swap exists which increases the objective function. The swapping decision calculation is simpler than the calculations involved in the static assignment problem.

At time zero, a static assignment algorithm is used to create an initial approximately optimal assignment. At regular time intervals of length ω, agents examine their assigned task and compute whether it would be beneficial to swap tasks with another agent. A swap is determined to be beneficial for time t when the objective function after the swap is greater than the current objective function.

In figure 3.1, let agent i, who is assigned task j, consider swapping with agent k, who is assigned task l. Then the objective function increase δ is calculated:

$$\delta = b_{il} + b_{kj} - b_{ij} - b_{kl}$$

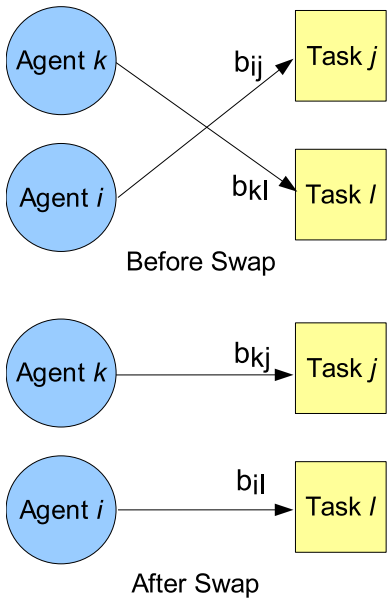

Figure 3.1: Assignment before and after a swap

if $\delta > 0$ then the swap is accepted, and agent i will service task l and agent k will service task j.

Swapping tasks with another agent may have a cost, c_{swap}, associated with it. In this case a swap is only performed when the $\delta > c_{swap}$.

The swapping algorithm can be considered an anytime algorithm, since it can be stopped at any time with an improved assignment. There are two approaches to using the swap decision criterion. One is to search for the swap opportunity with the greatest δ, and two is to search for the first beneficial swap opportunity. The first approach allows the number of swaps needed to decrease, but requires the time interval ω to be larger. The second approach allows the time interval ω to be smaller, but requires a greater number of swaps to be performed. We will consider these tradeoffs and it's implications to the variants of the swapping mechanism.

When developing an intuition for the swapping mechanism, we will begin with restrictive assumptions and then relax those assumptions. We begin by analyzing the swapping mechanism for static centralized snapshots of the dynamic environment, that is to say that the environment is fixed between swapping periods. We examine the complexity and correctness within those restrictions. We then begin reducing the complexity by introducing distributed processing and resolve the deadlock and thrashing problems associated with the increased parallelism. We then introduce the rate of change in the environment and attempt to provide some intuition for the affects of

continuously changing benefit functions.

3.2 Applicability

The cost and utility function must be continuous. By assuming continuity, a swap can be used to maintain optimality for a length of time $\epsilon > 0$. If the cost and utility functions were discontinuous then a static assignment must be used at all periods in time, since all assignments may need to be changed at every moment. In our problem formulation we assume that the utility function is quasi-linear which satisfies this restriction.

When swapping a task, if the task has already been partially completed by the previous agent, then the task must be exchangeable with no loss of previous work. An implicit assumption is that all agents are homogeneous, which means they can all perform the same tasks equally well. If they are heterogeneous then the benefit function must be tailored to quantify the abilities of expert agents vs agents with different skill sets.

3.3 Implementation Details

The assignments of each agent needs to be known to each other agent. Since cooperative agents are used, each agent can communicate to all other agents when it's assignment to a task changes. Each agent can then keep track of each agent's task in a simple manner. Over the course of a complex task, an agent may not require too many changes in the assignment, so realistically the number of messages that need to be exchanged to keep track of the assignments is minimal. Specifically, the number of messages required to maintain the assignment list is $n^2 + n \times E[\#ofswaps]$, since n^2 messages get sent for the initial assignment, and n messages get sent every time there is a swap.

An agent must also be able to evaluate the benefit function for all agents in order to compute δ. Thus, the costs and utility for each agent task pair must be transparent or determinable to all agents. This issue is generally dependent on the particulars of the system. For example, in some cases the benefit function can be easily computed by observing the environment. However, for generalized situations, each agent can share information on how tasks are valued by sending messages to each other. During each interval an agent will need to query the benefit function for each agent it is considering a swap with. We will consider several variants of the swapping mechanism, so the number of agents to consider for a swap may vary.

When the benefit function is difficult to determine explicitly but easy to estimate, then a probabilistic swap decision criteria can be introduced. The benefit for another agent to service a target j must be estimated with a degree of certainty. If the probability of a positive δ is high then the swap is performed. In cooperative multi-agents systems, agents can communicate with each other to verify that the swap is actually beneficial before executing the swap. This allows for some optimization under certain situations.

3.4 Correctness

We begin by analyzing the correctness and complexity of the swapping mechanism when restricted to centralized static snapshots of the dynamic environment. Under these assumptions, the correctness of the swapping mechanism is straightforward. Each agent can compute δ for each potential swap. Since by definition a swap is only performed when δ is positive, the swapping mechanism continuously improves the objective function.

Theorem. *The objective function will always converge to the global optima when there are no more swap opportunities where δ is > 0.*

Proof. By definition of δ and by definition of the linear sum assignment problem, there are no possibilities of increasing the objective function when no swap opportunity produces a δ greater than zero. Thus, the swapping mechanism will always increase the objective function until there are no more opportunities to increase the objective function. □

Another class of problems known as quadratic assignment problems would face problems with local optima using the swapping mechanism, but we limit our discussion to the linear sum assignment problem.

3.5 Complexity

Let a round be defined as the interval ω where swaps are considered and at most one swap is executed. The swapping mechanism complexity consists of two steps, one involves determining the swap for the current round, and the other involves determining the number of rounds until optimality is reached.

Theorem. *The swapping mechanism is $O(n^2)$ complexity for each round.*

Proof. Since each agent must examine the δ of each other agent, a single round must take $O(n^2)$ computational complexity. □

Theorem. *The number of rounds needed to reach optimality cannot be polynomial in the number of agents.*

Proof. Suppose the number of rounds was $O(n^p)$. Then we would be able to take any random initial assignment and begin swapping until optimality was reached. This would take $O(n^{p+2})$. Since we know that the general linear sum assignment problem is NP-hard, there is a contradiction. Thus the number of rounds cannot be polynomially bounded. □

The fact that the number of rounds in not polynomial does not inhibits the swapping mechanism's usefulness. By initially using well known approximately optimal assignment algorithms, the number of swaps required can be reduced significantly. The swapping mechanism also provides an anytime algorithm for improving and maintaining the optimal assignment problem which can be useful for both static and dynamic assignment problems.

3.6 Distributed Decision Computation

Since the number of rounds required is non-polynomially bounded, we will next consider the use of distributed computing for reducing the computational complexity of each round and reducing the number of rounds. Parallel computation is a natural paradigm for multi-agent systems since agents are often distributed across several computing devices.

