NTNU – Trondheim
Norwegian University of
Science and Technology

Recognizing and Learning Opportunities in Complex and Dynamic Environments

Fredrik Åsgård

Master of Science in Computer Science
Submission date: July 2015
Supervisor: Pinar Öztürk, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Recognizing and Learning Opportunities in Complex and Dynamic Environments

Fredrik Åsgård

June 2015

MASTER'S THESIS

Department of Computer and Information Science
Norwegian University of Science and Technology

Abstract: *An opportunistic agent need not only to identify, learn to recognize and to exploit opportunities. This is of particular interest in complex environments, where an agent is unable to attain full overview of the situation. Real-world environments are riddled with uncertainty, there are changes taking place everywhere, agents have severely limited observability, and there is no realistic way to evaluate all possible states/actions. We adapt a proven model of temporarily suspending goals (instead of permanently discarding them), should the goals constraints become invalidated. We propose a conceptual framework that uses reinforcement learning on observations in a Partially observable Markov decision process (POMPD) for learning to recognize future opportunities. The learned opportunities are combined with partial-specification planning in order to enable an agent to achieve its goals.*

Supervisor 1: Pinar Özturk
Supervisor 2: Kerstin Bach

Preface

This thesis concludes my Master of Science education in Computer Science at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. The thesis was performed throughout my 10th semester, spring of 2015, at the Department of Computer and Information Science.

Working on this thesis was very intense work, with hours upon hours of coding. Unfortunately, in the end only a very simplified implementation was still functional. There was simply not enough time and too many details that needed ironing.

I hope the contents of this document may become useful for those interested in learning to recognize opportunities.

I would like to thank my supervisors associate professor Pinar Øzturk and postdoctoral fellow Kerstin Bach for valuable ideas and comments throughout the work period. A big thank also goes to my friends who kept me motivated during the most intense periods.

Trondheim, 2015-07-03

FREDRIK ÅSGÅRD

Abstract

An opportunistic agent need not only to identify, learn to recognize and to exploit opportunities. This is of particular interest in complex environments, where an agent is unable to attain full overview of the situation. Real-world environments are riddled with uncertainty, there are changes taking place everywhere, agents have severely limited observability, and there is no realistic way to evaluate all possible states/actions. We adapt a proven model of temporarily suspending goals (instead of permanently discarding them), should the goals constraints become invalidated. We propose a conceptual framework that uses reinforcement learning on observations in a POMPD for learning to recognize future opportunities. The learned opportunities are combined with partial-specification planning in order to enable an agent to achieve its goals.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

We interact with uncertain, dynamic environments in our everyday lives, yet this has proved to be a daunting task for Artificial Intelligence (AI). It's become evident that creating agents capable of working in such environments is not simply a matter of filling in knowledge. A static knowledge base is quickly rendered obsolete in a rapidly changing environment. Additionally, for sufficiently complex environments we can't hope to anticipate every possible situation. Some circumstances even severely restrict our ability to reason about the environment in advance. As a result an intelligent agent must perform even with very limited prior domain knowledge, and potentially wast amounts of learned knowledge.

Many real-world domains have been characterized as messy. These domains have a weak theoretical basis, and there may be a large number of potential outcomes to every situation and an inherent uncertainty. If other agents are present one must also consider their interactions with the domain. This makes it imperative that intelligent agents are able to adapt to unforeseen changes in the environment, and also to learn from opportunities and failures (Hammond et al., 1993). Learning to recognize opportunities relies on learning from past experiences and understanding what caused the opportunity. However, the first step towards learning to anticipate opportunities is to recognize a circumstance where an opportunity may present itself (Pryor, 1996).

There exists a few different definitions of opportunism and opportunistic agents in the scientific literature (Francis, 1995; Pryor and Collins, 1994; Schank and Leake, 1989). We will use the following definition throughout the thesis unless stated otherwise:

> An opportunity is any circumstance in which selecting a particular course of action allows the agent to reach some goal with less effort than initially foreseen.

That is not to say that certain opportunities may not have an associated risk. In order to be opportunistic a system needs to be able to recognize opportunities, but also assess their worth with regard to the potential risks and rewards. The circumstances necessary for an opportunity may arise due to several different reasons. Examples include a sudden availability of a resource or tool, a change in the agents knowledge about the domain, a change in the external world, and more (see Section 3.2 on page 18).

