RESOURCE BASED PLAN REVISION IN DYNAMIC MULTI-AGENT ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

UTKU ERDOĞDU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JANUARY 2004

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Faruk Polat
Supervisor

Examining Committee Members

Prof. Dr. Faruk Polat

Assoc. Prof. Dr. İ. Hakkı Toroslu

Assoc. Prof. Dr. Göktürk Üçoluk

Asist. Prof. Dr. Halit Oğuztüzün

Asist. Prof. Dr. Bilge Say

ABSTRACT

RESOURCE BASED PLAN REVISION IN DYNAMIC MULTI-AGENT ENVIRONMENTS

Erdoğdu, Utku

M.S., Department of Computer Engineering

Supervisor: Prof. Dr. Faruk Polat

JANUARY 2004, 66 pages

Planning framework is commonly used to represent intelligent agents effectively and to model complex behavior. In planning framework, resource-based perspective is interesting in the sense that in a multi-agent environment, exchange of resources can form a cooperative interaction.

In resource based plan coordination, each agent constructs an individual plan, then plans are examined by a central plan revision unit for possibilities of removing actions.

Domain of this work is the classical postmen domain that is also modified to have non-sub-additive property.The domain is has numerous challenges that is not considered in the original plan coordination model. Moreover, the plan coordination algorithm is used in the re-planning phase, as the environment changes through plan execution.These issues are common for the realistic environments and the details of the original plan coordination perspective are renewed to cope with these issues.

Keywords: resource based planning, dynamic multi-agent environments

ÖZ

DEĞİŞKEN ÇOKLU-ETMEN ORTAMLARINDA KAYNAK TABANLI PLANLARIN GÖZDEN GEÇİRİLMESİ

Erdoğdu, Utku
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Faruk Polat

OCAK 2004, 66 sayfa

Planlama, zeki etmenlerin altyapılarında ve bu etmenlere karmaşık davranış şekillerinin kazandırılmasında sık kullanılan bir tekniktir. Bir tür planlama tekniği olan kaynak tabanlı planlama, çok etmenli sistemlerde kaynakların değiş tokuşunun doğal ve basit bir işbirliği yapısı oluşturması dolayısıyla önemlidir.

Kaynak tabanlı planların koordinasyonu esnasında her etmen kendi planını diğer etmenlerden bağımsız olarak hazırlar. Daha sonra bu planlar merkezi bir plan gözden geçirme birimi tarafından, plandaki fazlalık durumundaki hamlelerin tespit edilip çıkarılması için incelenir.

Bu çalışmada mektup dağıtım probleminin işbirliğinin her zaman karlı olmaması için değişiklikler yapılan bir örneği üzerinde çalışılmıştır. Bu problem temel kaynak tabanlı plan koordinasyon modelinde öngörülmeyen bir çok zorluğu barındırmaktadır. Ayrıca plan koordinasyon methodu planın uygulanması esnasında ortamdaki değişikliklere göre planı gözden geçirme sorununun çözümünde de kullanılmaktadır. Bu çalışmanın esas hedefi plan koordinasyonunun bütün bu zorluklara çözüm getirebilmesini sağlamaktır.

Anahtar Kelimeler: kaynak tabanlı planlama, değişken çoklu etmen ortamları

To my beloved mother Leyla Erdoğdu (1954-2001)

ACKNOWLEDGMENTS

I would like to thank to Prof. Dr. Faruk Polat for his support and supervision in this work. I would like to thank my uncle Zeki Erdoğdu and my aunt Sevilay Erdoğdu for their support. I would like to thank Güneş Erkan, Turan Yüksel and Alper Teke for their contribution in typesetting the thesis with LaTeX. I would like to thank all friends who were beside me whenever I looked around for the last two years, namely Devrim Saran, Alper Teke, Turan Yüksel, Emre Hamit Kök, Şenol Tekdal, Servet Kızıltaş, Sedar Gökbulut, Serdar Özcan, Ayşe Akgül, Güneş Erkan, Ayça Aksu and Nezahat Gülru Üstündağ. I would also like to thank specially to Gülgün Kayın, the most special woman who gave me all the joy and energy necessary to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Autonomous agents and multi-agent systems have been interesting research areas of artificial intelligence. These areas have been studied extensively from lots of perspectives, yet there are still many open problems. Many real world problems are distributed in modelling and hence appropriate for agency modelling. Autonomy factor increases the flexibility and capacity of agents in the environment. Multi-agent architectures are quite attractive because there are many domains where different entities should interact.

Planning had been one of the major topics in artificial intelligence long before autonomous agency and multi-agency concepts were so popular. Simple logic based planners were designed for effectively producing plans at many simple domains. They were quite successful, but the complexity of the planning approaches limited the use of planning in more realistic environments.

With the increasing importance of agent based artificial intelligence, applying planning methods to autonomous agents became a hot research area. The question everybody asks is "Could we compensate for the complexity of planning using distributed nature of multi-agent architectures?"

Before multi-agent planning concepts emerged, planning research had concentrated on producing more complex planning algorithms that could generate more efficient and simpler plans. However these complex planners were quiet useless for real time systems, environments where much computational power is not available, or distributed systems.

Thus, when planning work encountered multi-agency, the focus shifted to using planning for interactions between agents and using interactions between agents for planning. Planning is one of the most promising architecture for building autonomous agents. To apply planning effectively to multi-agent systems, interactions between agents should be related to planning process.

Focus of this work is on how agents behave in dynamic or real-time environments if they use planning as main reasoning framework. A planner agent should adopt to the changes in such an environment. Classical approaches for planning generally allow incremental planning that consists of extending plan for new goals, or removing redundant actions when some goals are no longer required. But these classical approaches do not provide means for interactions between agents regarding to changes in environment.

In this work we adopted a planning approach, namely resource based planning to a dynamic environment. Resource based approach is highly appropriate for simple interactions between agents. Using this approach, we built a planning system that could use a negotiation mechanism for interaction between agents. This system, together with the negotiation mechanism, provides modifications of agent's plan as the environment changes.

Our goal is to design and implement a planning system for a dynamic variation of classical postmen domain. The dynamic variation of postmen domain is a simple domain, but many kinds of cooperation forms are possible for the domain. We would be successful in achieving our goal if the designed agents may form the possible cooperation instances, when they are provided a reasonable negotiation mechanism.

Planning is a domain-independent approach in the classical sense. We would use properties of the domain when we are implementing the system, but all algorithms and techniques can be generalized for different domains or a general planning framework. This issue is provided by simplicity and flexibility of resource based planning approach.

In this work, first general planning concepts and problems will be presented as an overview. Than we would go into details of resource based plan representation, which is the representation method we will be using in this work. In chapter 4 we discuss the postmen domain studied and reveal the details of domain description,

including complexity of generating plans in that domain. In Chapter 5 we would present the planning, plan execution and plan revision methods agents use. Then we would present examples that demonstrate how agents act in the environment.

CHAPTER 2

MULTI-AGENT PLANNING

Planning is one of the major classical problems in Artificial Intelligence. Satisfying goals in a logical and effective manner is the aim of all agent based systems. In the context of agency, planning makes use of environmental dynamics and agent capabilities.

The aim is simply generating a plan for achieving a specified goal, given the initial state and action descriptions. Planning, with this simple description, has been studied much in AI in a variety of domains with different structures.

First planners were simple logic based planners. The world state and action descriptions were all represented in first order logic domain. Unification is used to match the abstract action descriptions, that are also called operators, with the clauses representing the world state. STRIPS [1] was first successful planning system that realized a robot controller.

Other planners followed STRIPS, but the problem remained attractive since optimum, complete and efficient plans for goals were not totally possible for most of the domains studied. Besides the planning systems were much more appropriate and applicable to a great variety of domains. The planning paradigm was goal directed, similar to planning mechanism of humans. When compared with learning, search and other problem solving methods used in AI, planning were the most promising and most realistic system to be used.

The initial planners were applicable to deterministic, discrete, accessible and static environments. Thus the domains worked were very simple. It was necessary

to extend these planners to more complex domains. New planners with different features were constructed, such as non-deterministic planners that work on domains with state transitions have a probabilistic structure, hybrid planners that use other approaches to handle the dynamic, probabilistic, continuous or inaccessibility of the environment. Much research is done on planning on different domains.

When multi-agent approaches became focus of studies on problems with distributed nature, it became necessary to consider the classical planning problem and its definition with the new multi-agent system architecture. The plans produced by classical planners were considering single *plan executer* that takes actions for satisfying the universal goals. This view was insufficient for multi-agent environments where different agents have different and possibly conflicting goals. Even if all agents have universal goals, the cooperation for executing the plan can not be constructed in a straightforward and efficient manner. Various works with different motivations were carried out to adopt the classical planning perspective to multi-agent systems. Following sections summarize the most common perspectives that the works on this topic concentrate on.

2.1 Execution of Single-Agent Plans by Multiple Plan Executers

Most of the single-agent planners are *partial order planners*, which means they do not specify the full ordering relation between actions. They only specify the actions that must specifically have an ordering constraint. In plan execution, an arbitrary action sequence is generated. The generated action sequence should only satisfy the ordering constraints.

In a multi-agent environment, if the agents have universal goals and are identical, then at the generation of action sequence a cooperation schema can be introduced. We can assign actions to agents considering the ordering constraints imposed. Unordered actions are carried out in parallel, thus a simple multi-agent cooperation is achieved.

Such an approach would decrease the plan execution length apparently. However it does not affect the cost of the plan and it is only applicable to a labor-agent community described above. Moreover this approach assumes the existence of a central authority which assigns actions to agents. Such a mechanism is not always

feasible or preferable.

This approach may also be extended to multi-agent systems where agents have different goals. In such systems, existence of a central planner resolves the conflicting goals, but the agents in the environment can not be considered autonomous any more.

2.2 Multi-Agent Planning Algorithms

As an attempt to overcome the problems of multi-agent environments numerous *multi-agent planning algorithms* were proposed. The simplest variations of these algorithms were similar to the approach described in Section 2.1. More interesting ones specifically produce multi-agent plans.

A simple perspective is shown by Boutilier and Brafman [2] where they modify the classical POP (Partial Order Planning) algorithm to handle multi-agents and concurrency. The resulting planner, namely PMOP, uses STRIPS based representation schema and generates plans for more multi-agents that share the same goal. The focus on this work is on domains where agents share the same goal and must execute some action concurrently for achieving goals. The example problem is two agents trying to carry a table together with the items on it.

McDermott and Burstein proposes another planner that generate heuristic-based multi-agent plans [3]. The planner uses regression graphs to represent actions and dependencies. It also handles some simple synchronization issues about multi-agent case.

Bowling, Jensen and Veloso [4] introduce some game theoretic perspective to the problem. They formalize the planning problem as a multiple decision problem with a solution equilibrium and model the plan as state-action table that lead to equilibrium.

Ephrati and Rosenschein [5] discuss a similar approach that is much more robust then other multi-agent planners, and able to generate plans in dynamic environments and multi-agent cases where agents do not have common goals.

Multi-agent planners generally assume a central planning authority that constructs plans for agents in environment. They have the same problems with primitive single agent planners. Also since agents do not interact they are not really suitable for domains where there is no central intelligence but autonomous agents.

2.3 Combining or Refining Single-Agent Plans in Multi-Agent Domains

A central entity that generates plans for multi-agents is not always possible. Sometimes it is just not feasible because of the highly distributed nature of the problem. And mostly the agents are expected to be autonomous, which includes constructing their own plans.

In such a setting it is important to be able to extract cooperation schemas from individual plans of agents. Cox and Durfee propose a synergy agent in [6]. The autonomous agents described in that work produce hierarchical plans structured as trees, where root is the ultimate goal, branches are sub-goals and leaves are actions. The synergy agent then examines plans of different agents, determining matching sub-trees, that represent the common sub-goals and eliminate the redundant actions in these sub-trees. This schema provides elimination of multiple identical plan segments, thus provides a simple cooperation between agents.

There are different versions of this schema. Once agents in environment construct their individual plans, these plans can be examined in many ways to determine possibilities for rearranging actions and distributing actions to other agents if necessary. All these versions assume the agents share their plans with the synergy agent and accept the modifications made by the synergy agent willingly. This is the case for community-like domains where agents satisfy global goals and seek for global utility.

However most of the time agent do not willingly share their plans to external entities, especially if the agents do not have common goals. There may be privacy considerations, or passing all plan information back and forth may not be feasible for time-critical and dynamic domains. In such cases cooperation schemas that include negotiation on common goals and refining individual plans may be introduced.

Gmytrasiewicz and Durfee [7] discuss such a mechanism that concentrates on rational agency perspective. In their model the agent models the state transitions by making commitments about world and other agent's possible plans.

Kraus, Wilkenfeld and Zlotkin [8] concentrate on negotiations about which plan segment should be refined. They explore the existence of agreements with agents having different kinds of commitments about each other. Their approach is utility based.

The focus of this work will be on refining single-agent plans for multi-agent domains. De Weerdt and Van Der Krogt [9] propose a schema for plan coordination. Planning framework is modified for adopting this plan coordination schema and the task oriented domain studied. Tonino, Bos , de Weerdt and Witteveen [10] extends this study to a further point. A *resource-based planning framework* is proposed which constitutes an appropriate mechanism for plan generation and plan coordination. Details of this paper will be discussed in Chapter 3.

CHAPTER 3

RESOURCE BASED PLAN REVISION

As mentioned in Chapter 2, this work constructs a planning framework where single-agent plans are refined in order to achieve a multi-agent coordination. Instead of using a classical logic based planning framework, a resource based planning schema is used as the core of the planning framework constructed.

The resource based plan schema used is proposed by Tonino, Bos, de Weerdt and Witteveen in [10]. This chapter includes details of the schema proposed in [10] and the plan revision algorithm proposed in that work. [1]

3.1 Resource Based Plan Representation

3.1.1 Classical Plan Representation

Classical planning algorithms use logic based representation. World state is described with first order logic clauses. Given a world state, when an operator is applied, it makes some clauses invalid and introduces new clauses. Operator descriptions are parametric and these changes are represented with clause lists. Figure 3.1 is an operator description in STRIPS action description syntax. Operator description is quoted from 1992 implementation of STRIPS [11].

The operator description is parametric for the object applied. Precondition list and filter list specify the clauses that should exist in the world state for operator to be

[1] Although this chapter explains the resource based planning schema and resource based plan revision in detail, some details and formal definitions are not quoted here. Formal definitions of the concepts and much detail can be found in [10].

```
(strips-op move-block-to-table
       :preconditions  ((clear ?x)
                        (on ?x ?y))
       :filter         ((block ?x)
                        (block ?y)
                        (:not-equal ?x ?y))
       :add-list       ((on ?x table)
                        (clear ?y))
       :delete-list    ((on ?x ?y))
       :command-string "move ?x from ?y to the table"
       :variables      (?x ?y)
)
```

Figure 3.1: An Example STRIPS Operator

applied. They are syntactically equal, while filter list generally contains basic axioms about the parameters. When an operator is applied, clauses in delete list are removed from world state, clauses in add list are added to the world state.