3.6.1 Distributed Swap Opportunity Computation

The swap opportunity computation between each pair of agents can be distributed across agent processes. A single agent can be responsible for examining the swap potential, δ, with each other agent's task. The agent determines the best agent for it to switch with, or decides that its current task should not be switched. There will always be one swap that will be the global best swap at the current time. When both agents decide that the best swap is to swap with each other then this swap must be the global best swap. Thus, each agent informs the agent with whom it has the best swap opportunity. When two agents receive each other as best swap pairs, then these two can execute the swap.

Thus each agent only does O(n) comparisons, and each agent only has to send one message to the agent's best swap partner. Thus only O(n) messages are sent throughout the system. This optimization provides a simple yet effective means of computing the swaps, which could allow increasing the rate of swapping by decreasing the interval time ω.

In order to accomplish distributed swap computation, each agent must have a synchronized view of the current round. This can be achieved through standard methods for creating synchronized distributed networks (for example, see [28] chapter 12).

3.6.2 Parallel Rounds

The synchronized view of the distributed computation can be broken apart into asynchronous steps. Rather than finding the best or first swap opportunity during a round, agents can initiate swaps in parallel. Each agent computes the best (or the first) objective function increasing swap and coordinates the swap with the other agent. This allows up to $n/2$ swaps at the same time. This optimization can drastically decrease the overall time necessary for convergence. However this increased parallelism requires careful execution and coordination between agents.

Theorem. *A pair of agents can swap at the same time another pair of agents swaps without making the assignments invalid or reducing the objective function.*

Proof. By the definition of δ a swap only affects the benefits from the two agents involved in the swap. In a linear sum assignment problem, changing the effects of two terms will not affect the way the other terms are interpreted. Since swaps with two pairs in parallel only affect the terms associated with the pair, the aggregate affect of parallel swaps will not reduce the objective function. Since all assignments are still maintained during a parallel swaps, the assignments cannot explicitly become invalid or incomplete. □

During execution of a swap some care needs to be taken to maintain consistency. While two agents are executing a swap, the agents cannot consider swaps with other agents, since that could potentially create incomplete assignments. This is not difficult to ensure, since each agent can simply reject swap requests by other agents during an interchange of tasks.

When multiple agents want to swap with an agent, the agent can decide to swap with the agent with the largest δ increase. Once the agent decides which other agent to swap with, the other agents must recompute their next swap, since the agent's new task may not have a $\delta > 0$.

When an agent requests a swap with another agent, the agent will wait for a reply for a swap request. Since the agent will be waiting for a reply, the agent cannot make progress when in this state. This can cause agents to

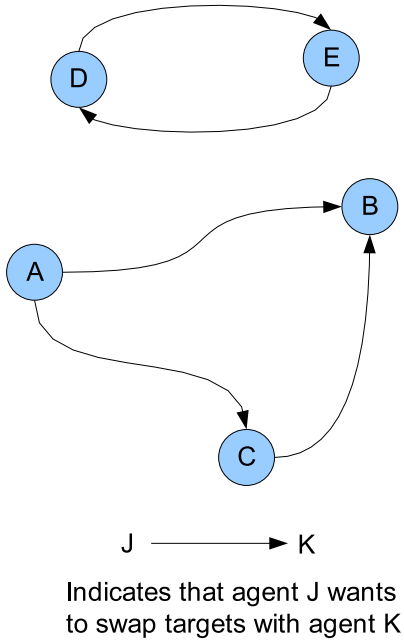

Figure 3.2: Parallel round cyclical preference problem

create circular links of swap requests, which leads to a deadlock condition since all agents wait for the other agents to reply with their swap request. This deadlock is not caused by traditional resource allocation problems; this deadlock is a result of communication requirements. However, the result of the deadlock is still the same condition of no progress. We illustrate this with an example depicted in figure 3.2. If there are three agents, A,B, and C, that want to swap so that A swaps with C, C swaps with B and B swaps with A, then a waits-for cycle is created which causes deadlock since each agent will expect a reply from another agent.

When resolving the deadlock, at most one of the swaps in the cycle can execute and all other swap requests must be cancelled which should force the agents to recompute their best swaps. For example, if A and B swap immediately and then B and C try to swap, then C will in effect be swapping with A. The swap between C and A might not always increase the objective function, so not resolving cycles appropriately can cause the objective function to decrease.

Theorem. *There can never be a cycle with n agents involved, when each agent only considers it's best swap opportunities.*

Proof. There exists an ideal swap which is the swap opportunity with maximum increase to the objective function. The ideal swap involves two agents,

which means that the two agents will desire a swap with each other. Thus, the two agents in the ideal swap cannot take part in any cycle since they will always want to swap with each other. This insures that the number of agents involved in a cycle is less than n. □

The previous theorem ensures that the minimum number of swaps that are performed during a round when using the best swap partner variant is one, since the ideal swap is always performed. This guarantees that each round will improve the objective function as long as any cyclic requests are handled without adverse effects. Since cycles can still cause deadlock, we consider deadlock resolution ideas.

- Since the deadlock here is analogous to other deadlock situations, standard deadlock resolution algorithms can be used. One example is the global-marking algorithm which uses snapshots to detect deadlocks. See section 10.4 in [28]. Once a deadlock is detected then all agents in the cycle can be forced to cancel requests and find a new best swap. Since snapshots can be expensive to maintain, the deadlock detection algorithm offers limited scope of use.

- A simpler solution is to use timeouts of swap requests. Cycles may be rare, since the agents must propose swaps at exactly the same time. If the swap requests are staggered even slightly than one swap will already be executed or in progress and the other agents will be forced to recompute the best swap opportunity with the agents new task. When an agent requests a swap and a cycle is created, then the agent will eventually timeout and recompute the best swap. Since the cost of actually detecting deadlocks is expensive and cycles are rare, this method suffices.

Once a cycle is resolved the agent must once again consider a swap partner. The algorithm should prevent the agent from wasting time continuously requesting a swap from an agent that has already rejected an earlier swap request. An easy method to resolve this conflict is to introduce a random component such as random selection or random wait.

- As described in section 3.8, the random selection method allows an agent to randomly consider a swap partner. This allows the agent to quickly request a swap and if it does not work, try another agent.

- By waiting a random period of time after a swap request, which can be approximately equal to ω, an agent can ensure that the environment

has changed due to it's dynamic nature or due to the completion of swaps. Since the environment has changed the agent can once again continue to consider all agents as potential swap partners.

3.7 Multiple Agent Requirements for Tasks

Not all systems will require exactly one agent to perform one task. There may be circumstances where multiple agents are needed to service a task.