One way of tackling these domains is by reusing strong knowledge, as in Case-Based Reasoning (CBR) systems. In essence these systems propose new solutions by adapting solutions from previous problem instances. Some domain knowledge can be used to improve case adaptation. While classical approaches were not intended for large cases with few important features, the similarity measure and retrieval is useful for discovering similar circumstances. However, in classical CBR designs, the quality of the similarity measures and adaptation knowledge are not within the scope of the learning process. When combined with other machine learning techniques CBR allows us to identify new opportunities by analysing the observations that lead to an opportunity.

This thesis focuses on designing a conceptual model that uses past experiences to identify and recognize opportunities in complex environments.

1.1 Objectives

Our objectives for this thesis is to define a conceptual framework that allows us to identify, recognize and learn several different types of opportunities in a complex environment, as well as exploiting the opportunities when appropriate. We will look at requirements for an autonomous agent that is placed in an unknown symbolic environment, with respect to knowledge representation, planning, execution and learning under soft real-time constraints.

This thesis focuses on the following research goals:

1. *Creating a framework for identifying opportunities in complex environments*

Identifying an opportunity entails knowledge of the circumstance necessary for the opportunity to arise, and reacting when such a circumstance is being observed. This can be particularly interesting in complex environments as observations may consist in large parts of noise with respect to the opportunity.

2. *Creating a framework for recognizing opportunities in complex environments*

   Once we have an idea about the circumstances that need to be for an opportunity to arise, we can index specific actions accordingly. The challenge is to create a framework that allows us to recognize as many types of opportunities as reasonable through different indexing strategies.

3. *Finding a representation of opportunities for reasoning and recognition*

   Assuming an agent knows the circumstance for a specific opportunity, we still need find a representation of this information in order to enable the actual reasoning and recognition.

4. *Specifying what an agent should learn in order to recognize future opportunities*

   In a complex environment there are many observed features that may have no relevancy for the opportunity. Remembering everything in order to analyse this information at a later time is likely intractable. We are interested in finding a method to narrow down which features without reducing the capability to recognize new opportunities.

5. *Specifying when an agent should create a new case in order to recognize future opportunities*

   As important as what the agent should focus on learning; *when* to form new memories are also highly interesting.

6. *Specifying how an agent should decide whether to pursue the recognized opportunity*

   Some measure of control must be applied when deciding whether to pursue an opportunity. At any given time frame, there are likely many opportunities that arise. There may even be several opportunities between two consecutive observations. Most of these opportunities are unlikely to improve the current plan, and we must be able to decide which.

1.2 Structure

The rest of the thesis is split into chapters and structured as follows.

- Chapter 2 on the next page (Background) covers basic theory about opportunism, planning, and CBR, in addition to an outline of the related work.
- Chapter 3 on page 11 (Conceptual Model) describes the components of the domain model and the agent model, and how they relate to the research goals.
- Chapter 4 on page 21 (Experiments) explains how the experiments were set up and performed, and also lists relevant results.
- Chapter 5 on page 27 (Discussion) discusses the validity of the approach and the results of the experiments. There is also a section on limitations of our conceptual framework.
- Chapter 6 on page 31 (Conclusions and further work) contains a short summary of the contents in this thesis, in addition to the conclusions and recommendations for future work. Finally, there is a short section on additional work.

Chapter 2

Background

Mastering the real world around us is challenging for computers because of uncertainty, complexity and dynamism. Any intelligent system intended to operate in the real world needs to handle the inherent difficulties such a world presents (Pryor and Collins, 1994). While our agent only exists in a symbolic environment, we intend to model these challenges. A large world entails that an agent can never know everything there is to know. In a dynamic world, changes are caused by several agents and/or processes. We define a dynamic world to include the creation and destruction of objects. Finally, uncertainty makes it impossible to know the exact outcome of an action or the state of the unobserved world. These properties allow us to model such an environment as a Partially observable Markov decision process (POMPD). We will assume that the world our agent shall operate in has all of these properties.

In order to achieve goals an agent must formulate a plan. However, complexity, uncertainty and dynamism make it very difficult to create detailed plans. At any point during the execution, a precondition may be invalidated and the rest of the plan rendered useless. This makes classical planning, where a problem is solved precisely by listing all the required steps, an unsuited approach to achieving goals in such worlds. Even when looking at very efficient planners such as the Hierarchical Task Network (HTN) planner SHOP2 (Nau et al., 2003), it quickly became intractable to do planning even in deterministic worlds of significant size. However, there are several possible approaches that still appear tractable in complex, dynamic and uncertain environments. Partial-specification planning (also sometimes know as just partial planning) techniques, while also working well within opportunistic reasoning, may allow

tractable implementations (Mancarella et al., 2005; Weld, 1994). A planner that only partially specifies the plan and keeps alternative plans could be said to find a strategy for solving the problem. The actual execution of each step is deferred until the agent has enough information to carry out the primitive actions, and every action may change as the agent carries out the plan.