It is not difficult to understand this choice of parameterized representation. In all planning algorithms abstract action descriptions are defined. These action descriptions should account for same kind of action applied to different objects or applied at different time instances. Unification mechanism handles the instantiation of parameters, thus the operator descriptions can be used as a template.

With a goal directed search mechanism, a planner produces a sequence [2] of operator instances producing the goal state.

3.1.2 Resource Based Schema

Resource based plan schema is an alternative to this classical schema. The underlying idea is using the add lists of operators as products. In most domains whole plan or most of the plan accounts for producing a single result. Initial world state clauses are consumed for producing this final result.

This simple idea is extended through a more complex *resource* concept, where every object, whether used or produced is called a resource [10]. This definition corresponds to clauses in a logic based plan schema. The predicate and parameters of a clause specify the properties of the clause in logic based schema. In resource based schema the *type* of the resource makes the distinction.

[2] STRIPS generates a strict sequence, while some planners output the partial ordered sequence.

By their definition, resources are not parametric. But this is a syntactic issue. Since resources correspond to objects in the domain, they have a type. Type definition of a resource denote the semantics of the resource. As they are used in plan construction, necessary parameters are embedded into resources.

In most domains resources simply follow the semantics of the domain, while in some domains resource based approach serves only as a syntactic variation.In a freight transportation domain, a truck moving from a city to another produces room for goods transported from departure to destination. The amount of produced resources is related to the capacity of the truck. This produced resource can be used in plan, for carrying goods. Freight transportation domain is an example where resources exists in the domain semantically.

In resource based planning, resources are used and produced by *skills*. Skills are correspond to operator descriptors in logical schema. They consume resources and produce new resources[3]. Skills are not also parametric. All the necessary parameters are embedded to them.

Note that the STRIPS operator description in Figure 3.1 had a precondition list and a delete list. Semantics of both these lists are handled by resources consumed by a skill. A skill requires the existence of a set of resources for application, and consumes these resource when applied. If skill requires existence of a resource, but consumption of the resource is not necessary, the resource can be reproduced by the skill after being consumed. In [10] two occurrences of the *same* resource is avoided. Thus in such situations a new resource of the same type is produced instead of producing the same resource. However a skill may be used many times in the plan. In this work this unique resource approach is relaxed and some resource types are allowed to have more than one occurrence[4].

3.1.3 Plan Representation in Resource Based Schema

In the logical framework, a plan is a collection of operator instances. An ordering is imposed on this collection regarding the dependencies (causal links). Explicit ordering constraints are also introduced for avoiding threats of some operators on other

[3] The resource and skill concepts are defined in a set theoretic manner in [10].
[4] See Chapter 5 for details.

operators[5]

In the resource based schema a relation based representation is usable. Since each resource is only used once such a schema does not contain threats. Although this kind of representation is sufficient for goal realizability and dependencies, gives little insight about the whole structure of the plan and indirect dependencies [10].

A *resource based plan graph* is a directed acyclic graph that represents a plan. It is a more effective representation for plans. Resource based plan graph contains two kinds of nodes:

- **Resource Nodes** that represent resources
- **Skill Nodes** that represent skills

Vertices of graph represent consumption of resources and production of resources by skills. Since a resource is produced and consumed once, there is one vertex directing a resource node and one vertex going out of a resource node. Skill nodes may consume and produce more than one resource. There is no vertex between two skill nodes or two resource nodes. The resource based plan graph is a bipartite graph. There are two repeating layers in the graph. Figure 3.2 shows a simple resource based plan graph.

There are resource nodes, to which no vertex directs. These resource nodes are not produced by the plan. They constitute the initial state. The nodes, from which no vertex goes out are not consumed by the plan. They are part of the goal state of the plan, or they are irrelevant side-effects of the plan.

This graph representation is superior to a simple relation representation. For example given a resource it is easy to determine skills that are applied to produce that resource. Or given a skill, we could determine the resources that would still be produced if we remove that skill. Plan revision and coordination framework of [10], which is discussed in 3.1.4, uses this graph representation as a basis.

[5] A clause may exist in the delete list of one operator instance, and in the delete list or precondition list of another operator instance.Thus applying first operator instance makes applying the second operator instance impossible. This issue is called a threat [12].

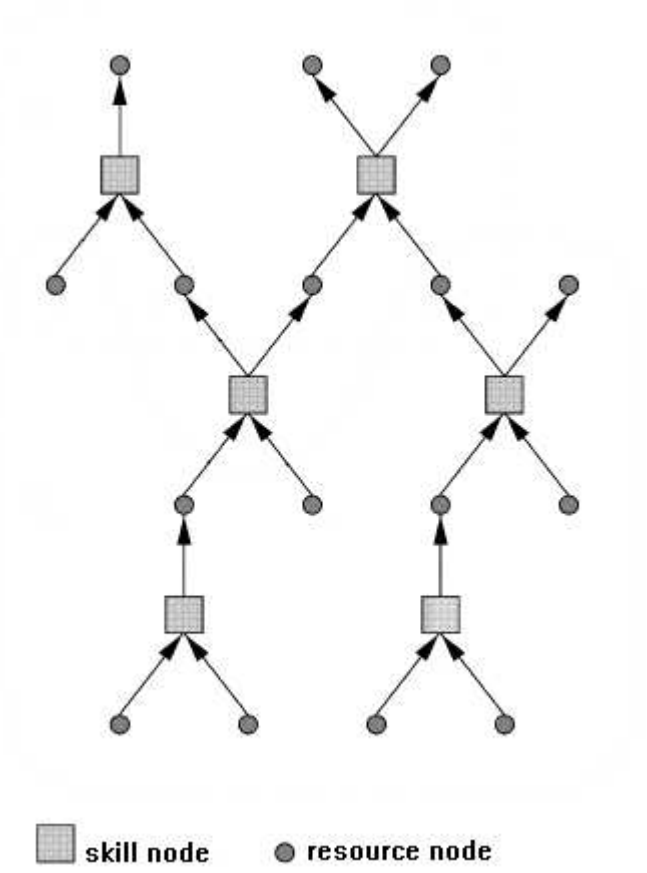

Figure 3.2: An Example Resource Based Plan Graph. This graph is a modified version of an example in [10].

3.1.4 Refinements in Resource Based Plan Graphs

A plan, which is represented in a resource based graph, can be examined for possibilities to refine the plan. The plan revision proposed in [10] uses the fact that given a skill node in the plan graph, the plan can be divided into three parts:

- A skill node given.
- Skill nodes and resource nodes that are dependent to skill given.
- Skill nodes and resource nodes that are not dependent to skill given.

if we remove the given skill node from the graph, some skill nodes are affected, some are not. This division represents this distinction. Nodes in the second group will be affected from the removal of the given skill, while nodes in the third group

will not be affected. Skills in the third group can be executed with the removal of the given node, but removal of the given node prohibits the execution of skills in the second group.

To give the formal definition of these graphs, we must define *subgraph* and *subplan* concepts [10] (pg.129).

Definition 1 *Let $P = \langle N, E \rangle$ be a plan and $N' \subseteq N$ be a subset of nodes. Then the subgraph P' of P generated by N' is the graph $P' = \langle N', E' \rangle$, where $E' = \{(n, n') \in E \mid n, n' \in N'\}$. The subplan of P generated by N' is the subgraph P'' of P generated by the smallest set N'' such that (i) $N' \subseteq N''$, and (ii) for every skill node $n_s \in N', in(n_s) \cup out(n_s) \subseteq N''$.*

Subgraph of P generated by N′ is the graph with nodes in *N′* and vertices between these nodes. *Subplan of P generated by N′* is the subgraph of *P* generated by *N′* and resource nodes produced or consumed by skill nodes in *N′*.

$In()$ and $Out()$ are two functions that denote the input and output resource set of their arguments. They are both applicable to skills and plans. When applied to a skill, they map to input/output resources of the skill. When applied to a plan they map to initial resources or goal resources of a plan.

Given these definitions, the graph parts described above are formally defined as [10] (pg. 129):

Definition 2 *Let n_s be a skill node to be removed from plan $P = \langle N_R \cup N_S, E \rangle$. Let $N^+_{n_s} \subseteq N_S$ be the set of skill nodes* dependent *on n_s, i.e., the set of skill nodes $n \neq n_s$ in P such that there exists a path in P from n_s to n. Removing n_s from P will result in two plans $P^+_{n_s}$ and $P^-_{n_s}$ where*

1. *$P^+_{n_s}$ is the* subplan *of P generated by the set $N^+_{n_s}$; and*
2. *$P^-_{n_s}$ is the* subplan *of P generated by the remaining skill nodes and the set of input resources $In(P)$, i.e., the subplan generated by the set $N^-_{n_s} = In(P) \cup N_S \setminus (\{n_s\} \cup N^+_{n_s})$.*

Figure 3.3 shows a proper division of graph in Figure 3.2 given a skill node n_s

There are some important details about this division. Most important one is the new subplans are not disjoint in the sense of resource nodes. For example resource node a is included in both subplans. The reason is a is both an initial resource of

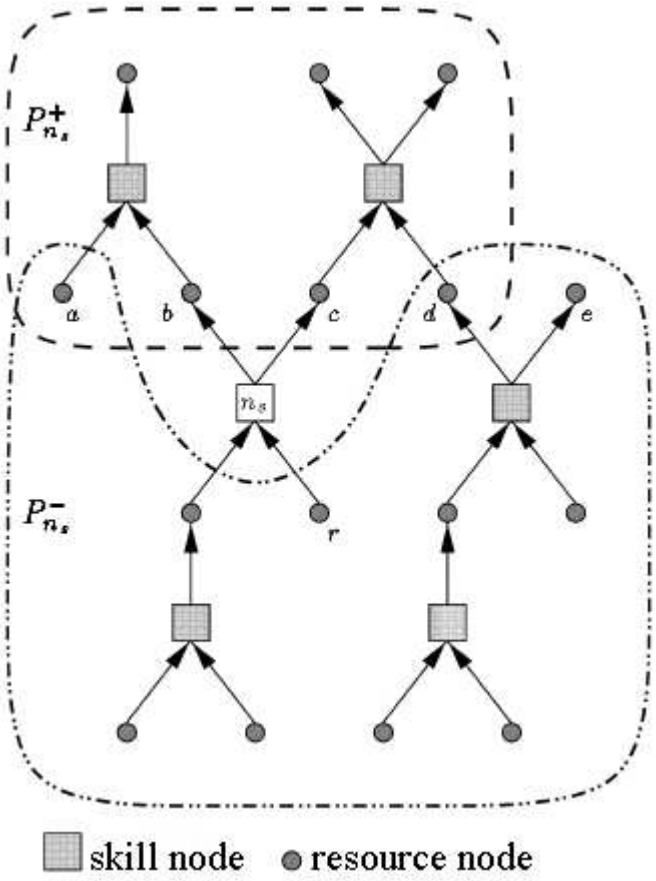

Figure 3.3: The subplans $P^+_{n_s}$ and $P^-_{n_s}$ of P w.r.t. n_s. [10].

the plan and an input resource consumed by some skill dependent to n_s. The node a is included in $P^-_{n_s}$ since it is an available resource regardless of execution of skill node n_s. The node a is also included in $P^+_{n_s}$ since a can not be consumed, unless n_s executes.

Also note that $In(P^-_{n_s}) = In(P)$, and that $Out(P^-_{n_s})$ is the set of resources that can be produced by the plan $P^-_{n_s}$, given the input resources $In(P)$, while $In(P^+_{n_s})$ is the set of resources needed to produce $Out(P^+_{n_s})$ [10].

The crucial point in this division is, given a skill node and division of the plan graph as formally defined above, $In(P^+_{n_s})$ is the set of resources necessary to execute the plan and reach the goals[6] Thus, if skill nodes in $In(P^+_{n_s})$ can be provided to the plan by other means, skill node n_s can safely be removed from the plan.

[6] We are assuming that plan does not contain any redundant actions. If it does, this division does not tell anything about goal realizability although the arguments are still valid for *all produced resources of the plan*, instead of goal resources.

3.2 Plan Revision and Coordination

Resource based plan graph, which is discussed in previous section is used as a tool for plan revision and coordination in [10]. The mechanism involves two agents, each having an initial plan for satisfying his goals. By using the tree division process defined in Section 3.1.4 agents determine the possible skills, that can be removed from their plan and compensate them with other resources they produce or request from other agents.

A skill node can be removed from the plan graph, if removing the node does not prevent production of goal resources. The agent may use the *graph division definition* given at Definition 2 to determine resource nodes resources to produce goal resources in the absence of the skill node. $In(P^+_{n_s})$ is simply what to examine for these necessary resources.

As mentioned earlier $In(P^+_{n_s})$ is the set of resources needed to produce $Out(P^+_{n_s})$. $Out(P^+_{n_s})$ is the set of goal resources that are produced by $P^+_{n_s}$, i.e., goal resources that could not be produced when n_s is removed from plan. If the agent may provide resources in $In(P^+_{n_s})$ by some means, then resources in $Out(P^+_{n_s})$ can safely be produced, and removal of the skill node n_s is safe.

Deciding on removing a skill node, agent tries to build a 1-1 mapping to resource nodes in $In(P^+_{n_s})$ from unused resource nodes in its plan and external resource nodes imported from plans of other agents. Note that since some resource nodes both exist in $P^+_{n_s}$ and $P^-_{n_s}$, some elements of the mapping are satisfied trivially.

In Figure 3.3 plan graph is divided into two subplans w.r.t. skill node n_s. Initial resources of the $P^+_{n_s}$ can be defined as $In(P^+_{n_s}) = \{a, b, c, d\}$, which means that the plan segment dependent on n_s requires these four resources as its initial resources. If the agent can find a 1-1 mapping from resource nodes available to these four nodes, then it is safe to remove skill node n_s.

Note that a and d are already in $P^-_{n_s}$. Thus agent need only construct the mapping to b and c.

Now assume that b and e are same type of resources, i.e., they are identical resources, but different instances[7]. Thus agent may map e to b. The semantic meaning of this mapping is, when n_s is removed, agent may use resource e instead of resource

[7] Type concept is discussed in details at [10].

b which would not be produced at that instance.

Finally agent should map a resource node to c. The agent first examines all its unconsumed (free) resources if they have the same type with c. If one found, mapping is constructed as it is done for b.