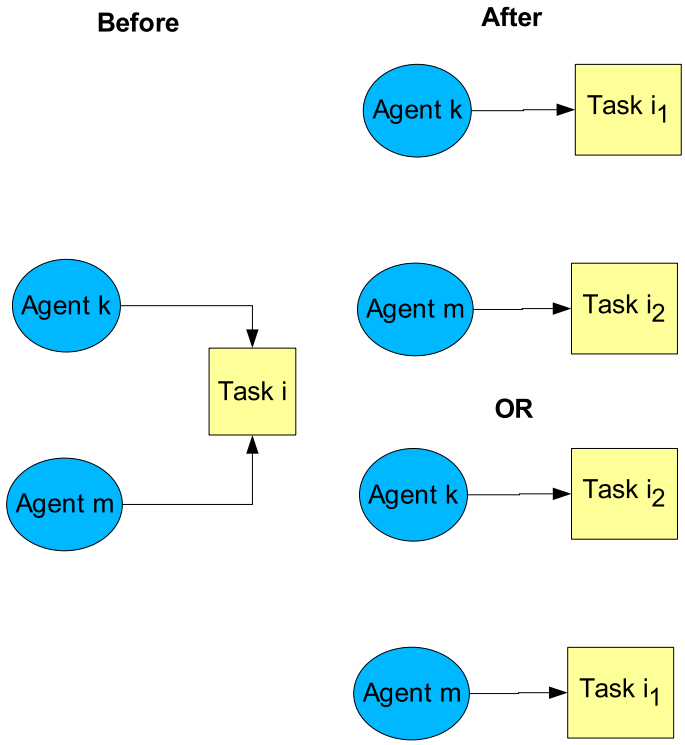

Figure 3.3: Multi Requirement Task modelling resolution. Here task i has a requirement of 2 agents.

This more generalized model does not require drastic changes to the swapping mechanism. A task with multiple requirements can simply be modelled as multiple identical subtasks with one agent requirement for each subtask. See figure 3.3. Swapping will not occur within agents that are assigned to the same complex task, since the δ will not improve from swapping. The characteristics of the task is duplicated so the benefit from swapping is always zero. This model also allows the agents to swap with other agents outside of the multi requirement task.

3.8 Randomized Decision Computation

In the original formation of the swapping mechanism, an agent tries to find the benefit maximizing swap among all other agents. A possible extension is to use randomized selection, where an agent randomly selects another agent to consider swapping with.

Since only one agent is compared at each time interval, the decision computation is only $O(1)$. The probability of finding an appropriate swap opportunity is $O(\frac{1}{n})$, since there could be only one agent with a beneficial swap and n agents to choose from. Within $O(n\omega)$ time, all agents will be considered for a swap, so the probability of making a needed swap in the time interval is $O(1)$. Thus, if the benefit function changes slowly then the randomized swapping computation will be an efficient mechanism.

This method combined with parallel swaps can increase the convergence of the objective function even more since each agent will be able to consider many swaps within the same time interval ω.

3.9 Rate of Change

We now extend our discussion to environments which can be changing during a swapping round. In a dynamic setting, the benefit function for each agent-task pair is continuously changing due to properties of the system. Let $\Delta b_{i,j}$ be the rate of change of the benefit function for agent i to service task j. Since we have discussed several variants of the swapping mechanism, we will generalize ω as the maximum interval where the expected probability of determining a needed swap is O(1). For example, when using parallel randomized decision computation, the generalized ω would be the time between swap considerations $\times n$. When using the generalized ω, the minimum rate of swapping is $\frac{1}{\omega}$.

To define some vocabulary for the dynamic nature of the multi-agent system, we generalize the rate of change of the system based on the individual $\Delta b_{i,j}$ pairs.

$$\begin{aligned} \forall i, k \in V, \forall l, j \in W \\ \delta = b_{il} + b_{kj} - b_{ij} - b_{kl} \\ \Delta\delta = \Delta b_{il} + \Delta b_{kj} - \Delta b_{ij} - \Delta b_{kl} \end{aligned}$$

Here $\Delta\delta$ is the increase in the δ potential improvement in the objective function per second (or the same time interval as used in the Δb equations.

Let γ be the number of swap opportunities created after t seconds.

$$\forall i, k \in V, \forall l, j \in W$$
$$\gamma = \#(\delta + \Delta\delta t > 0)$$

Here we are assuming $\Delta\delta$ stays constant for the time interval t.

The swapping mechanism maintains an optimal assignment when new swap opportunities are created less frequently than they can be executed. This helps develop some insight into how frequently new swaps must be executed. We could try to define ω such that the number of swaps created after t time, which is γ, is less than $\frac{t}{\omega}$. Since swaps are performed faster than swap opportunities are created, the optimal assignment can be maintained.

3.9.1 Thrashing

Since the benefit function for any agent task pair can change, a δ can move rapidly between positive and negative values. This will result in swap opportunities being taken and then immediately causing another swap. This could cause two agent task pairs to continuously swap their tasks resulting in no progress. The potential thrashing between tasks is highly dependent on the system characteristics. One way to reduce the thrashing is to limit the time between consecutive swaps between the same pair. This could prevent situations where there may be brief periods of thrashing followed by well defined long term optimal assignments.

3.9.2 Distance Example

The benefit function can be based on the distance between the agent and the task. Such situations are common in vehicle routing problems. The δ is changing based on the velocities of the agent and the task. The $\Delta\delta$ is thus the rate of change in the distance between the agent and the task based on the sum of their velocities. Since we know $\Delta\delta$, we could then assume that the velocities are constant for small intervals and compute γ for that small interval. This would provide us with an estimate of the number of swaps we need to perform in the small interval. ω thus needs to be carefully selected by changing the frequency of the swap decision computation, so that each swap that is necessary can be executed.

Chapter 4

Swapping Mechanism for Selfish Agents

In multi-agent systems, agents are often modelled as selfish agents. When assigning tasks to selfish agents, game theoretic mechanism design is a useful paradigm for analyzing algorithms. The static assignment problem for selfish agents has been solved by Nisan and Ronen using the centralized *MinWork* mechanism [20]. The mechanism results in agents truthfully revealing their benefit for servicing each target and providing a n-approximate optimal assignment by using a Vickrey auction [20].

Our proposed solution to dynamic assignment among selfish agents uses a static assignment algorithm followed by a series of swaps. A game theoretic mechanism, such as the *MinWork* mechanism, can be used to generate the initial assignment. Once the assignment is created, the swapping mechanism can be used to induce selfish agents to maintain optimality in a dynamic environment. The swapping mechanism is adapted for selfish multi-agent systems by incorporating incentive payments and modifying the decision criterion.

The analysis and implementation considerations for cooperative agents are generally applicable to selfish agents, thus in this chapter, we analyze the characteristics unique to selfish agents.

4.1 Game Theoretic Problem Formulation

Game theory is a mathematical framework designed for analyzing the interaction between several agents whose decisions affect each other. An interactive situation is described as a game in which players (agents) choose actions to create preferred outcomes. Each agent cannot unilaterally control the outcome of the game, thus each player must generate strategies based

on the likely rational choices of other agents. Mechanism design is used to create a game such that agent's rational choices satisfy global solution concepts.

Games can be classified among two broad forms: normal and extensive form. A normal form game is a single round of actions, such as a heads-or-tails game. Whereas an extensive form game consists of multiple rounds of actions, such as chess.

The dynamic assignment maintenance mechanism can be formulated as an extensive form game, where each agent is trying to maximize it's personal benefit at each step of the game. Thus each agent tries to be assigned to tasks with the most benefit (or utility) through each point in time.

4.2 Decision Criterion

In our original swap decision criteria

$$\delta = b_{il} + b_{kj} - b_{ij} - b_{kl}$$

if $\delta > 0$, then the swap is accepted. If $b_{ij} > b_{il}$, then agent i has no incentive to swap, since the swap is reducing it's personal benefit. Thus, the original swap decision criteria is no longer an applicable means of maintaining global optimality of the objective function.