Furthermore, learning to recognize and remember circumstances that are favorable (i.e. opportunities) requires memory. The less one knows about the domain in advance, the more general the memory needs to be. Our assumption is that we can only have limited knowledge about the domain in advance, and thus our model for memory must adapt to previously unseen data. Since the environment we need to remember is symbolic, there is an upper size limit on the set of states. This suggest that we could look at Markov decision processes. Also, we have an uncertain and complex environment, which in turn makes reinforcement learning an attractive candidate. Incidentally, reinforcement learning is also suited to large environments. We are also interested in using the memory structure in order to learn future opportunities. Instance-based learning and case-based reasoning are interesting candidates which both generalize observations

2.1 Opportunism

A more pragmatic definition of opportunism is that opportunistic behavior is the process of recognizing circumstances in which a suspended goal may again be pursued Simina and Kolodner (1995). While working towards a goal an agent may encounter a problem which prevents an immediate sub-goal from being achieved. Instead of abandoning the task in its entirety, the agent may choose to temporarily suspend the sub-goal and begin work on another problem. Eventually this may lead to new evidence or insight about the suspended sub-goal, and the deferred work can be resumed. In this definition the opportunity itself can be thought of as the proposition as to how a current or future challenge may be overcome. In many cases this challenge will be to overcome the perceived cost of solving the problem.

There are several required steps towards a fully opportunistic agent:

- **Identification of opportunities**: Identification is linked to observation. The agent needs to be capable of identifying an opportunity in the observed data.

- **Learning opportunities**: Once we have identified an opportunity, the agent can learn the circumstances required for the opportunity to arise.
- **Recognizing learned opportunities**: This step is similar to identification, but occurs after the opportunity has been learned. Recognizing is focused on quick evaluation of the benefits of the opportunity.
- **Exploitation of opportunities**: The process of deciding whether to pursue the recognized opportunity or not, given that not all opportunities improve utility at all times.
- **Chasing opportunities**: If the agent believes that a certain circumstance may improve the current plan by a significant amount, it could decide to actively go looking for an opportunity.

This definition of opportunity can be encapsulated through storytelling: A man wants to cook bacon and eggs for breakfast, but discovers that he is out of eggs. Realizing he won't have time to go to the store and return in order to make eggs without being late for work, he decides to eat something else. While driving to work he sees a storefront advertising a limited offer on eggs. Upon recalling that he is out of eggs he decides to quickly stop and buy a dozen before going on his way.

In this example the travel to work is temporarily suspended in order to work towards another goal. Even though he gains no momentary benefit from purchasing the eggs, this course of action allows him to immediately satisfy a precondition for a latent goal at low cost.

From the perspective of a planning agent, opportunities should be seen somewhat in contrast to planning failures. Whereas planning failures indicate when an agent need to avoid negative interactions, opportunities are about exploiting positive interactions according to Hammond et al. (1993). It could be argued that for every planning failure there must be some missed opportunity. However this may not always be true. Following the example; if the man had planned ahead by purchasing eggs the previous day he would have no use of buying more, missing the opportunity to save money on the special offer. While certain opportunities may arise at the planning stage, other are only revealed during execution.

Without knowing the schedule of promotional offers of grocery stores in his vicinity, the man had no realistic way to know that the eggs would be on sale. However, after acting on the opportunity he may now pay special attention to this particular store in case a similar offer appears. This reflects one way in which acting upon opportunities ties in with learning.

If one argues that it is impossible to predict such futures, one should at least react should a more favorable future occur. It is also possible to plan for futures that may happen, e.g. anticipate opportunities. Part of being able to utilise such opportunities is to unify how planning and execution works (Pryor and Collins, 1994). In the objectives following this section, we look closer at how we intend to realize an agent that is able to cope with the circumstances mentioned above.