If no resource node in agent's plan may be mapped to c, then agent asks other agents for the resource c. Any other agent having the same type of resource with c in his plan graph hands it to the requesting agent. Proposer agent creates an external resource, mapping to c. Acceptor agent reserves the resource for external use, thus it is not used as input for another skill, or provided to another request[8].

Building a 1-1 mapping for $In(P^+_{n_s})$, agent may safely remove n_s from its plan.

However, agents may not always be willingly to provide other agents their resources. [10] defines two kinds of cooperation mechanisms for the framework.

- **Fusion** is a way of cooperation where total utility of agents increase or stay same as a result of cooperation.

- **Collaboration** is a way of cooperation where individual utility of agents increase or stay same as a result of cooperation.

Removing a skill node always increases utility of the agent, since all skill nodes would yield a cost in a typical environment. Achieving his goals with less skill instances is always preferable for an agent. However the the acceptor agent may provide the requested resource with a price. In such a situation, if cooperation schema is collaboration, the proposer agent should ensure its utility would not decrease after buying the resource from acceptor and removing the skill node.

Normally the acceptor agent would always be willingly to provide its free resources, since they are not required for plan execution.

The plan coordination framework of [10] assumes two agents having initial plans, which are represented in resource based plan graphs. Algorithms for fusion and collaboration are provided. Both algorithms depend on skill removal. In both algorithms agents attempt for skill removal in turn, thus play roles of acceptor and proposer in turn.

In fusion algorithm, agents attempt to remove skills in turn. There is no need to control an increase in cost, even if resources are provided with some cost, since this

[8] Acceptor and proposer are terminology used in [10].

cost is paid to another agent it does not cause a decrease in total utility. The algorithm terminates when both agents could not remove any skill in a single turn.

Collaboration algorithm is similar to fusion algorithm, but proposer agents control if cost of receiving an extra resource is greater than profit gained by removing the skill node related. If so, proposer agent would not be willing to provide the resource from the acceptor agent.

3.3 Discussion on Resource Based Planning Framework

Resource based planning framework discussed in this chapter is not far too superior to conventional logical framework. The most important aspect of the framework is straightforward cooperation mechanism based on exchanging nodes of plan graph.

Another important aspect is since all resource and skill nodes are ground terms, a unification mechanism is unnecessary for the plan coordination schema described above. But it must be mentioned that the planning framework would need such a mechanism if plans were to be constructed with this perspective. Since plan construction is considered out of scope in [10], grounded descriptions of resources and skills were sufficient.

Most important drawback of the domain is simplicity in terms of *interactions between agents* and the definite difference between planning, coordination and plan execution phases. These drawbacks are also noticed in conclusion by the authors.

This work uses resource based planning framework as a basis due to its effective features. The approach is flexible enough to propose different schemas for the interactions between agents and dynamic domains. Our work, using this representation, extends this framework to a more appropriate schema for dynamic interactions.

CHAPTER 4

PROBLEM DEFINITION

In this chapter the domain of this work, the dynamic postmen domain is explained in details. The domain is a classical domain that is studied with different perspectives, including multi-agent negotiation [13] and graph partitioning.

The choice of domain is affected by two factors. Firstly, domain is task oriented[1], thus it is appropriate for our resource based approach. Tasks and sub-tasks can easily be represented by resources in a task oriented approach. Secondly, the domain is modified to have dynamic characteristic, since extending the resource based perspective of [10] through dynamic environments is one of the motivations of this work.

4.1 Classical Postmen Domain

4.1.1 Domain Description

Postmen Domain is a classical domain, based on route finding on graph [14]. Two (or more) agents should deliver the letters they have got, to nodes of a graph, starting at a predefined node (post office) and returning that node when all letters are delivered. They are surely expected to take the minimum cost route, provided that all graph edges has some travel cost. The problem can be formulated formally as follows.

- *INSTANCE:* An undirected graph $G = (V, E)$, length function $l(e) \in \mathbb{N}$ for each $e \in E$, post office node $PO \in V$, delivery points $D \subset V$.

[1] Task-oriented domains are explained in Section 4.1.1 in detail.

- *SOLUTION:* A cycle in G (possibly containing repeated vertices) that starts and ends at PO and contains all elements of D

- *MEASURE:* Total length of the cycle.

Note that this formulation is a single-agent view of the problem. All agents have distinct letters to deliver and each agent tries to minimize its own cost. Thus we have more than one instance of the problem.

This domain is a simple *task-oriented domain*. Being task-oriented is a view from multi-agent perspective. A task-oriented domain is a domain which agents participating in the domain has non-conflicting goals. Each agent is assigned a list of tasks to accomplish. In postmen domain each letter assigned for delivery is a task.

Postmen Domain is interesting for negotiation perspectives. It is a simple setting for justifying a negotiation protocol. Agents may negotiate at post office for exchanging letters. They may have a bargain of exchanging subsets of their letters or form coalition for delivering letters in order to reduce their individual costs. Rosenschein and Zlotkin [14] discuss simple negotiation mechanisms on this domain.

4.1.2 Subadditive and Non-Subadditive Domains

As Rosenschein and Zlotkin [14] point out, classical postmen domain is *subadditive*. A task oriented domain is subadditive if for all finite sets of tasks, the cost of the union of tasks is less than or equal to the sum of the costs of the separate sets [14].

In subadditive domains, agents are always willingly to participate in coalition based interactions, since participating in a coalition never increases the cost of the union of tasks of coalition members. However this statement is valid only if total cost is the main concern[2].

Contrary to subadditive domains, in *non-subadditive domains* forming a coalition may increase the total cost of tasks of coalition participants. In such domains forming coalitions is not always preferable by agents.

Consider a variation of postmen domain, where agents need not return to post

[2] Participating in coalitions, the cost of tasks that an agent undertakes may increase. However in coalition based protocols generally individual cost is not the main concern. Since tasks are re-distributed among coalition participants randomly, cost is also re-distributed. Analysis can be carried on total cost or expected value of the cost. Details of coalition protocols can be found in [14],[13] and [15].

office after delivering all letters. This domain is non-subadditive [14]. Also each agent may be assigned a fixed final destination instead of returning to post office. This variation is also non-subadditive [13].

For coalition forming protocols, subadditive property of the domain is extremely important. For protocols based on task exchange, the structure of the cost function is important. In domains where individual cost is concerned, the subadditive property of domain is irrelevant, because total cost is not the concern. In domains where total cost is concerned, agents will always be willing to exchange tasks if environment is subadditive. And exchanging tasks would possibly increase the total cost in non-subadditive domains.

If we think in terms of *resource based plan coordination* of Section 3.2, in subadditive domains the agents will always participate in fusion type exchanges. However in collaboration type of exchanges the agents should evaluate the cost of their tasks with and without exchange, whether domain is subadditive or non-subadditive. If the domain is non-subadditive the agents should also evaluate the cost of their task with and without exchange.

4.2 Dynamic Non-Subadditive Postmen Domain

4.2.1 Main Features

In this work the domain concerned is a non-subadditive variation of postmen domain, where each agent has a predefined final destination to go after delivering all letter he is assigned [13]. Choosing a non-subadditive domain we prevent instances where agents would always cooperate regardless of the terms of cooperation.

Also two major modifications are made on this domain.

Firstly, in classical postmen domain and its non-subadditive variations, each letter is a delivery task. Initially the agent possesses all the letters he should deliver. He just decides on the route to deliver the letters and to return to his final destination, if there is any. In the modified version of the domain some letters must be collected from their departure points prior to delivery.

The motivation of this modification is increasing complexity of the domain and creating more interesting opportunities of cooperation between agents with the ap-

proach in this work applied. For example with letters having a departure and destination point, an agent may have another agent to collect a letter and pass it to him at a rendezvous point agreed upon. Moreover, this modified version of the problem is more realistic, since package delivery companies mostly work on the principal of collecting packages and delivering them.

Secondly, and as a more major modification, we introduce real-time dynamics to domain. In the classical postmen domain, each agent is assigned a set of tasks initially and is expected to accomplish them. Environment is static, thus agent need only construct a route for delivery (a plan from the planning perspective) and execute his plan without caring for any change in environment.

In the variation studied in this work, new letters could be assigned to agents at plan execution. For the sake of having a realistic setting, agent should collect the letter from its departure prior to delivery[3]. Thus while processing his initial task list, and agent would also be able to receive new tasks, modify his plan by adding collection and delivery of new letters and seek for possible task exchange with other agents regarding the new tasks received.

Moreover, in this work the main concern will be on *new tasks assigned to the agent*. We will not concentrate on initial route finding or initial task exchanging. The main concentration will be on processing tasks assigned to agents while executing the plan[4].

4.2.2 Details and Technical Issues

Using the notation used in Section 4.1.1, we can formalize out domain as follows:

- *INSTANCE:* An undirected graph $G = (V, E)$, length function $l(e) \in \mathbb{N}$ for each $e \in E$, post office node $PO \in V$, letter descriptions $L = \{(s, d) \mid s, d \in V\}$, final destination $FD \in V$.

- *SOLUTION:* A path in G (possibly containing repeated vertices and cycles) that starts at PO, ends at FD, contains all n where $(n, d) \in L$ or $(s, n) \in L$ and visits s prior to d for all $(s, d) \in L$.

[3] This fact is actually another reason for the first modification made. Assuming letters would mysteriously appear in postman's sack is not a wise setting.

[4] See Chapter 5 for details.

- *MEASURE:* Total length of the path.

This is only description of the initial state of the problem from a single-agent perspective. When executing the solution, new elements are added to L and the agent should adopt the solution for collecting and delivering these new letters.

The domain is designed to have a synchronized modification structure. There is a master clock that synchronizes the environment in intervals of *unit time*. At each synchronization the environment is modified regarding the actions executed by the agent since last synchronization and new environment state is transmitted to the agent[5]

Moreover the cost of travelling through graph nodes is related to time directly. A vertex, having a unit cost is travelled in unit time. For example, assume A and B are two nodes in the graph, $\lambda = edge(A, B)$ and $l(\lambda) = 3$. If an agent is at node A and executes action *move from A to B* as in Figure 4.1 , the action is executed in three units of time, starting at next synchronization. Any command, agent tries to execute in next two units of time is ignored, since the agent is in progress of executing an action. However, agent may initiate execution of an action at the third unit time interval, since an action initiated at that interval would be executed at the next synchronization.

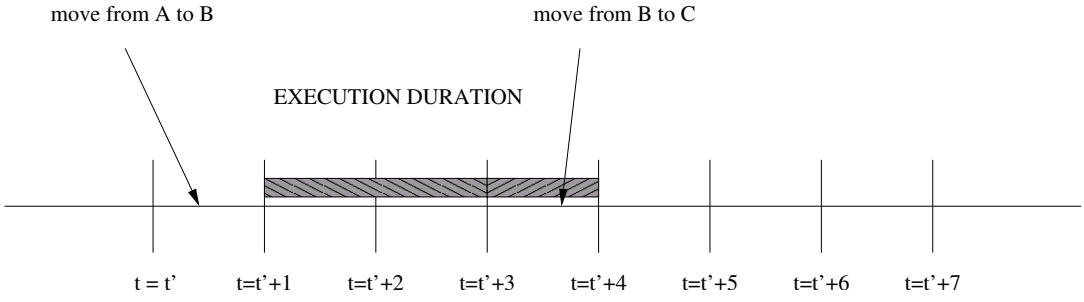

Figure 4.1: An example time line.

Note that in this schema, in order to start executing an action at $t = t' + n$, the agent must submit the action to environment at interval $(t' + n - 1, t' + n)$. Multiple submission in a unit time interval overwrite the previous ones, environment executes the last action submitted by agent. In the example given in Figure 4.1, if the agent

[5] In fact it is sufficient only to report errors in action execution (e.g. an invalid action), then agent may deduce its state regarding the action executed.

does not submit an action at interval $(t'+3, t'+4)$, then he stays still at node B at interval $(t'+4, t'+5)$ without executing any action. Agents must submit new actions before execution of previous actions ends, in order to continue executing their plans without a break. However ,initially agents are given some time to construct their initial plan.

Environment does not inform agents about states of other agents. Since in a task oriented environment, the tasks of agents are independent, they do not need to know about each other except during interaction.

Agents are assigned new letters at synchronization points, together with new environment state. Time synchronization flows regardless of the new letters. Agent should modify his plan considering new letters. In order to continue execution of the plan without a break, the agent should finish this modification until the synchronization which he should submit a new action to environment. If agent fails to finish the modification, he may submit an action without considering the modification, or he may prefer to submit no action and stay still. In this work we assume that the unit time intervals are sufficient for modifications agent make on his plan. However, if modification takes more time than an unit interval, agent is assumed to submit no action at that interval.

There are three actions in the environment.

- move ⟨node⟩ ⟨node⟩ for moving through connected nodes.
- collect ⟨letter⟩ for collecting a specified letter[6].
- deliver ⟨letter⟩ for delivering a specified letter.

Collect and deliver actions have unit cost, while cost of a move action is the weight of the edge between specified nodes.

Let us finish the discussion on domain constraints with presenting an example environment and example plan for the environment. Figure 4.2 is a simple graph. Assume that an agent is given a letter list

$$L = \{L_1 : (PO, A), L_2 : (PO, E), L_3 : (D, C)\}$$

[6] Each letter is identified with an integer

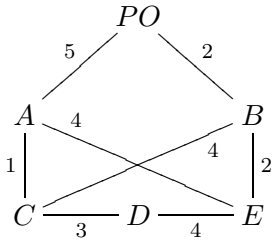

Figure 4.2: An Example Graph.

and final destination of agent is node A. The optimum plan for delivering given letters travels the graph with the route

$$R = PO, B, E, D, C, A$$

which has a cost of 16 units.

Assume that at node B, a new letter $L_4 : (D, A)$ is received by agent for delivery. Note that for this letter, agent need not change his route. Adding a collect and deliver action to plan will be sufficient. New plan would have a cost of 18.

Table 4.1: An Example Action Execution History.

Time	Agent	Environment	Time	Agent	Environment
I	move PO B		9	collect L_4	
0		Synchronization	10		Synchronization
0		Command OK	10		Command OK
1		Synchronization	10	move D C	
1	move B E		11		Synchronization
2		Synchronization	11		Command OK
2		Command OK	12		Synchronization
2		New Letter L_4	13		Synchronization
3		Synchronization	13	deliver L_3	
3	deliver L_2		14		Synchronization
4		Synchronization	14		Command OK
4		Command OK	14	move C A	
4	move E D		15		Synchronization
5		Synchronization	15		Command OK
5		Command OK	15	deliver L_1	
6		Synchronization	16		Synchronization
7		Synchronization	16		Command OK
8		Synchronization	16	deliver L_4	
8	collect L_3		17		Synchronization
9		Synchronization	17		Command OK
9		Command OK			

Table 4.1 shows the execution history of the constructed and later modified plan. Leftmost column indicate time flow, middle column indicate actions agent submitted, and rightmost column represent environment responses to agent.