4.2.1 Mutual Benefit Maximization

The decision criterion can be modified to

$$b_{il} - b_{ij} > 0 \quad \&\& \quad b_{kj} - b_{kl} > 0$$

then preform a swap. This results in both agents gaining a mutual benefit. Thus, both agents have incentive to perform a swap. With this decision criteria, the global objective function might not be optimal since there may be maximizing opportunities from a rejected swap which requires one agent to reduce it's benefit.

4.2.2 Incentive Payments

A mechanism must be created to induce agent i to swap with agent k even when $b_{ij} > b_{il}$. Agent k can provide some bonus for swapping tasks by including a payment p. The payment $p = b_{ij} - b_{il}$ from agent k to agent i

will properly induce a swap, since the swap will not have an adverse effect on agent i or agent k.

4.3 Solution Concepts

When the agent decisions promoted by the mechanism are optimal given the other agent's decisions, then the system is said to be in *Nash Equilibrium.*

Theorem. *The swapping decision mechanism results in a Nash equilibrium.*

Proof. For agent i, receiving the payment will result in no loss, which will make the swap decision an optimal strategy (some arbitrary small ϵ increase in the payment will result in a small net gain which solidifies the optimality of the swap decision). For agent k, the payment will still result in a net gain since $\delta > 0$, which implies that the difference between the benefit gained by agent k must be larger than the benefit lost by agent i. Thus, agent k's optimal strategy at any step in the game is to accept a swap opportunity defined by the incentive payment decision criteria.

When mutual benefit maximization decision criterion is used, by definition both agents benefit from the swap. Thus, all swap opportunities that match the decision criteria are always accepted. □

When all agents cannot increase their utility without reducing another agent's utility, the system is said to have reached *pareto-optimality.*

Theorem. *The swapping mechanism for selfish agents reaches pareto-optimality.*

Proof. By definition a swap is always performed when there is a benefit maximizing opportunity for both agents, either by payment or mutual benefit decision criterions. Thus, swaps continue to increase the utility of each agent. When no more swap opportunities exist, the utility of an agent cannot be increased further without reducing the utility of another agent. Thus, the swapping mechanism is pareto-optimal. □

There are two possible cases for the swapping mechanism. One, the $b_{ij} : \forall i, j$ is known to all agents. In this case, each agent will always know exactly when a swap is beneficial to both parties and thus the swapping mechanism with payment or mutual benefit can be used. Two, the benefits for other agents are unknown and must be queried. Here, a mechanism must be designed so that each agent truthfully reveals their benefit for servicing a target. Creating a strategy-proof mechanism for benefit revelation in the swapping mechanism is left for future study. However, there are still

many systems which have transparent benefit functions. For those cases, the proposed swapping mechanism is appropriate.

4.4 Swapping with Coalitions

In multi-agent environments, coalitions of selfish agents are often formed in order to maximize payoff by aggregating the talents of individual agents. Coalitions are cooperative amongst its members but competitive amongst other coalitions. Dealing with coalition based environments requires a mix of cooperative and selfish swapping strategies. When assigning tasks within a coalition, the agents can use cooperative swap decision criteria, but when considering swaps outside of the coalition, agents can use selfish swap decision criteria. See figure 4.1.

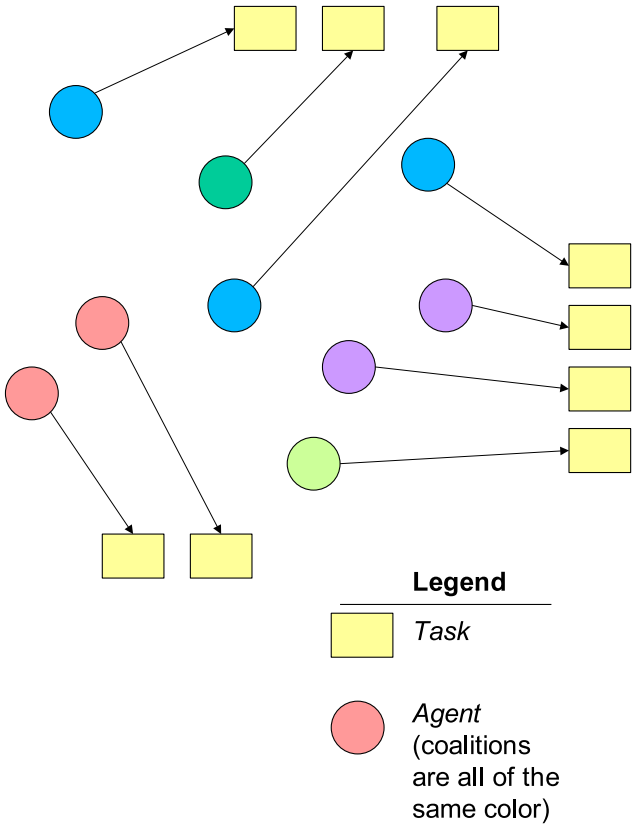

Figure 4.1: Coalition based multi-agent systems can use cooperative swapping mechanisms for intra-coalition optimization and can use selfish swapping mechanisms for extra-coalition optimization.

Chapter 5

Related Work

There are several approaches that use swapping for task assignment. We highlight the literature and define the context for this work. We also present several approaches used to solve the Dynamic Assignment Problem.

5.1 Contract Net Protocol: Swap Contracts

The Contract Net Protocol specifies the interaction between agents for competitive negotiation through the use of contracts [26]. Each agent can formulate a bid for each contract announcement received; the contract will be awarded to the agent which sends the best bid. The Contract Net Protocol is a general framework which has many applications including task assignment where a contract denotes an exchange of a task for money.

Swapping was introduced into the Contract Net Protocol with a swap contract [15, 16]. Agents announce tasks they are willing to exchange and bid for tasks that increase their personal welfare.

Sandholm et al. further extended the work by incorporating three contracts for task assignment [25, 23, 4]. The most common contracts are contracts in which only one task is considered at a time. These O-contracts are conducted between two agents where one task is transferred from one of the agents to another. Cluster contracts (C-contract) allow the agents to exchange more than one task in each contract. In a swap contract (S-contract), one agent gives one task to another agent and at the same time it receives a task from the same agent. A side payment may be paid between the agents to compensate the party that is worse off after the transfer of the task [5].

When exchanging a cluster of tasks the bundles can be judged on its temporal and spacial dimensions [14]. The spatial distance between two tasks is defined as the spatial distance between the two resources (or re-

source types) on which the tasks must be executed. Clustering tasks on the temporal dimension means grouping the tasks which must be executed within time windows that are near by. The clustering of tasks using these dimensions improves the success rate of negotiating cluster contracts.

The agents considered in the swap based contract net protocol are self-interested and bounded rational. Each agent may have different skills and cost evaluation functions. Thus, the utility of a given task may differ substantially for the different agents.