The several different classifications of opportunities. An opportunity may arise because of a change in the environment (exogenous) or in the agents beliefs (endogenous). The change may be discovered by passively observing or actively chasing an opportunity. This can happen during planning-time or execution-time, and may affect the agents current goal or behavior. Finally, the opportunity could arise as a consequence of change in availability of a resource or tool, a change of the agents location, or simply because of passing time.

In order to successfully exploit opportunities, an agent must first identify the opportunity. There has been done much research on the identification and recognition of opportunities previously. See (Hammond et al., 1993; Hayes-Roth, 1993; Pryor and Collins, 1994; Schank and Leake, 1989) for a good starting point. The next step is to decide whether to pursue the opportunity, as not all opportunities end up increasing the utility of the current plan. Francis (1995) proposes a utility controlled approach in order to ignore certain opportunities. Finally, the agent could learn the circumstances for the opportunity, in order to actively look for opportunities in the future. In this thesis, we intend to look at learning opportunities and to recognize them in real-time in complex environments.

2.2 Related work

2.2.1 Trucker

TRUCKER was developed to deal with the problem of recognizing execution-time opportunities in the context of a resource-bound agent that is forced to suspend planning in order to attend to execution (Hammond et al., 1993). The goal of this model was to capture the ability of an agent to suspend goals, yet still recognize execution-time opportunities to satisfy them. In response to a suspended goal, TRUCKER performs a planning-time analysis of the conditions that would favor the satisfaction of the goal. It then suspends the goal in memory, indexed by a description of those conditions. Once those conditions are present, the goals are reevaluated in terms of the new conditions. Either the goals fit into the current plan or they are again suspended.

2.2.2 Pareto

PARETO operates a simulated robot delivery truck in an unpredictable world: The truck has limited perception, and can sense only those objects that are at its current location. The world is also dynamic, since objects may spontaneously change location, appear, or disappear. Finally, the results of the truck's actions are uncertain (Pryor, 1996; Pryor and Collins, 1994). PARETO receives delivery orders at unpredictable intervals during execution-time. Calculating plans that allow for every possible combination of goals is intractable; instead it uses a separate plan for each of its goals. These plans are only partially-specified, and PARETO can choose *primitive actions* during execution-time.

2.2.3 Nicole

NICOLE is divided into three components: Memory, reasoning, and control. These three components work together in order to allow an opportunistic memory (Francis, 1995). The reasoning components is tasked with limiting what reminders the memory component produces while the control component decides if the plan should change. The control component, TASKSTORM, continuously evaluates what it can do, deciding what it should do, and deciding to change what it is currently doing.

2.2.4 Opportunistic Control

The *opportunistic control model* comprises three component processes (Hayes-Roth, 1993). An agent must be able to react to event during execution-time. In order to control the direction of the agents overall actions, a strategic planner dynamically modifies a strategy that constrain the intended actions. Together, these two aspects allow the agent to perform a match-based control process, selecting specific *primitive actions* to execute which satisfy the overall strategy.

Chapter 3

Conceptual Model

Our target environment is complex; There are potentially an infinite number of states. The state of the environment is only partially visible to an agent. There is an inherent uncertainty about the outcome of actions. There are other agents which may also interact with and change the environment. There are, however, a few assumptions that help us reduce the complexity slightly; Observations made by the agent are correct (no inaccuracies or noise). The agent has perfect memory.

There are a few important consequences that arise from those points. Primarily, we can not expect to be able to perform precise planning (Hauskrecht, 2000); The action and state space is too large and any plan that details a course of action may be obsolete by the time we receive the next observation. As such, we need to formulate an overall strategy to achieve our goals that will not be easily obsoleted[1], then during execution quickly evaluate our possible courses of actions to achieve each step in the overall strategy. This is sometimes referred to as partial-specification planning or just partial planning. Hierarchical Task Network (HTN) planners allow us to achieve the above given we allow specification of *abstract actions*. An abstract action is not a directly executable as is, but functions as a starting point for a specialized planner. Path-finding would be a prime example, where the abstract action could be `Goto`. The HTN planner only needs to know the result of executing the action and the associated cost, while the specialized planner is first consulted when that action is up for execu-

[1] There is another possibility of instead approximating the solution. However, this is more relevant when set of actions is very large and the result of an action is continuous. Example: rotating a robot arm and arbitrary amount, with an error $\epsilon > 0$. Situations like this does sometimes not require a precise rotation, and it may be corrected later on.

tion.