Before the initial planning duration expires, agent submit his first action. Horizontal lines represent synchronization points. At each synchronization point environment reports agent if his previous action is executed successfully or not and transmits new letters assigned to agent, if any. Note that after environment assigns L_4 to agent, agent should refine his plan until submitting command *deliver* L_2.

Also note that actions *collect* L_4 and *deliver* L_4 are added to agent's plan after environment assigns letter L_4 to agent.

4.3 Complexity of The Problem

The problem presented is an optimization problem. Similar to the other optimization problems in graphs, the initial formulation of this problem, which is given at page 22, is NP-Complete. In this section we will attempt to prove that the problem formulated is NP-Complete.

Firstly, reword the problem definition for the sake of clarity.

Problem 1 *Minimum Non-Subadditive Postmen Problem:*

- INSTANCE: *An undirected graph* $G = (V, E)$*, length function* $l(e) \in \mathbb{N}$ *for each* $e \in E$*, post office node* $PO \in V$*, letter descriptions* $L = \{(s, d) \mid s, d \in V\}$*, final destination* $FD \in V$*.*

- SOLUTION: *A path in G (possibly containing repeated vertices and cycles) that starts at* PO*, ends at* FD*, contains all* n *where* $(n, d) \in L$ *or* $(s, n) \in L$ *and visits* s *prior to* d *for all* $(s, d) \in L$*.*

- MEASURE: *Total length of the path.*

Claim 1 *Problem 1 is NP-Complete.*

Proof Assume an instance of Problem 1 such that

Problem 2 *Modified Minimum Non-Subadditive Postmen Problem:*

- INSTANCE: *An undirected graph $G = (V, E)$, length function $l(e) \in \mathbb{N}$ for each $e \in E$, post office node $PO \in V$, letter descriptions $L = \{(PO, d) \mid d \in V\}$, final destination $FD \in V$.*
- SOLUTION: *A path in G (possibly containing repeated vertices and cycles) that starts at PO, ends at FD, contains all n where $(PO, n) \in L$.*
- MEASURE: *Total length of the path.*

If Problem 2 is NP-Complete, Problem 1 is trivially NP-Complete, since its complexity class is apparently higher than Problem 2.

Claim 2 *Problem 2 is NP-Complete.*

Proof In the proof we'll construct a reduction from *Minimum General Delivery Problem*.

Problem 3 *Minimum General Delivery Problem.*

- INSTANCE: *An undirected graph $G = (V, E)$, length function $l(e) \in \mathbb{N}$ for each $e \in E$, start node $s \in V$, final node $f \in V$.*
- SOLUTION: *A path in G (possibly containing repeated vertices and cycles) that starts at s, ends at f, contains all nodes in V.*
- MEASURE: *Total length of the path.*

Claim 3 *Problem 3 is NP-Complete.*

Proof is simple by a reduction from Minimum Travelling Salesperson.

Assume an instance of M-TSP. For all $(s, f) \in V \times V$, we can create an instance of Problem 3. Solving each Problem 3 instance, and calculating shortest path between each between f and s, we can solve M-TSP problem. The solution of Problem 3 instance for a given (s, f), concatenated with the solution of shortest path problem from f to s will produce a tour in G. Choosing (s, f) pair that produces minimum cost tour, we output a solution to M-TSP. It can be shown that there is no less costly solution than produced, however we will skip this demonstration.

Finding shortest paths can be executed in polynomial time [7] There are V^2 (s, f) pairs. Thus whole reduction is handled in polynomial time. We may conclude Problem 3 is NP-Complete.

For a Problem 3 instance, we can construct a Problem 2 instance. Note that the difference is Problem 3 visits all nodes in graph, while Problem 2 does not. We can divide each vertex into two and place a dummy node between new vertices as shown in Figure 4.3.

Figure 4.3: Vertices are divided into two with dummy nodes between.

Let G' be new graph constructed by dividing vertices of G. When dividing a vertex with weight l, we can distribute weight to two new vertices in any way we like. Define $L = (PO, n) \mid \forall n \in V$ where $G = (V, E)$. The construction of G' and L both have polynomial complexity. If this Problem 2 instance can be solved in polynomial time, Problem 3 can also be solved in polynomial time. We can conclude Problem 2 is NP-complete.

Since a specific instance of Problem 1 is NP-Complete, we can conclude that general formalization of Problem 1 is NP-Complete, too.

[7] All shortest paths can be computed in V^3 by Floyd's Algorithm [16]

CHAPTER 5

PLAN EXECUTION WITH REVISIONS

In Chapter 4 the dynamic modified postmen domain is discussed in detail. Here we describe how agents collect and deliver letters in the domain through planning and plan modification.

An agent constructs an initial plan when he is submitted the initial set of letters to be delivered. Proceeding to execute this plan, the agent need to modify his plan each time a new letter is assigned to him. This may happen in two different ways: (i) *re-planning process*, which inserts collection and/or delivery of a new letter to the plan, (ii) *interaction process*, where an agent asks other agents for delivery and /or collection of a letter.

The initial planning algorithm, re-planning algorithm and plan modification schema of the agent are discussed separately in detail in the following sections. The focus will be on agent and agent's reasoning mechanism, and architectural issues will be covered in Chapter 6.

5.1 Initial Planning Algorithm

In Section 4.2.2, we gave a formal description of the initial state of problem. When an agent is initially given a set of letters, he must generate a plan to collect letters and deliver them to their destinations. Moreover the cost[1] of the plan must be optimized.

As we have briefly shown in Section 4.3, this problem is NP-Complete. The in-

[1] The cost of the plan is measured in terms of path length and execution length, which are identical in our schema.

tractible algorithm for planning collection and delivery tasks of each agent initially is given in Algorithm 1.

The initial planner is only executed once in agent's lifetime as it is costly and cannot be afforded for re-planning due to any change in the environment.

Algorithm 1: Initial Planning Algorithm

input : Weighted, directed graph $G = (V, E)$, weight function w
Post Office Node $PO \in V$, Final Destination Node $FD \in V$,
Letter set $L = \{(s, d) \mid s \in V, d \in V, d \neq PO\}$

output : Path in G that collects and delivers all letters and have minimum cost

begin

1 Compute all shortest paths in G with Floyd's algorithm and store the result in matrix ASP;
2 Construct $G' = (V', E')$, with weight function w';
3 $V' = \{PO'\}$ where PO' is a new node;
4 $E' = \{FD'\}$ where FD' is a new node;
5 **for** *each* $(s, d) \in L$ **do**
6 Add new nodes s' and d' to V';
7 Add edges (PO', s') and (PO', d') to E';
8 $w'(PO', s') \leftarrow ASP[PO][s]$;
9 $w'(PO', d') \leftarrow ASP[PO][d]$;
10 Add edges $(s', n'), (n', s')$ for all $n' \in V', n' \neq FD'$;
11 Add edges $(d', n'), (n', d')$ for all $n' \in V', n' \neq FD'$;
12 $w'(s', n'), w'(n', s') \leftarrow ASP[s][n]$;
13 $w'(d', n'), w'(n', d') \leftarrow ASP[d][n]$;
14 Add edges (s', FD') and (d', FD');
15 $w'(s', FD') \leftarrow ASP[s][FD]$;
16 $w'(d', FD') \leftarrow ASP[d][FD]$
17 $PATH \leftarrow$ A minimum Hamiltonian path in G', for all $(s, d) \in L$ visiting corresponding s' prior to corresponding d';
18 **for** *each* $(n', m') \in PATH$ **do**
19 Replace it with shortest path from n to m;

end

The shortest path that collects and delivers all letters can be computed with an exhaustive search. However this approach calculates all paths in the graph exhaustively. What we really should compute is the order in which letters are collected and delivered, not the order in which nodes are visited. Thus instead of working on the problem graph, we construct a collection-delivery graph that represents only letters, their collection and delivery.

The nodes of this graph represent delivering or collecting a letter to a specific node of original graph. For example consider a letter set $L = \{L_1 : (PO, A), L_2 : (PO, E), L_3 : (D, C)\}$ from the example given in Section 4.2.2. The delivery-collection graph constructed from this letter set is shown in Figure 5.1.

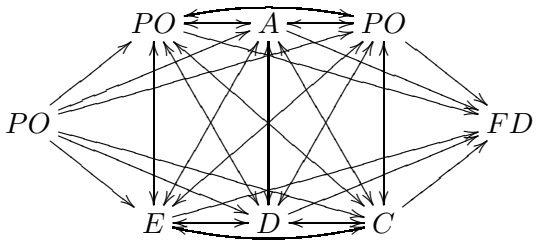

Figure 5.1: An example delivery-collection graph produced by the letter set given as an example in Section 4.2.2

The vertices of the graph represent the order in which delivery or collection takes place. The weight of the vertices represent the shortest path between two connected nodes, w.r.t. the original graph. In the solution Hamiltonian path, selected vertices impose an ordering of deliveries and collections.

Note that if there is more than one delivery or collection at a node, a unique node is inserted for each. In Figure 5.1, there is a node for each collection from PO. However weight of vertices between these nodes are all 0. We can generalize this as

$$\forall x \in V;\ n, m \in \alpha, where\ \alpha = \{x' \mid x' \in V', x' \text{is inserted to } V' \text{for } x\} \Rightarrow w'(n, m) = 0$$

It is clear that $PATH$ in Algorithm 1 would visit all elements of α once one of them is visited. In fact our Hamiltonian path search algorithm uses this fact as a heuristic. Once a node is visited, all nodes with the same source node are visited before expanding next visited node. This heuristic is equivalent to collecting and

delivering all letters addressed to and from a node once the node is visited [2].

PO has also another significance. Every agent is initially at PO. All letters addressed from PO are initially available to agents, and agents need not execute any action to collect them. Thus initially an agent is assumed to have collected all letters from PO.

Similarly if agent is assigned to deliver letters to his final destination node, he would prefer to make this delivery as last action in plan. However dependency issues may surely force agent to visit FD node more than once.

Also note that, the fact that "a letter should be collected before delivery" is not represented in the structure of the produced graph. Thus the algorithm explicitly enforces this order.

This order is also used for decreasing the fan-out of the search tree, while calculating Hamiltonian path. A *possible next node list* is maintained, apart from the graph. This list represents the deliveries or collections that can be handled for the time being. Collections can be handled anytime in the path, while each delivery must antecede the collection of the related letter.

The possible next node list initially contains delivery nodes of letters that are departed from PO and collection nodes of other letters. When searching for Hamiltonian path, only nodes that are in possible next node list are visited. Each time collection node of a letter is visited, delivery node of the letter is replaced with the collection node. Each time a delivery node is visited, it is removed from the list. This list clearly decreases the fan-out of the search together with the above heuristic.

One last detail about this algorithm is that, it is successful in optimizing delivery and collections that are dependent to each other, even in circular way. Assume two letters $L_1 = (A, B)$ and $L_2 = (B, G)$ in the letter set. The heuristic discussed above leads the plan to deliver L_1 as collecting L_2, or vice versa. It is up to implementation which alternative is selected, but they have the same cost.

Assume a third letter in set $L_3 = (G, A)$. This letter introduces a circular dependency in the sense of deliveries and collections. If delivery of L_1 is handled right

[2] All letters do not include ones that are addressed to the visited node but not yet collected. The plan should prefer to visit the departure node first in order to generate a less-costly plan, but there might be other dependencies, or circular dependencies that enforce a node visited twice. In such occurrences, all letters would not be collected or delivered at first visit. Note that this does not contradict with characteristic properties of solution of Hamiltonian path problem, since there are more than one node representing the deliveries and collections.

after (or before) collection of L_2, then it must precede delivery of L_2. Similarly to optimize deliveries and collections, delivery of L_2 should precede delivery of L_3 and delivery of L_3 should precede delivery of L_1 which is a contradiction. Thus one of the nodes A, B and G should be visited twice. The exhaustive search algorithm expands all possible paths of these three. The possible next node list guarantees that one of the deliveries is not handled while collection from the same node is handled. Thus algorithm returns one of the nodes once more for making the delivery.

The output of the algorithm is a node string $p \in V^*$. This path is the shortest path that allows the agent to collect and deliver all letters and end up in FD. For example consider the example in Section 4.2.2, where $L = \{L_1 : (PO, A), L_2 : (PO, E), L_3 : (D, C)\}$,$FD = A$ and graph is given in Figure 4.2. The path generated by Algorithm 1 is $PO \cdot B \cdot E \cdot D \cdot C \cdot A\cdot$.

5.2 Resource Based Plan Graph Construction

As we discussed in Chapter 3, resource based perspective is the focus of this work. Thus after generating the initial plan, a resource based plan graph is constructed. Plan execution and plan revision issues will all be handled using this resource based graph.

The resources used to represent the domain are

Connected Resource ($CO^{a,b}$) represents the connectivity of two nodes of graph

Delivered Resource (DE^l) represents delivering a letter

Have Letter Resource (HL^l)represents holding a letter

Location Resource (LO^a) represents being at a node of the graph

Regarding the schema of [10] we did not use patterns or incomplete resources. All resources used in initial resource based graph together with others to be appear later are completely defined, featuring all necessary information.

The basic skills used in domain are

Collect Skill (C^l) represents collecting a letter

Deliver Skill (D^l) represents delivering a letter

Go Skill ($G^{a,b}$)represent moving from one node to another

As mentioned before in Section 4.2.2, there are three possible actions for agents corresponding to these skills[3].

The abstract type definition of skills as follows

$$\mathbf{S} = \left\{ \begin{array}{lllll} C^l: & \{HL^l, LO_2^s\} & \leftarrow & \{LO_1^s\} \\ D^l: & \{LO_2^d, DE^l\} & \leftarrow & \{HL^l, LO_1^d\} \\ G^{a,b}: & \{LO^b, CO^{a,b}\} & \leftarrow & \{LO^a, CO^{a,b}\} \end{array} \right\} (s,d) \in L$$

Right sides represent resources consumed by the skill, left sides represents resources produced by the skill. As proposed in [10], we used unique instances of a resources each time it is produced or consumed in plan. We use subscripts to distinguish between different instances. LO_1^d and LO_2^d represent different instances of a resource of the same type. Collecting a letter does not affect the location of agent but the consumed and produced resources are different resources, even if they have same type. The superscript d designates being in location d.

However $CO^{a,b}$ type resources are treated differently. They are labelled as *domain resources*. They are characteristic property of the domain and would not cease unless graph structure is altered. It is desirable to maintain only a single instance of them in plan, since they are not consumed or produced.