The swap contract approach is quite similar to the swapping mechanism. The swap contract analysis in the literature has been focused on allocating tasks in a static environment. Whereas we focus on the role of swapping for adaptive maintenance of optimal task assignment in dynamic environments. We explicitly incorporate the rate of change into the properties of the swapping mechanism. However, we do this in a more limited context than the swap contracts approach. The generic negotiation based approach results in minimal assumptions about the utility function or objective function. In our work we restrict our discussion to quasi-linear utilities and linear sum objective functions. The contract approach also allows for more variety of swapping using the O, S and C contracts, whereas we focus on a single task being swapped for another task. The swap contracts also optimize the swap contract negotiation with spacial and temporal clustering of tasks.

We also go into more details of the implementation considerations when parallel swaps are made. The swap based contract net protocol is an abstract notion of a negotiation process and does not state the decisions surrounding the negotiation initiation and execution.

Future work may entail incorporating our swapping mechanism ideas into the context of swap contracts, since the swap contracts offers a more general framework for negotiation with minimal assumptions.

5.2 Interchange Heuristic for the Quadratic Assignment Problem

A commonly employed heuristic is interchange, where the basic idea is to juggle with the current solution to see if we can greedily improve it by interchanging(swapping) two assignments. A good example of the use of interchange methods occurs in solving the Quadratic Assignment Problem.

The Quadratic Assignment Problem (QAP) is a classical combinatorial optimization problems and is widely regarded as one of the most difficult problems in this class. Given a set $N = 1, 2, ..., n$, and nxn matrices $F = f_{ij}$,

$D = d_{ij}$, and $C = c_{ij}$, the QAP is to find a permutation ϕ of the set N which minimizes

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\phi(i)\phi(j)} + \sum_{i=1}^{n} c_{i\phi(i)}$$

As an application of the QAP, consider the following campus planning problem. On a campus, new facilities construction is being planned and the objective is to minimize the total walking distances for students and staff. Suppose there are n available sites and n facilities to locate. Let d_{kl} denote the walking distance between the two facility sites k and l. Further, let f_{ij} denote the number of people per week who travel between the facilities i and j. Then, the decision problem is to assign facilities to sites so that the walking distance of people is minimized. We will denote c_{ik} as the cost of erecting facility i at site k. Each assignment can be mathematically described by a permutation ϕ of $N = 1, 2, ..., n$ such that $\phi(i) = k$ means that the facility i is assigned to site k. The product $f_{ij} d_{\phi(i)\phi(j)}$ describes the weekly walking distance of people who travel between facilities i and j. Thus, the problem of minimizing the total walking distance reduces to identifying a ϕ that minimizes the function z defined above [3].

One of the few known methods of solving the QAP is to use the interchange heuristic similar to the one described in this work [2]. Unlike previous work using the interchange heuristic, this work is defined in a dynamic context with applications specific to multi-agent system requirements.

5.3 Dynamic Programming

One method of approaching the dynamic assignment problem uses dynamic programming methods. One can determine the optimal action at each stage by calculating the objective function for all possible states at all possible times recursively [19, 22]. However the number of states increases exponentially with the number of agents and tasks, making traditional applications of dynamic programming intractable. Forward dynamic programming methods [10, 9] help mitigate the state space problem by using simulation and Monte Carlo sampling, rather than explicitly calculating the objective function for all possible states in a backwards manner. However, the state space of a dynamic assignment problem is still too large for forward dynamic programming methods to handle, and the challenge of estimating the objective function for a large number of states remains as well.

5.4 Multi-Stage Linear Programming

A second class of techniques uses multi-stage linear programs to solve the dynamic assignment optimization problem. Scenario methods attempt to explicitly enumerate the space of possible outcomes and solve the large-scale LP [17]. Often the dynamic assignment problem LP is modelled as a stochastic LP by defining probability functions for the dynamic parameters, and using stochastic LP optimization methods [17]. These methods still require computationally intensive algorithms and enumerating or understanding the random nature of future outcomes. Thus these methods are not suited for computing assignments in an on-line, dynamic fashion.

Dynamic methods of computing multi-stage LPs based on Bender's decompositions [11] are more computationally acceptable. They use Monte Carlo sampling to generate Bender cuts to approximate the impact of decisions on future time periods. This general approach suffers from extremely slow convergence with even a modest number of dimensions [13].

5.5 Markov Decision Processes

The dynamic nature of the assignment problem can also be formulated as a Markov Decision Process [27]. The exogenous information process is modelled stochastically, and dynamic programming is used to incorporate approximations of future behavior into a solution to the assignment LP. This approach is the most reasonable method of solving the dynamic assignment optimization problem. However, it requires finite predictable attributes of agents and tasks.

Chapter 6

Simulation Studies

The swapping mechanism described above has been used in a simulation based study to examine the usefulness of the swapping mechanism. The task assignment problem used in the simulation is a specific to coordinating routing of cooperative Unmanned Aerial Vehicles to targets [21, 18].

6.1 UAV Robot Simulation Objectives

The mechanics for Unmanned Aerial Vehicles[UAV] is beginning to mature, but effective ways of coordinating their behavior has yet to be examined. The DARPA TASK project is trying to determine coordination algorithms for a set of UAVs which need to service targets. The framework is generalized so that the notion of a target and servicing the target can be anything from search and rescue to surveillance.

A set of targets are scattered in the mission area. Each target moves in a predetermined path, which is unknown to the UAVs. Targets require a set of UAVs to service them, if the number of required UAVs are not present then no utility is gained from servicing the target. Each target differs in their utility and requirements. In our demonstration, a target is serviced by physically reaching the target.

A group of UAVs roam in the mission area looking for targets. Upon detection of targets, UAVs coordinate to form groups to service them subject to:

$$Max \sum_{i \in servicedtargets} utility_i - \sum_{i \in servicedtargets} \sum_{j \in group(i)} Cost_{ij}$$

These UAVs can be modelled as cooperative agents in a MAS, whose goal is to maximize the observed utility acquired. Essentially this problem is a dynamic task assignment optimization problem, where each UAV (agent) must be assigned to a target (task) to service.

6.2 The Auction Algorithm

We extended a Linear Sum Assignment solution [8, 7, 6] which uses the relationship between the primal and the dual of the assignment problem formulation to generate a near-optimal assignment.

The primal of the assignment problem is the LP model originally proposed in section 2.1. For each target j, it is useful to introduce a dual variable p_j called the *price* of j. We call the vector with coordinates p_j, $j = 1, ..., N$ a *price vector*. For a given price p the scalar $\pi_i = max_{j \in A(i)} b_{ij} - p_j$ is called the *profit margin* of person i corresponding to p. A dual problem to the assignment problem is

$$min \sum_{i=1}^{N} \pi_i + \sum_{j=1}^{N} p_j$$
$$s.t. \pi_i + p_j \geq b_{ij} \quad \forall i, and j$$

For a given price vector p, the cost of the dual LP is minimized when π_i equals the maximum value of $b_{ij} - p_j$.