While our environment potentially contains multiple agents, we do not consider it to be a multi-agent problem. Each agent is not directly aware of the existence of any other intelligent agents and we do not consider behavior modeling. Interaction between agents is modelled the same way as any other interaction in the environment, through the execution of actions. This enables cooperative planning if the actions are tailored as such, while still allowing us to model our problem as a Partially observable Markov decision process (POMPD). A POMPD is a Markov decision problem in which an agent knows about the set of states, actions, transitions between states and the associated reward in addition to a set of observations. As defined above, the agents observations are without error, but beliefs about no longer visible parts of the environment are uncertain.

3.1 Knowledge representation

We model our representation of a complex environment based on a set of `concepts`. Each concept describes a class of objects and has zero or more `attributes` and/or `actions`. E.g. a door might be closed (attribute) and you could lock/unlock(actions) it. Attributes and actions are collectively described as *members* of the concepts, and we refer to a specific attribute or action using the following notation throughout the thesis: The concept name followed by a period and the member name (`Concept.member`).

Concepts are organized in an inheritance hierarchy (see Figure 3.1 on the facing page) in order to facilitate generalization of induced rules for opportunities. We will revisit rule induction and generalization of opportunities in Section 3.2 on page 18. Concepts closer to the top in the hierarchy are *generalizations* of concepts further down. Concepts closer to the bottom are *specializations* of entities further up in the hierarchy. Specializations can have more than one parent and inherit all the attributes and actions of the parent concepts.

Concepts themselves do not exist in the environment. They are merely a representation of a class of objects that an agent may utilize when reasoning about the environment. *Instances* of concepts on the other hand, exist in the environment and agents may directly observe and interact with them (see Figure 3.2 on page 14). Each concept may have more than one instance in the environment at any time. As per our previous definition of dynamic environments; instances may also be created and

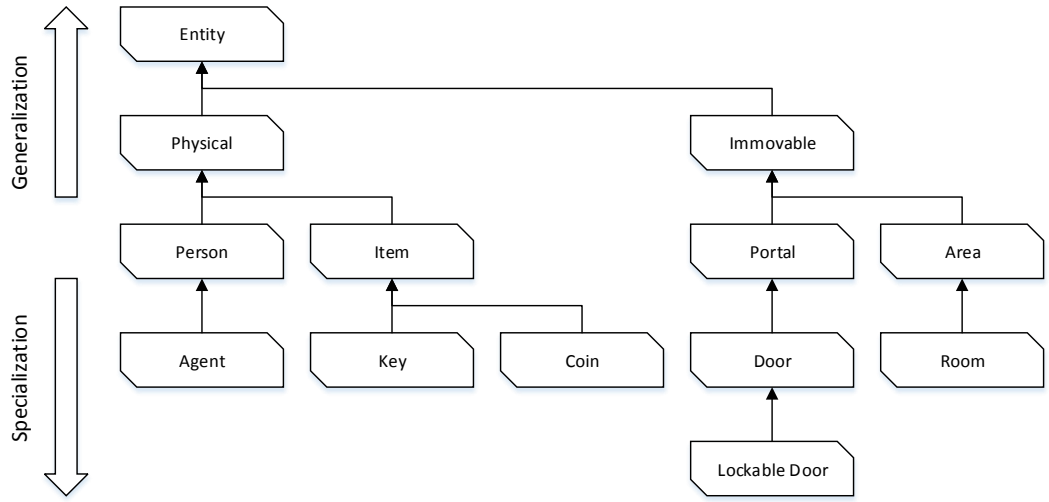

Figure 3.1: An example of a concept inheritance hierarchy.

destroyed at any time.

3.1.1 Attributes

Concept attributes are straight forward. They are strongly typed and can be restricted to a range of values or transitions by an optional data model. Attributes are inherited from the parent concept. This means that if the parent concept has an attribute, the specialization has an exact copy of the same attribute. Note that the attributes are not assigned any actual values; This is deferred until a concept has been *instantiated*, as every instance may have different values for each attribute (see Table 3.1 on page 15).

Once instantiated, every attribute is assigned an unambiguous value. The value might change between observations and before the agent has a chance to perform an action. An agent may only observe a subset of all attribute values at a time, as a result of partial observability of the environment. Only instances in the same location as the agent may be observed, along with their attribute values.