All possible skills are generated initially and made available for the environment. To be efficient, we would not attempt to generate all skills, but only necessary ones. However this would not be confused with *using abstract type definition of skills*. The abstract definitions given above are totally useless in resource based framework, since they do not specify all parameters of skills. A usable skill has the form

$$C^{l_i}: \{HL^{l_i}, LO_2^{g_i}\} \leftarrow \{LO_1^{g_i}\}$$

where l_i is a valid letter identifier and g_i is the collection node of letter.

Initially every agent has all the connected resources of the graph, an instance of LO^{PO} type resource and "have letter resources" for each letter he has to collect from PO. Goal state of agent contains delivered resources for all letters of his initial letter set and an instance of LO^{FD}.

[3] This skill set will be expanded in Section 5.4.2 with skills for exchanging letters.

Resource based plan graph is constructed as explained in Section 4.2.2. Figure 5.2 depicts the resource based plan graph of the ongoing example.

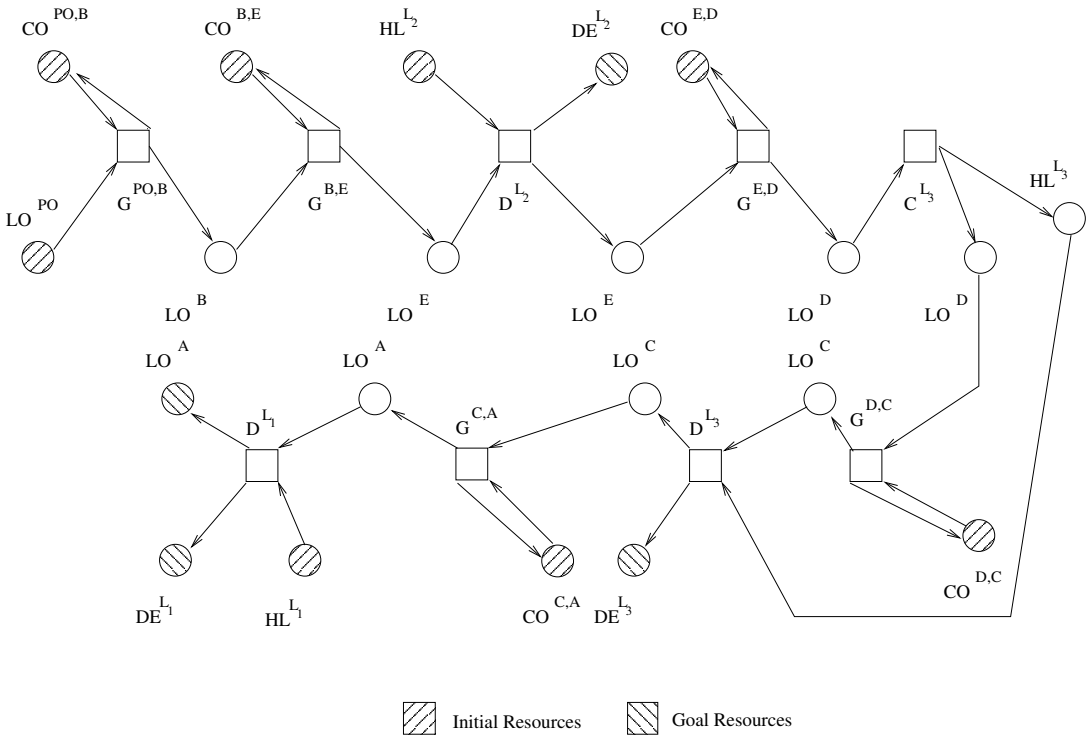

Figure 5.2: Resource Based Plan Graph of Ongoing Example.

Note that in our domain the most essential resource is location resource type. An agent possesses one and only one location resource for an instance and each skill consumes some location resources. Thus the structure of our plan graph is *linear*. The location resource of the agent is consumed and re-produced each time a skill is applied. Other resources are consumed and produced in this chain.

This linear structure of plan graph has some practical advantages. If we represent the parent-to-child links in two direction, the graph would be represented as a *doubly linked list*. Thus we can apply simpler forms of basic operations to graph. For example in a generalized resource based graph, finding all delivered skills dependent on a given resource is a tree traversal problem. In our domain, this operation can be implemented by simply reading all the linked list.

In the following sections we will use algorithms and schemas for modifications and operations on plan graph. Implementation of these algorithms and schemas will rely on the linear structure mentioned. Thus some of the implementaions would be

useless for a generalized non-linear plan graph. However all of the schemas and algorithms may be adopted for generalized non-linear graphs with different implementations.

5.3 Plan Execution

Generating an initial plan and constructing the appropriate resource based plan graph, every agent begins to execute his plan. The synchronization structure of the domain was discussed in Section 4.2.2. Algoritm 2 summarizes the life cycle of an agent.

Algorithm 2: Life Cycle of an Agent

```
cost ← 0;
repeat
    if cost ≤ 1 then
        s ← possible skill for execution;
        cost ← cost of s;
        submit the execution request for s to the environment;
        wait next synchronization;
        if s is reported to execute successfully then
            modify plan regarding s;
        else
            cost ← 1;
    cost ← cost − 1;
    receive new letters;
    if cost ≠ 0 then
        wait next synchronization;
until goal achieved ;
```

The costs of actions in the environment were discussed in Section 4.2.2. Cost of skills are identitical to costs of actions. Moreover costs of skills are also directly related to time. Each agent becomes idle for a period when executing a skill and this period is proportional to cost of skill. If an agent fails to submit an action or if the action agent submits can not be executed, agent remains idle for a single unit time.

Unit time and synchronization concepts were also discussed in Section 4.2.2. En-

vironment sends *synchronization signal* to agent in unit time intervals. Durations of actions are determined using these synchronization signals. Every agent stays idle for a unit time when waiting for the next synchronization.

Algorithm 2 demonstrates execution of a skill by an agent. The agent first chooses the skill he would execute. Since all location resource instances in the plan are unique, at an instance the agent has only one possible skill to execute. Agent just searches the skill set which consumes the current location resource agent possesses, and executes that skill.

In order to execute that skill agent submits the action related to skill to the environment. Then the agent waits for the next syncronization, where the environment would confirm the action if it is appropriate and reject it if producing resources are not existent.

If environment rejects the action agent submits a new action. If the environment confirms the action, the agent waits execution of action for $cost - 1$ more synchronization, where $cost$ is the cost of action executed. However the agent would not be totally idle for this $cost - 1$ unit time. For each *synchronization signal* received, agent checks if there are new letters to be delivered. If there are any, he modifies his plan to handle them.

Note that if the action has a single unit cost, the agent would not wait for its execution. He would wait for a single unit time, for receiving the confirmation from environment. Then he would submit the new action, without any further delay.

After each action agent checks whether goal state is achieved i.e., $G \subseteq C$, where G is the set of goal resources and C is the set of current resources agent possesses. If goal state is achieved a specific halt command is sent to environment and execution of agent terminates.

So far we discussed how an agent generates initial plan and executes it. In next section we discuss the missing part in this picture; how a plan is revised when a new letter is assigned to the agent. This part is the main focus of the schema, since our main motivation is to build an agent that can modify his plan with great flexibility in real-time in case new goals are introduced.

5.4 Plan Revision to Deliver New Letters

When an agent is submitted a new letter, a delivered resource related to that letter is added to goals of the agent. In order to deliver this letter, the agent should visit the destination node of the letter. Also the agent should visit the departure node of the letter, in order to collect it prior to delivery.

In fact the agent first generates the modified version of his plan that would handle new goals and calculates the cost of this plan, without directly applying it. Then the agent explores the cooperation alternatives and if beneficial tries to find a cooperation mechanism better than delivering the letter himself. If he fails to find such a mechanism, then the modified plan is put in execution.

We will first demonstrate how an agent modifies his own plan in order to deliver new letters, then we will proceed to how the agent interact with other agents in order to find a cooperation mechanism.

In case more than one letter assigned, an agent processes them one by one. Different orderings in which the new letters are processed might change the final plan that delivers all letters. In the agent design we did not attempt to process the letters in parallel, or group them. Instead a simple heuristic is applied.

The environment submits the letters in an arbitrary order. Instead of processing them in this order, the agent generates an ordering regarding a simple heuristic. The heuristic is simply based on the minimum distance of nodes of letters to the plan. By minimum distance of a node to plan, we mean the minimum of the shortest path values from collection and delivery nodes of new letter to plan nodes.

This distance is just an approximation of the additional cost that agent pays to collect and deliver the letter. It is not the minimum cost, or a fraction of the minimum cost. It can be shown that changing the initial plan path may result in less costly plans. However this heuristic is effective in ordering letters for processing.

The agent first calculates this heuristic value for all letters, and sorts the letters regarding to this heuristic value in increasing order. This schema first processes letters that are already on plan path, or can be delivered with short diversions from the plan path. More complex letters are processed later.

5.4.1 Modifying Plan for New Letters

When an agent receives a new letter, there may not exist any beneficial cooperation schema. This means that the agent should carry out the delivery on his own, by revising his plan accordingly.

The initial plan is constructed to deliver all letters in minimal cost. However it is not trivial to incrementally modify this plan to handle new letters. In fact it would be trivial, if the agent should only deliver that letter. However, the agent should collect the letter prior to delivery and the cost we try to minimize is total additional costs of delivery and collection. Moreover, a new letter might completely change the least costly possible plan.

In fact, given a plan and a new goal, constructing a new plan including new goal and ensuring the new plan has least possible cost is not possible incrementally in our domain. One could still run the initial planning algorithm with new letter set to generate a new optimal plan. But as we discussed in Section 5.1, this is not feasible if the plan and graph is large and reaction time to new letters is critical. Thus we would not attempt to generate an optimal plan from scratch, instead a sub-optimal one.

Consider the plan graph in Figure 5.2. We might benefit from the linearity of the graph. The graph starts with go skills, followed by delivery skill. In fact in most general terms the graph starts with go skills, then follows delivery or collection skills, then again comes go skills. These blocks repeat until the end of the graph.

This fact is obvious if we think in terms of agent's plan. The agent would leave post office to a node where he would collect/deliver some letters. After finishing deliveries in the node, he leaves for another node and makes collections/deliveries there. Thus we may view agent's plan as delivery/collection skills separated by go skills.

The collection and delivery of a new letter might be inserted between two delivery/collection blocks, without changing the structure of the initial plan. To formally present this method we present formal definition of a plan from [10].

A plan can be formally expressed as

$$I = R_0 \vdash_{s_1} R_1 \vdash_{s_2} R_2 \ldots \vdash_{s_n} R_n = G$$

where R_i is a resource set, I is set of initial resources, G is set of goal resources and

relation $\vdash_{s_i}$ is application of skill s_i to a resource set, producing another resource set. Note that a skill consumes some resources and produce others, and relation $\vdash_{s_i}$ represents this issue.

We might define two new relations for our domain, namely $\vdash_\alpha = \vdash_{s_i}$ for some delivery or collection skill s_i and $\vdash_\beta = \vdash_{s_i}$ for some go skill s_i. The plans in our domain can be expressed as

$$I = R_0 \vdash_\beta^+ R_1 \vdash_\alpha^* R_2 \vdash_\beta^+ R_3 \vdash_\alpha^* R_4 \ldots \vdash_\beta^+ R_{n-1} \vdash_\alpha^* R_n = G$$

Defining $R_i \vdash_\gamma R_j = R_i \vdash_\beta^+ R_k \vdash_\alpha^* R_j$

$$I = R_0 \vdash_\gamma^+ R_n = G$$

This last expression precisely represents the blocks of skills we discussed above.

Collection and delivery of a new letter can be inserted into plan, between application of two $\vdash_\gamma$. The agent, finishing collections and deliveries in a node, proceeds to collection/delivery node of new letter. Collecting/Delivering the letter, agent proceeds to node which he would have proceeded if the new delivery/collection had not been inserted into plan.

Still it is not trivial to insert delivery and collections. Agent should decide, after which delivery/collection he must proceed to delivering/collecting the new letter. Moreover the delivery of the letter should precede the collection of it. Also the agent should examine if destination or departure node of letter is already visited by the plan.

Agents use two algorithms for inserting a given letter into his plan[4]. The first algorithm is applied when an agent is already visiting both collection and delivery node of the letter. Second algorithm is applied if the agent should change its route i.e., is not visiting at least one of the collection and delivery nodes. First algorithm is presented directly.

[4] Two algorithms are used instead of a single combined one. This is done intentionally. The reason will be apparent in Section 5.4.2.

Algorithm 3: Modifying plan with minor changes for a new letter.

input : Plan P, letter l
output : *true* if letter is inserted, *false* elsewhere

```
1 if P does not visit source(l) then
2     return false
  else
3     s ← first location resource of node source(l) that is produced by a
      collection/delivery skill and consumed by a go skill;
4     if P does not visit destination(l) after s then
5         return false
      else
6         d ← first location resource of node destination(l) that is produced by
          a collection/delivery skill and consumed by a go skill and dependent
          to s;
7         insert a collection skill for l, consuming s;
8         insert a delivery skill for l, consuming d;
9         return true
```

The algorithm is very simple. It just checks if the plan visits departure node and then destination node of letter. If both nodes are visited and ordering is fine, collect and deliver skills are inserted into the plan right after delivering all other letters to nodes.

Second algorithm searches the plan graph on location resource nodes that are produced by delivery or collection skills and consumed by go skills. The producing skill of these nodes are last delivery of collection skill executed at corresponding node. The algorithm tries to insert the delivery or collection sub-plan of letter after each such node.

Firstly algorithm decides if departure or destination node of the letter is already visited by plan[5]. Thus algorithm decides if the search would consist of collection, delivery or both.

Then algorithm tries to glue the search objective into all possible places in the

[5] It is guaranteed that not both of them are visited. If they were, Algorithm 3 would handle the letter.

plan. Note that only nodes specified above are considered in search.

Gluing a collection or delivery after a location resource consists of (i) removing all go skills after the given location resource i.e., removing path from the corresponding node to the node at which next delivery or collection takes place, (ii) inserting a path to the collection or delivery node of the letter from corresponding node (iii) inserting a collect or deliver skill (iv) inserting a path from the collection or delivery node of letter to location resource of which producing go skill is removed at step (i).

Note that the collection and delivery might be glued to different parts of the plan, or they might be glued to the same part of the plan. In this case two steps are added to gluing collection after a location: (iii-b) inserting a path from collection node of the letter to delivery node of the letter (iii-c) inserting a delivery skill.

Algorithm 4 is summarized explanation of this second algorithm. Examples for execution of both algorithms are presented in Chapter 6.

5.4.2 Cooperation Schema

At previous subsection we discussed how agents might modify their plans for new letters. However modifying plans mostly increases cost of the plan, because purpose of plan modifications are new goals. This subsection discusses how agents might form cooperation mechanisms for delivery or collection of new letters.