This problem formulation can be solved using an auction based algorithm. Let :

u_j : Utility of target j

req_j : Number of Required UAVs to service target j

asn_j : Number of Assigned UAVs to target j

c_{ij} : Cost to service j (path length)

p_j : Current price of target j

UAV i's bidding strategy :

- Calculate the benefit b_{ij} of servicing target j.

$$b_{ij} = \frac{u_j}{req_j - asn_j} - c_{ij} - p_j$$

- Find j_{i1}, j_{i2} as follows

$$j_{i1} = \arg\max_{j \in targets}(b_{ij})$$
$$j_{i1} = \arg\max_{j \in targets^j \neq j_{i1}}(b_{ij})$$

- Bid for target j_{i1} with value $\pi_{j_{i1}}$

$$\pi_{j_{i1}} = a_{ij_{i1}} - a_{ij_{i2}}$$

Auctioneer for target j's strategy :

- Keep the best bids and propagate the new price (p_j = the $(req_j)^t h$ highest received bids) until the end of the auction round.
- If there are not enough bids then create better incentive for UAVs to bid for target j by reducing its price

6.3 Implementation Architecture

The TASK project included two demonstrations of the framework, one that was physical and one that was simulated. In both demonstrations, the UAVs and targets were modelled as actors in order to facilitate concurrent computation for agents [1, 18]. The physical demonstration used robot cars to represent the capabilities of UAVs and targets. We used two models of robot cars with IPaqs on each one to act as the control system. The first model named Garcia was used for the UAVs and the second model named PPRK was used for the targets. The targets were slower moving than the UAVs to insure that each UAV had the capability to service each target. A set of UAVs serviced a target by surrounding the target. The first UAV to reach the target caused the moving target to become stationary, and when the other UAVs arrived, the target became serviced.

The simulated demonstration modelled the behavior of the cars in order to visually show the decisions made by each UAV, without needing extraneous equipment. The simulated demonstration and the physical demonstration used the same code base in order to maximize reuse of code and facilitating rapid testing using the simulator.

Figure 6.1 shows the architecture of the shared code between the simulated and real demonstrations. Each UAV communicated messages with other UAVs using the input and output handlers to coordinate their actions. Once a decision was made the movement commands were sent to the simulated motion controller or robot car motion controller. UAVs gathered information about the world state using a vision system in the real demonstration and a simulated vision in the simulator. The simulated motion controller would simply send updated location information to the simulated vision which would propagate this new information to each UAV. The real

vision system incorporated cameras with imagine recognition to update location information and relay this new information to the UAVs. Thus, a cycle is created where a UAV coordinates to make a decision which impacts the world (be it simulated or real) which may require new decisions.

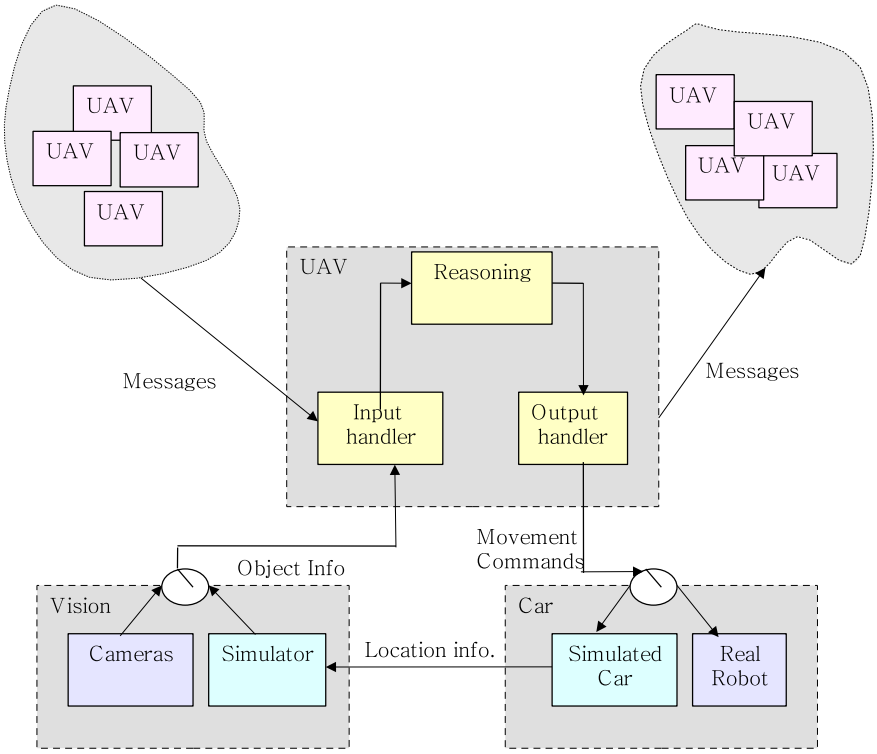

Figure 6.1: Hardware/Software Shared Code

Figure 6.2 shows the UAV reasoning architecture. The *World Model* incorporates the UAVs beliefs of the world environment such as the location of targets and UAVs. Each module makes decisions based on the current *World Model* state, which is updated by the *ObjectInfo* message sent by a real or simulated *Vision* module. The *Coordination Module* performs the auction based assignment optimization approximation by bidding for targets. Once an assignment decision has been made the *Coordination Module* uses the swapping mechanism to improve the assignment decision. Once a group is formed to service a target, the *Target Handling* module plans a route to the target and sends motion commands to a real or simulated motion controller. The *Target Handling* module also communicates with the group members to coordinate their movement to the target. Once a target is serviced the *Target Handling* module sends messages to notify other UAVs of the completion of the task.

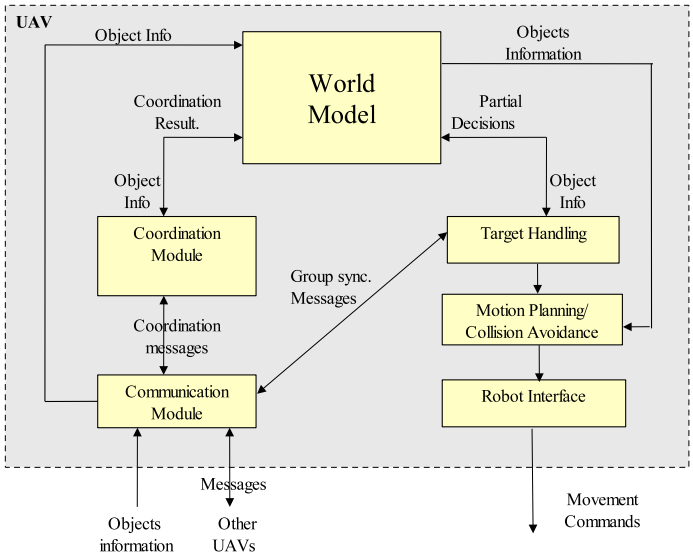

Figure 6.2: UAV Agent Architecture

Figure 6.3 depicts the auction algorithm for assigning free tasks. Targets must be auctioned by an auctioneer which is controlled by a UAV acting on behalf of a target according to the formulation described in section 6.2. Each UAV is responsible for performing the auctioneering duties for a evenly distributed set of targets. The auctioneer announces itself as the auctioneer for a particular target so each UAV will know who to send bids to. The auctioneer collects as many bids as it can within a auction round which is set experimentally. Each time a bid is received the price of the target is updated, which causes new bids. When the auction ends the target decides to either accept the current bids or reduce the price of the target if there is a lack of bids.

Each UAV determines if there are targets with requirements that can be satisfied by the UAV and it's teammates. The UAV then bids for the target based on it's marginal benefit formulated in section 6.2. Once the target accepts the bid, the *Coordination Module* notifies the *Target Handler* of the new assigned target and begins chasing the target with the formed group.

The *Swapping Mechanism* is embedded in the *Coordination Module*. The *Swapping Mechanism* determines if the current target assigned to it can be swapped by any other UAVs assignment. When it finds a global reduction opportunity, it performs a swap of assignments. The synchronization of the groups must be maintained carefully by notifying each group member of the intensions of the swap so each member knows it's new group member. Once the swap is executed the *Target Handling* module starts chasing the new

target with it's new group.

6.4 Experiments and Results

The experiments were run on a single machine in the simulator. A mission is a particular set of UAVs with starting locations and Targets with starting locations, movement patterns, utilities, and requirements. Several missions with varying parameters were tested. The auction-based algorithm without swapping was used as a control group in the experiments. The auction algorithm was calibrated using a round time of 1 sec and a reverse auction price reduction of 50%. See figure 6.4 for the experiment metrics.

Mission 1 was 6 UAVs versus 9 targets. Each target had varying utilities and requirements. When swapping was used the mission was completed faster and with less variance. Swapping made the biggest difference when the UAVs were poorly assigned to targets, since the UAVs quickly swapped into more optimal assignments.

Mission 2 was 6 UAVs versus 9 targets, with each target having varied utilities and requirements. This mission had a target with very high utility furthest from the group of UAVs. When the mission starts one UAV is assigned to the high utility target, but the UAV always has difficulty maneuvering to the assigned target. After the other UAVs have serviced other targets, other UAVs are often in a better position to service the highly valued target. When swapping was enabled the highly valued target was serviced efficiently since the target was assigned in a dynamic manner. Swapping produced quicker average completion with less variance.

Mission 3 was 3 UAVs versus 9 targets, with each target having the same utility and the same requirements. This mission was barely affected by swapping since the mission was very straightforward and little changed in the environment.

The data shows that the swapping mechanism is an useful way of dealing with dynamic environments in the simulated UAV coordination demonstration, since it reduces overall service time and produces more consistent results. The conditions in the UAV coordination demonstration made it easier to use swapping since the robots were moving in a steady manor and the agents were cooperative. The usefulness of the swapping mechanism is highly dependent on the nature of the tasks and the rate of changing environment variables. Under conditions where the swapping mechanism is applicable, the system designer can optimize assignments in dynamic environments.

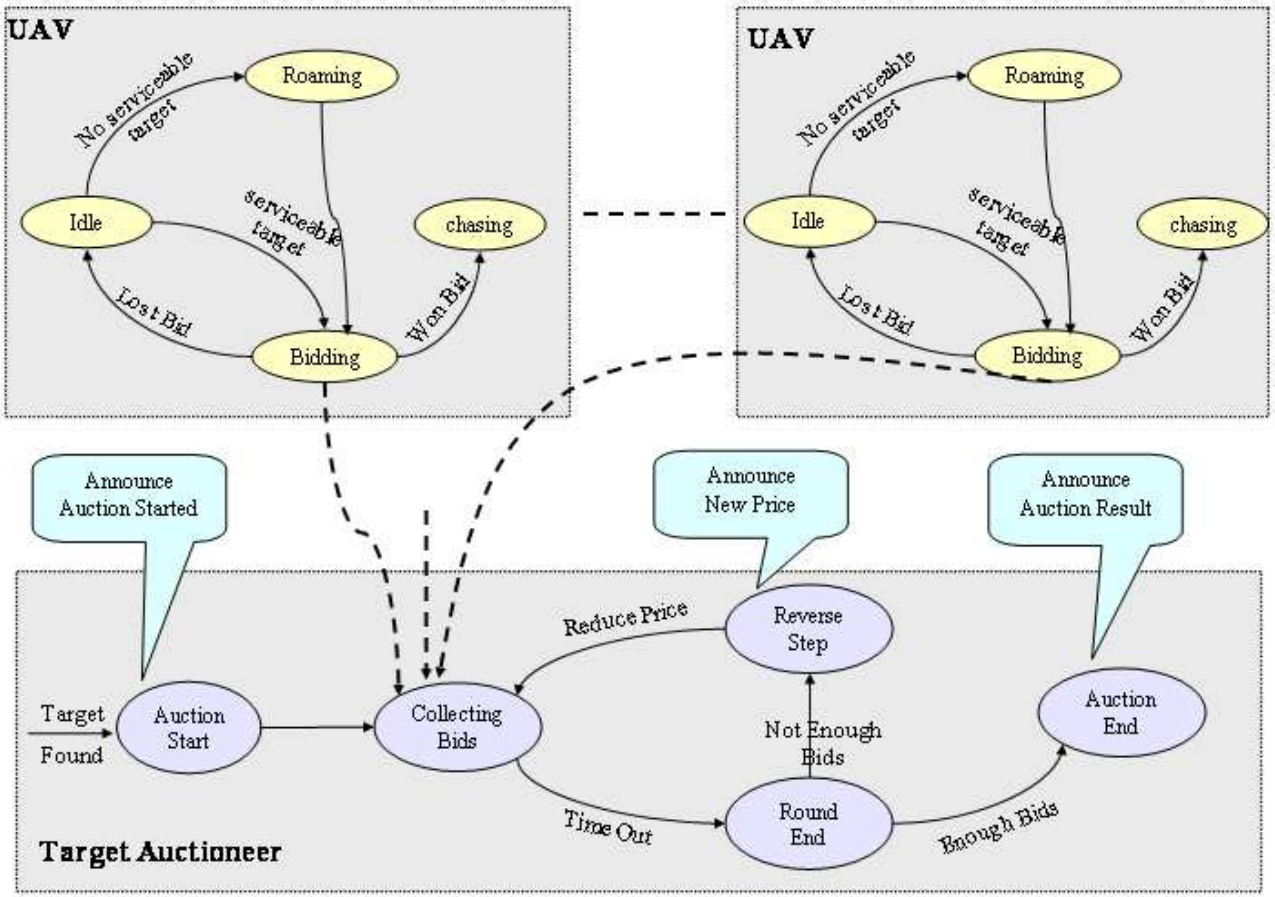

Figure 6.3: Auction State Diagram

	Mission 1		Mission 2		Mission 3	
	w/o swapping	w/ swapping	w/o swapping	w/ swapping	w/o swapping	w/ swapping
Trial 1	211	175	147	94	90	84
Trial 2	243	209	125	104	88	89
Trial 3	234	179	182	88	76	86
Trial 4	206	186	136	112	94	96
Trial 5	252	169	141	108	94	84
Trial 6	241	212	153	96	87	82
Trial 7	213	183	136	103	91	79
Trial 8	227	192	121	91	83	83
Avg completion time (s)	**228.38**	**188.13**	**142.63**	**99.5**	**87.88**	**85.38**
Std dev	**16.92**	**15.46**	**19.07**	**8.52**	**6.03**	**5.18**

Figure 6.4: Swapping Mechanism Metrics

Chapter 7

Conclusions

We have shown an approach to solving the linear sum assignment problem under dynamic conditions where the nature of the tasks and agents are continuously changing. The swapping mechanism allows each agent to swap assignments when there is an opportunity to increase the global objectives for cooperative agents and personal/global objectives for selfish agents. We include an analysis of the swapping mechanism for both cooperative and selfish agents which characterizes the complexity of the swapping mechanism under changing agent-task benefit functions. By using concurrent processing, which is often inherent within multi-agent environments, we can more efficiently compute swapping decisions.