Receiving a new letter agent initially examines if the letter could be inserted into plan using Algorithm 3. Algorithm 3 is the least costly way to insert the letters into plan as it only inserts a collection and a delivery skill to plan.

If Algorithm 3 fails to insert the letter, the agent should examine the means of cooperation with other agents, before trying to handle the letter himself. A simple resource based cooperation mechanism is used for this purpose.

Note that an agent might have different kinds of utility and cooperation conceptions. The collaboration based cooperation mechanism we considered is outlined bellow and will be examplified in 6.

The mechanism is based on requesting resources. This kind of approach is used in [10] for initial plan revision stage. Here, we considered a modified version of this approach to handle changes in the environment.

Receiving a letter an agent adds to his goal resources, a delivered resource related

Algorithm 4: Modifying plan for a new letter.

input : Plan P, letter l

$S \leftarrow$ Set of all Location Resources produced by a Delivered/Collect Skill and consumed by a Go Skill;

for *each* s *in* S **do**

    $next(s) \leftarrow$ First Location Resource following s, produced by a Go Skill and consumed by a Delivered/Collect Skill.

$c \leftarrow$ collect node of l;

$d \leftarrow$ delivery node of l;

$sp \leftarrow$ shortest path function on graph of P;

if *Collect node of l is in plan path* **then**

    insert a Collect Skill after $m \in S$, $location(m)$ is c;

    choose $n \in S$ that minimizes $sp(n, d) + sp(d, next(n)) - sp(n, next(n))$;

if *Delivery node of l is in the plan path* **then**

    insert a Delivery Skill after $n \in S$, $location(n)$ is d;

    choose $m \in S$ that minimizes $sp(m, c) + sp(c, next(m)) - sp(m, next(m))$;

else

    choose $(n, m) \in S$ that minimizes
    $sp(m, c) + sp(c, next(m)) - sp(m, next(m)) + sp(n, d) + sp(d, next(n)) - sp(n, next(n))$ if $n \neq m$
    $sp(m, c) + sp(c, d) + sp(d, next(m)) - sp(m, next(m))$ if $n = m$;

insert sub-plan to visit c after m, d after n where necessary;

to this letter. The agent asks all other agents of this resource or have letter resource related to the letter. Asking delivered resource simply means asking for collecting and delivering the letter. Asking for have letter resource means asking for collecting the letter and handing it to him at some rendezvous point.

The agent might have implicit preferences of asking one kind of resource. Here we assume that agent firstly asks for delivered resource, then have letter resource if first request is not satisfied.

It is quite likely that most of the time other agents would not have the requested resource of a resource of the same type free in their plans. However they should look for producing the resource themselves, when resource is requested, if they are

willing to cooperate.

An agent receiving a request for a resource first checks his plan if it contains the requested resource or any resource of the same type. If the requested resource or one of the same type exists and is free, he fulfills the request.

If the resource does not exist, he should calculate the cost of producing that resource. If requested resource is a *delivered resource*, this cost is the cost of collecting and delivering letter as if it was assigned by environment. If the requested resource is a *have letter resource*, the cost consists of collecting the letter (possibly travelling to collection node of the letter) and cost of rendezvous with other agent.

Note that producing a requested resource is itself a re-planning issue similar to the one in Section 5.4.1. After calculating the cost of a resource the agent should decide

- If he should accept providing this requested resource,
- If he should decline providing it,
- If he should accept providing it in exchange with a resource other agent provide to him.

There are two resources that agent might ask for exchange with requested resource. He might ask for the *delivered resource* of another letter, or if a *delivered resource* is requested, he might ask for *have letter resource* of the related letter. First counter-request has semantic meaning of exchanging letters. Second counter-request is simply accepting to deliver the letter, if requesting agent could handle the collection himself and hands the letter to agent at a rendezvous point.

We assumed that agents could only request single resource for exchange. This assumption is necessary for reducing complexity of negotiation.

The agent who makes the initial request, examines the situation. He calculates the cost of the skill that his party requests as his party examines his request. If he is satisfied with the exchange, both parties modify their plans regarding the exchange.

The basic agent model discussed so far can be summarized in following algorithm.

Algorithm 5: Processing New Letters.

```
input : Plan P, letter l
if P visits collection and delivery nodes of l then
    add collection and delivery skills to P;
else
    r ← delivered resource of l ;
    request r from other agents;
    if response is OK then remove r from P;
    else if response is a request for have letter resource of l then
        remove r from P;
        insert collection of l to P using re-planning algorithm;
        insert rendezvous to P;
    else if response is a request for delivered resource of another letter then
        assess the request;
        if request is beneficial then
            remove r from P;
            insert collection and delivery of requested letter using re-planning
            algorithm.;
    else
        r ← have letter resource of l ;
        request r from other agents;
        if response is OK then
            insert delivery of l to P using re-planning algorithm;
            insert rendezvous to P;
        else
            insert collection and delivery of l to P using re-planning
            algorihtm;
```

Now we attempt to analyze possible outcomes of this exchange schema. Name the agent that makes the first request as proposer, other party as acceptor.

1. Parties may exchange delivered resources. Acceptor collects and delivers the letter of proposer. If acceptor had not collected the letter he asked for, pro-

poser collects and delivers it. If acceptor had collected the letter, parties meet to exchange the letter.

2. One party might request a delivered resource. Other party might request related have letter resource. Party that requests delivered resource collects the letter. Parties meet and exchange the letter. Other party delivers it.

3. Proposer might request a have letter resource. Acceptor might request delivered resource of another letter in exchange. Proposer delivers the letter acceptor asked for. Parties meet to exchange the letter Proposer asks. If acceptor had not collected the letter he requested, proposer also collects it. If acceptor had collected it, they exchange that letter too at the rendezvous.

One important issue is, how acceptor chooses a delivered resource to request in exchange. To store or calculate an approximation of the delivery cost of each letter would be sufficient for this decision. This approximation is simply how much the plan cost would decrease if a letter did not exist, and travels to collection and delivery nodes of letter are removed from plan. These costs are calculated at initial planning stage. They might be stored and modified when necessary.

Another issue is how to handle the rendezvous. In order to accomplish a rendezvous, both parties should agree on a node and time. In negotiation, when exchange schema is guaranteed to contain a rendezvous, both parties declare their constraints for rendezvous time. Constraints might prevent other party from participating into exchange. If both parties stay willing to cooperate, acceptor determines candidate nodes for rendezvous and determines when he would be on these nodes. The proposer finalizes the rendezvous node and time by using choosing the least costly node-time pair.

The candidate nodes selected by acceptor are simply uniformly selected nodes from his plan path. This schema is used to not to reveal all plan, but reveal enough information for arranging rendezvous. 20% of the nodes are selected, which means for each five node on plan path, one node is selected as a candidate. 20% is an emprical ratio. Selecting more node increases the chance of arranging an optimal rendezvous. However we do not want much of the plan to be revealed. Once the proposer chooses his candidate nodes it is easy to determine when he would be on these nodes, simply

traversing plan graph.

The proposer might wait for the acceptor, if he reaches the designated rendezvous point before the acceptor. It is proposer who waits, since he is mainly the one that benefits from the cooperation. The acceptor is also possibly reducing his plan cost. But the cost of proposer's plan would definitely be increasing if cooperation fails.

When both parties are at rendezvous node they simultaneously execute exchange skills. There are two exchange skills defined.

Exchange From Skill (E^l) represents receiving a letter from another agent

Exchange To Skill (T^l) represents handing a letter to another agent

Other details of cooperation schemas will be presented with examples in Chapter 6.

CHAPTER 6

IMPLEMENTATION OF SYSTEM, RESULTS AND DISCUSSION

In this chapter we would present architectural and conceptual details of the system. We will not go into much detail in these issues but most of the architectural detail skipped in Chapter 5 will be considered here.

Also we would demonstrate results of the implementation by presenting examples of different kinds of cooperation and re-planning instances.

6.1 System Design

6.1.1 Architectural Issues

In order to test and observe the concepts presented in Chapter 5 a testbed is implemented. Instead of using a symbolic implementation environment such as LISP, we preferred using Java classes to represent agents, skills, resources, plans and other entities in the system.

Each agent and environment are designed as separate threads in the same runtime environment. They do not have access right to data of others. Agents communicate with environment through network sockets. This provides ability to communicate with remote environments.

Agents communicate with each other through method calls of a communication layer similar to a blackboard architecture. An agent could post a request to communi-

cation layer, and this request is distributed to all agents. The first positive answer to the request is conveyed to requesting agent. This mechanism might also be extended to a more distributed nature by using RMI [1] technology but we did not attempted it.

Figure 6.1 is a sample screen shot from implementation.

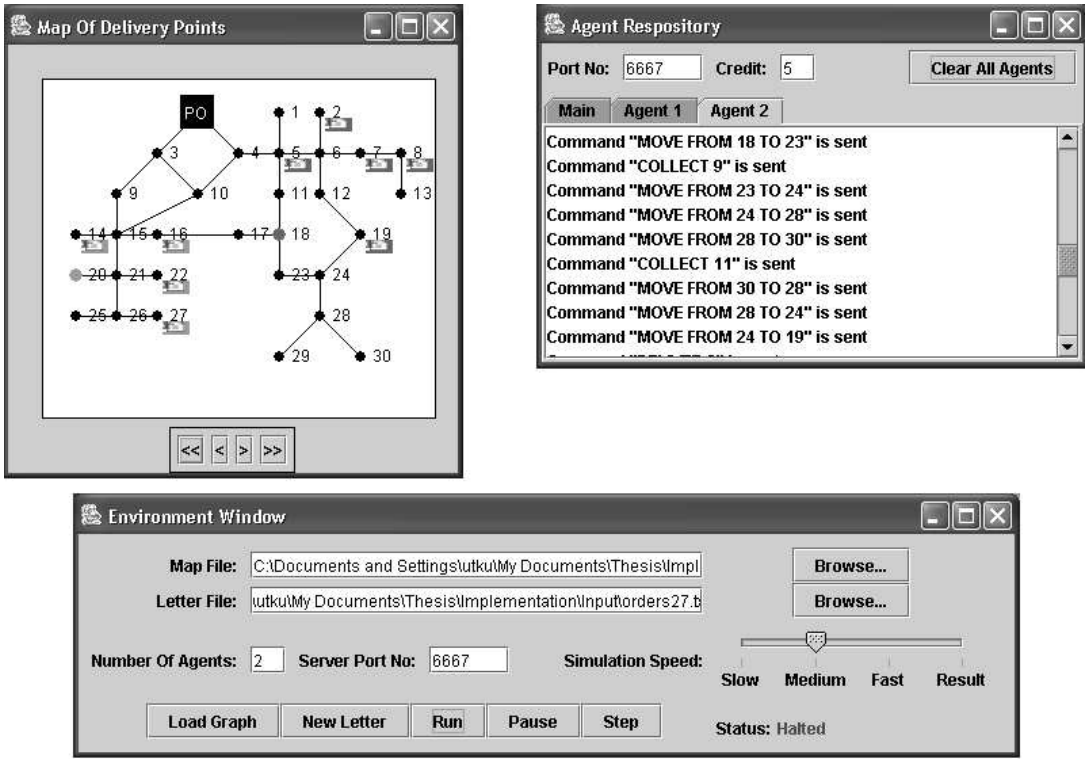

Figure 6.1: A Screenshot From Implementation.

6.1.2 Conceptual Issues

The most important issue in implementing the system was choosing a cooperation strategy for agents. We adopted a collaboration based strategy, for providing agents with a more autonomous behavior.

In the system implemented agents have a utility function based on the cost of its own plan. The total costs of the multi-agent plan or costs of the other agents is not the concern of agent. The agent only tries to minimize cost of his plan as much as possible.

[1] RMI is abbrevation for Remote Method Invocation, a technology used with Java for distributed computing.

However it is clear that such a pure collaboration limits cooperation opportunities. As we mentioned in Section 5.4.2 in our domain, requested resources generally do not exist free in other agent's plans. Acceptor agent should produce them, and no matter how trivially they are produced, agent should at least execute a new collect or deliver skill. Since even collecting a letter from a node in his path and delivering it to another node in his path, agents would not be willing to cooperate when pure collaborative schemas are applied to our domain.

Thus we inserted a concept of *credit*. Each agent has a measure of "credit of other agents for him". This value determines the maximum increase in his cost agent could stand in a cooperation without taking anything in exchange. For example if the credit value of agent is greater than or equal to two, agent will be willing to collect and deliver a letter whose collection and delivery nodes are on agent's path. Note that both collection and delivery of a letter have unit costs.

Moreover after each cooperation this credit value is updated. If agent is proposer, he must have gained some profit by cooperating and not delivering the assigned letter himself[2]. Agent adds this profit to credit value. If the agent is acceptor, he might have gained some profit by exchanging one of his deliveries with the requested resource. Or the agent might have fulfilled the request due to its credit value. If cost of the agent's plan increases, agent decreases credit value by same amount; if cost of his plan decreases, increases the credit value.

In previous chapters we mentioned that our plan revision schema is applied only when a new letter is assigned to agent. We do not apply any plan revision algorithm when constructing initial plans. We did not attempt to apply an initial plan revision since our focus is on dynamic plan revision. However mostly in task oriented domains tasks are almost never divided between agents randomly. An initial plan revision is mostly applied by agents or the environment assigns the tasks to agents in an mutually exclusive fashion. We also applied second schema for the system implemented. Initial letters are assumed to be divided between agents nearly mutually exclusive.

Since we provided the cooperation strategy for agents, we now present replanning, cooperation and negotiation examples for the domain.

[2] If he did not profit from exchange, he would deliver the letter himself.

6.2 Examples on Execution of Resource Based Plan Revision System

In this section we present some examples from execution of resource based plan revision system implemented. These examples provide the details of plan revision algorithm, and how agents form cooperations applying the negotiation strategy used.

First we present a graph that is used all through the examples.