When the speed of swapping is fast enough to cope with the maximum rate of swap opportunity generation, the system can maintain an optimal assignment. However, no matter what the rate of change is within the system, the swapping mechanism provides a simple method to cope with a dynamic environment.

The swapping mechanism also improves upon the inherent sub-optimality of traditional static assignment optimization solutions by continually seeking opportunities to increase the global objectives. The solution proposed is applicable to a wide range of scenarios where the nature of the agent utility function is quasi-linear and the optimization problem is characterized as a linear sum assignment problem. We intend to further explore the swapping mechanism by relaxing the utility and objective function restrictions and incorporating the mechanism into the contract net protocol.

This approach has been used in the Unmanned Aerial Vehicle coordination demonstration, where mobile UAVs attempt to service moving targets by coordinating with other teammates. By incorporating the swapping mechanism, we were able to optimize and decrease the variance of mission performance with marginal computational overhead. The experimental ev-

idence supports the possible usage of the swapping mechanism to manage assignment in dynamic multi-agents systems.

With the possibility of dynamic assignment optimization, agents can better explore complex environments where agents coordinate to solve sub-problems.

References

[1] G. Agha. Actors: A Model of Concurrent Computing in Distributed Systems. The MIT Press, Cambridge, MA. 1986.

[2] R.K. Ahuja, J.B. Orlin, S. Pallottino, M.P. Scapparra, M.G. Scuttella. A Multi-exchange Heuristic for the Single Source Capacitated Facility Location Problem. In *Massachusetts Institute of Technology Sloan School of Management Working Papers*. Number 4387-02

[3] R.K. Ahuja, J.B. Orlin, A. Tiwari. A Greedy Genetic Algorithm for Quadratic Assignment Problem. In *Computers and Operations Research*. Vol. 27, Issue 10, pages 917-934. September 2000.

[4] M. Andersson, T. Sandholm. Contract Types for Satisficing Task Allocation: II Experimental Results. In *AAAI 1998 Spring Symposium: Satisficing Models*. Stanford University, California. March 23-25, 1998.

[5] M. Andersson, T. Sandholm. Sequencing of Contract Types for Anytime Task Reallocation. In *Second International Conference on Autonomous Agents (AGENTS), Workshop on Agent-Mediated Electronic Trading (AMET)*. Minneapolis, MN. May 1998.

[6] D.P. Bertsekas. The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem. In *Annals of Operations Research*. Vol. 14, pages 105-123. 1988.

[7] D.P. Bertsekas, D.A. Castanon. Forward Reverse Auction Algorithm for Asymmetric Assignment Problems. In *Computational Optimization and Applications*. Vol. 1, pages 277-297. 1992.

[8] D.P. Bertsekas, L.C. Polymenakos, P. Tseng. Epsilon-Relaxation and Auction Methods for Separable Convex Cost Network Flow Problems. In *Network Optimization*, *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, New York. pages 103-126. 1998.

[9] D. Bertsekas, J. Tsitsiklis. Neuro-Dynamic Programming. In *Athena Scientific*. Belmont, MA. 1996.

[10] D. Bertsekas, J. Tsitsiklis, C. Wu. Rollout Algorithms for Combinatorial Optimization. In *Journal of Heuristics*. pages 245-262. 1997.

[11] J. Birge. Decomposition and Partitioning Techniques for Multistage Stochastic Linear Programs. In *Operations Research*. Vol. 33(5), pages 989-1007. 1985.

[12] B. Gerkey. On Multi-Robot Task Allocation. Diss. University of Southern California, Los Angeles. 2003.

[13] G. Godfrey, W.B. Powell. An Adaptive, Dynamic Programming Algorithm for Stochastic Resource Allocation Problems I: Single period travel times. In *Transportation Science*. Vol. 36(1), pages 21-39. 2002.

[14] M. Golfarelli, S. Rizzi. Spatio-temporal Clustering of Tasks for Swap-Based Negotiation Protocols in Multi-Agent Systems. In *Proceedings 6th International Conference on Intelligent Autonomous Systems*. Venice, Italy. pages 172-179. 2000.

[15] M. Golfarelli, D. Maio, S. Rizzi. Introducing Swap-Based Negotiations in the Contract Net Protocol. In *International Joint Conference on Artificial Intelligence*. Nagoya, Japan. 1997.

[16] M. Golfarelli, D. Maio, S. Rizzi. A Task-Swap Negotiation Protocol Based on the Contract Net Paradigm. In *Technical Report CSITE*. number 005-97. 1997.

[17] G. Infanger. Planning Under Uncertainty: Solving Large-scale Stochastic Linear Programs. In *The Scientific Press Series*. Boyd & Fraser, New York. 1994.

[18] M. Jang, S. Reddy, P. Tosic, L. Chen, G. Agha. An Actor-based Simulation for Studying UAV Coordination. In *15th European Simulation Symposium (ESS 2003)*. Delft, The Netherlands. pages 593-601. October 26-29, 2003.

[19] B. Lageweg, J. Lenstra, A.R. Kan, L. Stougie. Stochastic Integer Programming by Dynamic Programming. In *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, New York. pages 403-412. 1998.

[20] N. Nisan, A. Ronen. Algorithmic Mechanism Design. In *Games and Economic Behavior*. Vol. 35, pages 166-196. 2001.

[21] T. Predrag, M. Jang, S. Reddy, J. Chia, L. Chen, G. Agha. Modeling a System of UAVs on a Mission. In *Proc. 7th World Multiconference on Systemics, Cybernetics, and Informatics (SCI '03)*. pages 508-514, July 27-30, 2003.

[22] M.L. Puterman. Markov Decision Processes. John Wiley and Sons, New York. 1994.

[23] T. Sandholm. Contract Types for Satisficing Task Allocation: I Theoretical Results. In *AAAI 1998 Spring Symposium: Satisficing Models*. Stanford University, California. March 23-25, 1998.

[24] T. Sandholm. Distributed Rational Decision Making. In collection Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, Cambridge, MA. Editor Gerhard Weiss. pages 201-258. 1999.

[25] T. Sandholm. Necessary and Sufficient Contract Types for Optimal Task Allocation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Nagoya, Japan. Poster presentation. 1997.

[26] R. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. In *IEEE Transactions on Computers*. Vol. 29(12), pages 1104-1113. December 1980.

[27] M. Spivey, W.B. Powell. The Dynamic Assignment Problem. In *Transportation Science*. (to appear)

[28] G. Tel. Introduction to Distributed Algorithms. Cambridge University Press, Cambridge, UK. 2001.