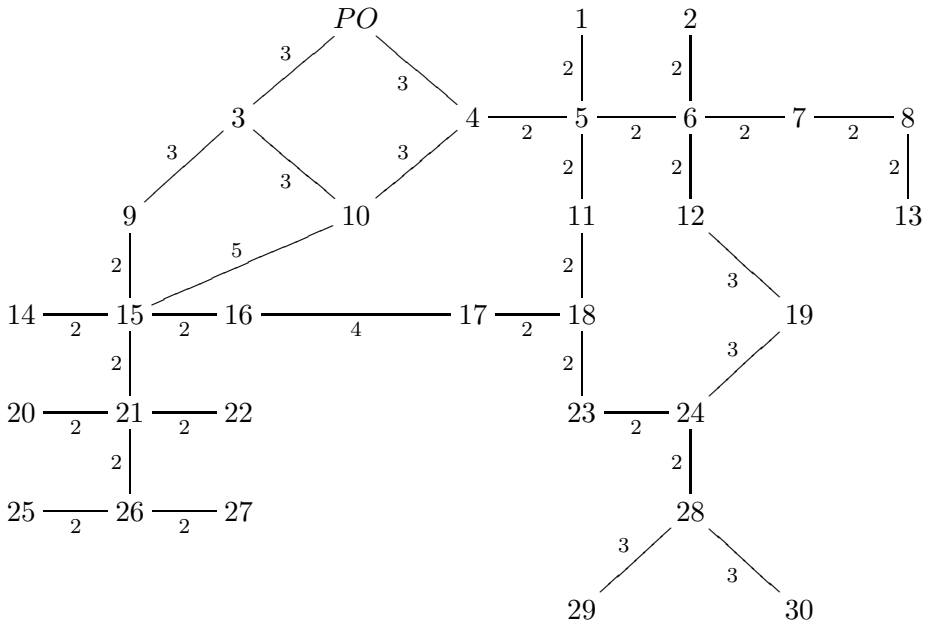

Figure 6.2: This example graph will be used through all examples presented in this chapter

Assume two agents a and b are delivering letters in this graph. The initial letter sets are

$$L_a = \{(PO, 14), (PO, 22), (PO, 27), (25, 22), (17, 27), (14, 16)\}$$

$$L_b = \{(PO, 5), (PO, 7), (23, 19), (10, 2), (30, 8)\}$$

Final destination of a is 20, and final destination of b is 18. Here are the initial plans generated by both parties.

$$\begin{aligned} P_a &= PO \cdot 3 \cdot 9 \cdot 15 \cdot 14 \cdot D_1 \cdot C_6 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot C_5 \cdot 16 \cdot 15 \cdot 21 \cdot \\ &\quad 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot 26 \cdot 21 \cdot 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \end{aligned}$$

$$\begin{aligned} P_b &= PO \cdot 3 \cdot 10 \cdot C_4 \cdot 4 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \cdot 23 \cdot C_3 \cdot 24 \cdot 28 \cdot 30 \cdot C_5 \cdot \\ &\quad 28 \cdot 24 \cdot 19 \cdot D_3 \cdot 12 \cdot 6 \cdot 7 \cdot D_2 \cdot 8 \cdot D_5 \cdot 7 \cdot 6 \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot 11 \cdot 18 \end{aligned}$$

These initial sets and final destinations are used all through the examples, if something contrary is not indicated.

6.3 Example Plan Revisions with Letters Assigned

In the following subsections we would consider the outcome of our system, when new letters are assigned. All new letters are assigned to agent a for simplicity. Agent a is always proposer, while agent b is always acceptor.

When we are specifying the time that the new letter is assigned, we assume plan execution starts at $t = 0$. Each synchronization increments t by one.

Collection of new letter is indicated by C_n, delivery of new letter is indicated by D_n through all examples. Also note that the examples are independent from each other. The new letters discussed are not assigned in order.

6.3.1 A Simple Modification Example

Assume a new letter $(15, 25)$ is submitted to proposer at $t = 5$.

Proposer would first check if collection and delivery nodes of this letter are in the plan path. Since both of the nodes are in the plan path, the agent would not attempt to initiate a resource request or modify his path. He would only insert collection and delivery skills necessary to his resource based plan graph.

Modified plan would be as follows:

$$\begin{aligned} P_a \quad = \quad & 9 \cdot 15 \cdot C_n \cdot 14 \cdot D_1 \cdot C_6 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot C_5 \cdot 16 \cdot 15 \cdot 21 \cdot \\ & 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot D_n \cdot 26 \cdot 21 \cdot 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \end{aligned}$$

Note that the plan does not contain the executed part $PO \cdot 3$.

6.3.2 A Simple Resource Request Example

Assume a new letter $(24, 6)$ is submitted to proposer at $t = 5$. Assume $credit_a = 5$ and $credit_b = 6$

Proposer would first check if collection and delivery nodes of this letter are in the plan path. Since none of them are not in plan path, the agent would initiate a resource

request. The agent would submit the delivered resource for this letter to other agent and ask if he could compensate.

Acceptor checks the cost of delivering letter. Seeing that both collection and delivery nodes of letter are in his plan path, he needs only a collect and deliver skill to insert the letter to his plan. The additional cost he would spend for delivering the letter would be 2. Since this cost is less then $credit_b$, he would produce the requested delivered resource for proposer.

Thus proposer would not need to modify his plan. The acceptor would modify his plan as follows:

$$\begin{aligned} P_b &= 10 \cdot C_4 \cdot 4 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \cdot 23 \cdot C_3 \cdot 24 \cdot C_n \cdot 28 \cdot 30 \cdot C_5 \cdot \\ & \quad 28 \cdot 24 \cdot 19 \cdot D_3 \cdot 12 \cdot 6 \cdot 7 \cdot D_2 \cdot 8 \cdot D_5 \cdot 7 \cdot 6 \cdot D_n \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot 11 \cdot 18 \end{aligned}$$

6.3.3 A Delivered Resource Exchange Example

Assume initial letter sets of a and b are

$$L_a = \{(PO, 14), (PO, 22), (PO, 27), (25, 22), (17, 27), (14, 16)\}$$

$$L_b = \{(PO, 5), (PO, 7), (23, 19), (10, 2), (25, 20)\}$$

Final destination of a is still 20, and final destination of b is still 18. The initial plan of a is same. Here is the initial plan generated by b.

$$\begin{aligned} P_b &= PO \cdot 3 \cdot 10 \cdot C_4 \cdot 15 \cdot 21 \cdot 26 \cdot 25 \cdot C_5 \cdot 26 \cdot 21 \cdot 20 \cdot D_5 \cdot 21 \cdot 15 \cdot 16 \cdot 17 \cdot \\ & \quad 18 \cdot 23 \cdot C_3 \cdot 24 \cdot 19 \cdot D_5 \cdot 12 \cdot 6 \cdot 7 \cdot D_2 \cdot 6 \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \end{aligned}$$

Assume a new letter (29,30) is submitted to proposer at $t = 5$. Assume $credit_a = 5$ and $credit_b = 6$.

Proposer would first check if collection and delivery nodes of this letter are in the plan path. Since none of them are in plan path, the agent would initiate a resource request. The agent would submit the delivered resource for this letter to other agent and ask if he could compensate.

Acceptor checks the cost of delivering letter. Without checking his original plan, acceptor may glue collection and delivery of new letter after C_3. This is the least

costly modification to deliver new letter. He might modify his plan as

$$\begin{aligned} P_b &= PO \cdot 3 \cdot 10 \cdot C_4 \cdot 15 \cdot 21 \cdot 26 \cdot 25 \cdot C_5 \cdot 26 \cdot 21 \cdot 20 \cdot D_5 \cdot 21 \cdot 15 \cdot 16 \cdot 17 \cdot \\ &\quad 18 \cdot 23 \cdot C_3 \cdot 24 \cdot 28 \cdot 29 \cdot C_n \cdot 28 \cdot 30 \cdot D_n \cdot 28 \cdot 24 \cdot 19 \cdot D_3 \cdot 12 \cdot 6 \cdot 7 \cdot D_2 \cdot \\ &\quad 6 \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \end{aligned}$$

Acceptor calculates cost of this modification as 15. This is clearly greater than $credit_b$. However acceptor checks if he might exchange one of his letters with this new delivery.

The individual costs of deliveries and collections are as follows

$$C_4 \to 0, C_5 \to 20, D_5 \to 20, C_3 \to 8, D_3 \to 8, D_2 \to 4, D_4 \to 4, D_1 \to 0$$

These values are calculated as explained in Section 5.4.2. For example C_3 is 8. Because travelling from node of D_5 to node of D_2 costs 26 units with the current plan and it would cost 18 units if L_3 did not exist[3].

Regarding these values, exchanging L_5 with L_n is profitable for acceptor. Thus he accepts the resource request, only if proposer could supply him delivered resource of L_5.

Proposer calculates the cost of delivering this letter. He finds out the collection and delivery nodes of letter are already in plan path and delivering the letter costs him only 2 units. He accepts the exchange and both parties modify their plans. New plans are

$$\begin{aligned} P_a &= PO \cdot 3 \cdot 9 \cdot 15 \cdot 14 \cdot D_1 \cdot C_6 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot C_5 \cdot 16 \cdot 15 \cdot 21 \cdot \\ &\quad 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot C_5^* \cdot 26 \cdot 21 \cdot 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \cdot D_5^* \end{aligned}$$

$$\begin{aligned} P_b &= PO \cdot 3 \cdot 10 \cdot C_4 \cdot 4 \cdot 5 \cdot 11 \cdot 18 \cdot 23 \cdot C_3 \cdot 24 \cdot 28 \cdot 29 \cdot C_n \cdot 28 \cdot 30 \cdot D_n \cdot \\ &\quad 28 \cdot 24 \cdot 19 \cdot D_5 \cdot 12 \cdot 6 \cdot 7 \cdot D_2 \cdot 6 \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \cdot \end{aligned}$$

Note that b removed the collection and delivery of L_5 from his plan. After making the collection before C_5, he directly proceeds for collection after D_5. The shortest path between that previous collection (C_4) and next collection is inserted for the part removed.

[3] 18 is the shortest path distance between nodes 20 and 7.

6.3.4 Rendezvous Examples

Assume initial letter sets of a and b are

$$L_a = \{(PO, 14), (PO, 22), (PO, 27), (25, 22), (17, 27), (14, 16), (21, 17)\}$$

$$L_b = \{(PO, 5), (PO, 7), (12, 23), (7, 23), (10, 12)\}$$

Final destination of a is still 20, and final destination of b is still 18. Here are the initial plans generated by agents.

$$\begin{aligned} P_a &= PO \cdot 3 \cdot 9 \cdot 15 \cdot 14 \cdot C_6 \cdot D_1 \cdot 15 \cdot 21 \cdot C_7 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot D_7 \cdot C_5 \cdot 16 \cdot \\ &\quad 15 \cdot 21 \cdot 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot 26 \cdot 21 \cdot 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \\ P_b &= PO \cdot 3 \cdot 10 \cdot C_5 \cdot 4 \cdot 5 \cdot D_1 \cdot 6 \cdot 7 \cdot D_2 \cdot C_4 \cdot 6 \cdot 12 \cdot C_3 \cdot D_5 \cdot 19 \cdot 24 \cdot \\ &\quad 23 \cdot D_3 \cdot D_4 \cdot 18 \end{aligned}$$

Assume a new letter (6,25) is submitted to proposer at $t = 5$. Assume $credit_a = 5$ and $credit_b = 6$.

Proposer calculates the cost of delivering this letter. Delivery point of letter is already on proposer's plan path. However cost of collecting letter is 16, if proposer proceeds to 6 after collecting L_5 from 17, collects the letter and proceeds to 15. Proposer initiates a resource request for delivering this new letter.

Acceptor calculates the cost of delivering this letter. The collection node of letter is already on his plan path. Cost of delivering the letter is 30, if he directly proceeds to 25 after collecting L_3 from 12. This cost is greater than $credit_b$, thus acceptor seeks for exchanging one of his deliveries with proposer and calculates individual costs of each collection and delivery . However all letters are dependent to each other somehow and removing a letter does not decrease the cost of the plan. Thus acceptor declines the resource request.

When the request of proposer is not satisfied proposal seeks for another alternative. Since delivery node is on his plan path, he requests the have letter resource of the same letter. Another agent may collect the letter and hand to him.

The acceptor agent would be interested in this request, since node 6 is on his plan path. He would accept the request, specifying rendezvous taking place after $t = 15$.

The proposer accepts this exchange since delivering the letter is trivial for him. He acknowledges with estimated delivery time of letter, which is $t = 46$. The rendezvous could not take place after that time.

The acceptor, having the boundary information for rendezvous time, determines his candidates for rendezvous place. He samples some nodes uniformly from his plan path. The number of nodes is 1/5 of nodes visited crudely. The chooses one node (in fact last node) for each five node visited. He also calculates the time he would visit these nodes and use this as candidate rendezvous time for each node. For our examples his only candidate would be

$$24 \rightarrow t = 32$$

Acceptor would send these candidates to proposer. He would also transmit his credit value for possible rendezvous modifications. However he would apply the change for collecting the new letter, while transmitting. He would transmit credit value of 5.

Proposer, receiving the candidate rendezvous points and times, determines if minimum possible cost of rendezvous is feasible. For our example, for being at 24 at $t = 32$ proposer should proceed to 24 after delivering L_6 at 16. Cost of this path is 12. Proposer would be at node 24 at $t = 32$ and would return to 17 at $t = 37$. Note that while passing through 17, proposer would not collect L_5. Also 18 is a better rendezvous point for both parties. But our schema generates a sub-optimal solution, trying to find a solution even in very complex graphs.

For proposer, the cost of this rendezvous is less then delivering the letter, thus proposer accepts this candidate of rendezvous. He calculates the time he could reach 24 and it is $t = 32$. (24,32) pair is the rendezvous coordinates transmitted to acceptor.

The proposer also generates a refined version of the rendezvous. He considers the credit value of the proposer. If there is an intermediate node on path to rendezvous point where acceptor could come with the credit value he transmitted, he would propose that node as an alternative. For our example, acceptor might come to 23, since extra cost would be 4 for him. The proposer also transmits (23,30) pair to acceptor as a refined alternative.

The acceptor gives the final decision. If he could be at 23 at $t = 21$, which is not the case, chooses that alternative, else the original choice of proposer is determined as a rendezvous.

Proposer inserts collection of letter and rendezvous into his plan. Acceptor inserts rendezvous and delivery of letter in his plan. When one agent reach rendezvous

point, he starts executing *exchange to/exchange from* actions. If other party is not at rendezvous point and executing reciprocal action, action would fail. Agent would continue executing the same action until other party arrives and action succeeds.

New plans of agent are

$$\begin{aligned} P_a &= PO \cdot 3 \cdot 9 \cdot 15 \cdot 14 \cdot C_6 \cdot D_1 \cdot 15 \cdot 21 \cdot C_7 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot 18 \cdot 23 \cdot 24 \cdot \\ &\quad 23 \cdot 18 \cdot 17 \cdot D_7 \cdot C_5 \cdot 16 \cdot 15 \cdot 21 \cdot 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot D_n \cdot 26 \cdot 21 \cdot \\ &\quad 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \\ P_b &= PO \cdot 3 \cdot 10 \cdot C_5 \cdot 4 \cdot 5 \cdot D_1 \cdot 6 \cdot C_n \cdot 7 \cdot D_2 \cdot C_4 \cdot 6 \cdot 12 \cdot C_3 \cdot D_5 \cdot 19 \cdot 24 \cdot \\ &\quad 23 \cdot D_3 \cdot D_4 \cdot 18 \end{aligned}$$

Now let's consider a different scenario. Assume a new letter $(9, 8)$ is submitted to proposer at $t = 5$. Assume $credit_a = 5$ and $credit_b = 6$.

Proposer calculates the cost of delivering this letter. Collection point of letter is already on proposer's plan path. However cost of delivering letter is 24, if proposer proceeds to 8 after collecting L_5 at 17. Proposer initiates a resource request for delivering this new letter.

Acceptor calculates the cost of delivering this letter. The delivery node of letter is already on his plan path. Cost of collecting the letter is 12, if he directly proceeds to 9 after collecting L_4 from 10. This cost is greater than $credit_b$, thus acceptor seeks for exchanging one of his deliveries with proposer and calculates individual costs of each collection and delivery . Collection of L_5 has the greatest individual cost in acceptor's letters, which is 10. If these letters are exchanged, acceptor's cost increases by 12 due to new letter, decreases 11 due to collection of L_5, decreases by 5 due to delivery of L_5. Acceptor responses proposer with exchange request with L_5.

Proposer considers the exchange request and clearly this exchange in undesirable for proposer since the cost is 40. Proposer rejects this exchange.

In this situation, there is another exchange alternative that acceptor might try. Acceptor asks proposer for have letter resource of the new letter. This request is logical since acceptor could easily deliver the letter once proposer hands it to acceptor. The rendezvous point and rendezvous time has to be decided, thus acceptor transmits the estimated delivery time with the resource request. This estimated delivery time is important because rendezvous could not take place after that time, it would be too costly for acceptor. In this example the estimated delivery time for new letter is

$t = 47$.

Proposer accepts this exchange since collecting the letter is trivial for him. He acknowledges with estimated collection time of letter, which is $t = 7$. The rendezvous could not take place before that time.

Acceptor, having the boundary information for rendezvous time, determines his candidates for rendezvous place. He samples some nodes uniformly from his plan path. The number of nodes is $1/5$ of nodes visited crudely. The acceptor chooses one node (in fact last node) for each five node visited. He also calculates the time he would visit these nodes and use this as candidate rendezvous time for each node. For our examples his candidates would be

$$23 \rightarrow t = 19$$
$$24 \rightarrow t = 33$$

Acceptor would send these candidates to proposer. He would also transmit his credit value for possible rendezvous modifications. However he would apply the change for delivering the new letter, while transmitting. He would transmit credit value of 5.

Proposer, receiving the candidate rendezvous points and times, determines if minimum possible cost of rendezvous is feasible. For our example, for being at 23 at $t = 19$ proposer should proceed to 23 once after collecting new letter, without visiting 14. Cost of this path is 21. Proposer would be at node 23 at $t = 19$ and would return to 15 at $t = 30$. The cost of this rendezvous is less then delivering the letter. Moreover the cost of other candidate is clearly greater, thus proposer accepts this candidate of rendezvous. He calculates the time he could reach 23 and it is $t = 19$. (23,19) pair is the rendezvous coordinates transmitted to acceptor.

The proposer also generates a refined version of the rendezvous. He considers the credit value of proposer. If there is an intermediate node on path to rendezvous point where acceptor could come with the credit value he transmitted, he would propose that node as an alternative. For our example, acceptor might come to 18, since extra cost would be 4 for him. Proposer also transmits (18,17) pair to acceptor as a refined alternative.

The acceptor gives the final decision. If he could be at 18 at $t = 17$, which is the case, chooses that alternative, else the original choice of proposer is determined as a

rendezvous.

Proposer inserts collection of letter and rendezvous into his plan. Acceptor inserts rendezvous and delivery of letter in his plan. When one agent reach rendezvous point, he starts executing *exchange to/exchange from* actions. If other party is not at rendezvous point and executing reciprocal action, action would fail. Agent would continue executing the same action until other party arrives and action succeeds.

New plans of agent are

$$\begin{aligned} P_a &= PO \cdot 3 \cdot 9 \cdot C_n \cdot 15 \cdot 16 \cdot 17 \cdot 18 \cdot T_n \cdot 17 \cdot 16 \cdot 15 \cdot 14 \cdot D_1 \cdot C_6 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot \\ &\quad C_5 \cdot 16 \cdot 15 \cdot 21 \cdot 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot 26 \cdot 21 \cdot 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \\ P_b &= PO \cdot 3 \cdot 10 \cdot C_4 \cdot 4 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \cdot E_n \cdot 23 \cdot C_3 \cdot 24 \cdot 28 \cdot 30 \cdot C_5 \cdot \\ &\quad 28 \cdot 24 \cdot 19 \cdot D_3 \cdot 12 \cdot 6 \cdot 7 \cdot D_2 \cdot 8 \cdot D_5 \cdot D_n \cdot 7 \cdot 6 \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot 11 \cdot 18 \end{aligned}$$

6.3.5 Replanning Example

Assume initial letter sets of a and b are

$$L_a = \{(PO, 14), (PO, 22), (PO, 27), (25, 22), (17, 27), (14, 16)\}$$

$$L_b = \{(PO, 5), (PO, 7), (23, 19), (10, 2)\}$$

Final destination of a is still 20, and final destination of b is still 18. The initial plan of a is same. Here is the initial plan generated by b.

$$\begin{aligned} P_b &= PO \cdot 3 \cdot 10 \cdot C_4 \cdot 4 \cdot 5 \cdot D_1 \cdot 11 \cdot 18 \cdot 23 \cdot C_3 \cdot 24 \cdot 19 \cdot D_3 \cdot 12 \cdot 6 \cdot 7 \cdot \\ &\quad D_2 \cdot 7 \cdot 6 \cdot 2 \cdot D_4 \cdot 6 \cdot 5 \cdot 11 \cdot 18 \end{aligned}$$

Assume a new letter (19,6) is submitted to proposer at $t = 5$. Assume $credit_a = 5$ and $credit_b = 15$.

Proposer would initiate a delivered resource request and acceptor could handle the delivery of this letter. The $credit_b$ is updated as 13, since cost of delivering letter for acceptor is 2.

Then assume a new letter (29,30) is submitted to proposer at $t = 5$.

Proposer would first check if collection and delivery nodes of this letter are in the plan path. Since none of them are in plan path, the agent would initiate a resource request. The agent would submit the delivered resource for this letter to other agent and ask if he could compensate.

Acceptor checks the cost of collecting and delivering letter. The cost is greater than $credit_b$. The cost is also greater than individual estimate cost of all letters. Moreover the cost of only delivering the letter is also greater then $credit_b$. The acceptor rejects the request.

Proposer initiates a request for *have letter resource* of the same letter. The cost of only collecting letter is also greater than $credit_b$. Acceptor rejects that request too.

Thus proposer modifies his plan to collect and deliver on his own. Here is the modified plan.

$$\begin{aligned} P_a \;=\; & PO \cdot 3 \cdot 9 \cdot 15 \cdot 14 \cdot D_1 \cdot C_6 \cdot 15 \cdot 16 \cdot D_6 \cdot 17 \cdot C_5 \cdot 18 \cdot 23 \cdot 24 \cdot 28 \cdot 29 \cdot C_n \cdot \\ & 28 \cdot 30 \cdot D_n \cdot 28 \cdot 24 \cdot 23 \cdot 18 \cdot 17 \cdot 16 \cdot 15 \cdot 21 \cdot 26 \cdot 27 \cdot D_3 \cdot D_5 \cdot 26 \cdot 25 \cdot C_4 \cdot 26 \cdot \\ & 21 \cdot 22 \cdot D_2 \cdot D_4 \cdot 21 \cdot 20 \end{aligned}$$

6.4 Discussion on Complexity of Implementation

The complexity of algorithms were discussed in previous chapters. However the complexity issues about some implementation details are also important.

Firstly, we choose syncronization intervals of 0.5 sec. for our systems. All the plan revision process is expected to end in a single syncronization interval, thus 0.5 sec. is a reasonable interval length for our system. The implementation may work flawlessly with smaller syncronization intervals, but this lenth is sufficient for our performance criteria.

Secondly, the negotiation phase between agents always terminate in a few message passing. The longest interaction schema consists of:

- Proposer requests a delivered resource.
- Acceptor asks for a have letter resource in return.
- Proposer agrees and specifies rendezvous constraints.
- Acceptor generates rendezvous candidates.
- Proposer does not accept rendezvous candidates.
- Proposer requests a have letter resource.
- Acceptor agrees and specifies rendezvous constraints.

- Proposer agrees and specifies rendezvous constraints.
- Acceptor generates rendezvous candidates.
- Proposer generates his preferences among candidates and an alternative.
- Acceptor determines rendezvous.

This scenario consists of 11 messages send and received. For a two agent setting messages are sent and received immediately. Generaly an agent waits for 0.005 sec. for message delivery[4] Thus 0.5 sec. is far more relaxed interval for all communication to terminate.

[4] Continues waiting if message is not delivered yet.

CHAPTER 7

CONCLUSION

In this work we proposed a plan coordination framework, at which agents coordinate their plans regarding to the dynamically changing environment. The plan representation and plan revision schemas designed provide an effective and simple way to interact with different agents.

The most important feature of the framework is the generality it proposes. The negotiation, initial plan construction and re-planning algorithms are all proposed independent of the resource based representation framework. The resource based representation and using resource exchanges are two key factors of the framework. Once they are used in the system, different planning algorithms could be used for generating the initial plan, different negotiation protocols could be applied for achieving a resource exchange that is beneficial for all parties and different cooperation concepts could be used for making decisions in negotiation phase.

Despite its generality, the plan revision schema used by agents in the postmen domain produced many interesting results of cooperation as shown in previous chapter. The main factor providing these successful results is the portability of the negotiation protocol to plan representation schema.

The replanning and negotiation schemas are designed to find effective plan modification and cooperation interaction where possible. However, as examples of previous chapter reveal, the output cooperation interaction is not always optimal. In fact generating optimal interactions require a central system that have all the knowledge on agents' plans and possibly a better re-planning unit. Also the replanning algo-

rithm does not make major modifications to initial plan, thus is not optimal clearly.

The sub-optimal solutions are in fact inevitable for more time critical domains, since the agent might not have much time to respond to changes in environment. In such environments it might be vital to determine or estimate the response deadline before starting to replan or interact with other agents. Then, according to response deadline, a solution closer to optimal can be generated.

The negotiation mechanism used is designed as simple as possible. There are finite states in the mechanism and no looping, thus there is no possibility of non-termination. The longest chain of negotiation consists of eight messages. Since negotiation mechanism is not long, it is easily executed in one synchronization interval.

The credit schema used is a simple measure on cumulative interactions with other agents and own utility. It prevents accepting all requests. Moreover the more requests satisfies an agent, the easier his requests would be satisfied by others. This simple schema reinforces cooperation in many ways.

We did not discuss the performance of the system in details. Our domain is not strictly real-time, thus response time is not critical. However we also expected to make plan re-assessments as efficiently as possible, not spending any time interval idle, or executing irrelevant actions. Our system satisfied this criteria too, for reasonable synchronization intervals.

Most important defect of the system is absence of a planner that works directly on resource based plan graphs. It is an advantage to be able to use planners regardless of their representation, but for some simple planning or re-planning a built in planner that works directly on resource based graph might be beneficial.

One possible extension to this work is an auction mechanism for new letter assignments. If environment distributes the letters by auction and pays agents for delivering letters, agents would have to reason when bidding for new letters. The plan revision framework can be extended to forming bids and sharing tasks with other agents.

One other possible extension is gradually processing the new letters and utilizing the computation time. For example when a new letter is to be inserted into plan, agent could divide the plan graph into two parts: part of plan that needs modification and part of plan that is irrelevant to delivery of new letter. Then agent could estimate

when modified part would be executed and build a modification schedule to use all the time until execution to generate a better plan.

In fact there are numerous extensions possible, some of the future work mentioned in [10] like auction related protocols, market based coordination and handling conflicting situations can also be applied here. The generality of the system is the most important asset of this work and makes this framework not just a solution to a problem, but a conceptual point for applying new ideas and techniques.

REFERENCES

[1] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3–4, pp. 189–208, 1971.

[2] C. Boutilier and R. I. Brafman, "Planning with concurrent interacting actions," in *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, (Providence, Rhode Island), pp. 720–726, AAAI Press / MIT Press, 1997.

[3] D. Mcdermott and M. H. Burstein, "Extending an estimated-regression planner for multi-agent planning," in *Proceedings of the AAAI-2002 Workshop on Multiagent Planning*, August 2002.

[4] M. Bowling, R. Jensen, and M. M. Veloso, "A formalization of equilibria for multiagent planning," in *Proceedings of the AAAI-2002 Workshop on Multiagent Planning*, August 2002.

[5] E. Ephrati and J. S. Rosenschein, "A heuristic technique for multiagent planning," *Annals of Mathematics and Artificial Intelligence*, vol. 20, no. 1–4, pp. 13–67, 1997.

[6] J. S. Cox and E. H. Durfee, "Discovering and exploiting synergy between hierarchical planning agents," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 281–288, ACM Press, 2003.

[7] P. J. Gmytrasiewicz and E. H. Durfee, "Rational coordination in multi-agent environments," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 4, pp. 319–350, 2000.

[8] S. Kraus, J. Wilkenfeld, and G. Zlotkin, "Multiagent negotiation under time constraints," *Artificial Intelligence*, vol. 75, no. 2, pp. 297–345, 1995.

[9] M. M. de Weerdt and R. P. van der Krogt, "A method to integrate planning and coordination," in *Planning with and for Multiagent Systems* (M. Brenner and M. desJardins, eds.), no. WS-02-12 in AAAI Technical Report, (Menlo Park, CA), pp. 83–88, AAAI Press, 2002.

[10] J. Tonino, A. Bos, M. M. de Weerdt, and C. Witteveen, "Plan coordination by revision in collective agent-based systems," *Artificial Intelligence*, vol. 142, no. 2, pp. 121–145, 2002.

[11] N. Short and L. Dickens, *STRIPS: What All Others Are Measured Against*, June 1992.

[12] S. Russell and P. Norvig, *Artifical Intelligence: A Modern Approach,*. Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-103805-2, 912 pp., 1995, 1995.

[13] H. Ulusoy, "Forming coalitions through negotiation in multi-agent environments," Master's thesis, Middle East Technical University, Department Of Computer Engineering, December 1999.

[14] J. S. Rosenschein and G. Zlotkin, "Designing conventions for automated negotiation," *AI Mag.*, vol. 15, no. 3, pp. 29–46, 1994.

[15] T. W. Sandholm, "Distributed rational decision making," in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (G. Weiss, ed.), pp. 201–258, Cambridge, MA, USA: The MIT Press, 1999.

[16] R. Sedgewick, *Algorithms in C*. Addison-Wesley Longman Publishing Co., Inc., 1990.