While the exact types of attributes depend on the implementation, there are three main classes that each type belong to:

- **Enumeration**: a relatively small set of values (E.g. weekdays). Enumerations are the simplest, and are candidates for exact planning and reasoning in small

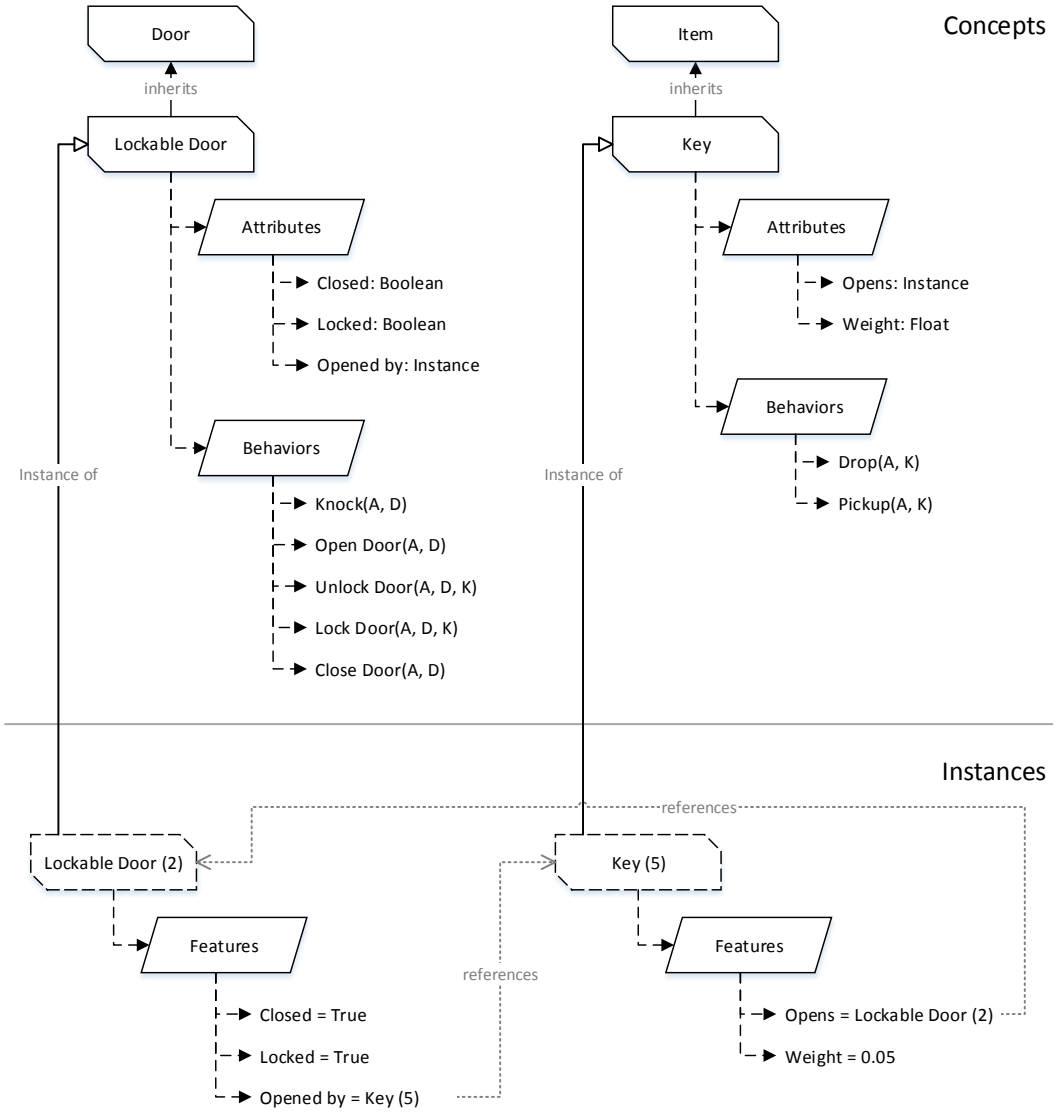

Figure 3.2: An example of concept instances and how they reference each other.

Attribute examples

Name	Type	Model	Example value
World.Time	Time	24h-Time	17:07
World.Weather	Enumeration	{Sunny, Rainy, ...}	Cloudy
Store.Location	Instance	instance-of Location	Manhattan
Store.Open	Boolean	{False, True}	True
Agent.Name	String	String	"Jarvis"
Agent.Money	Integer	$x \in [0, 10^{10}]$	133
Agent.Happiness	Float	$x \in [0.0, 1.0]$	0.8

Table 3.1: A list of a few example attributes

environments.

- **Discreet**: a large range of values (E.g. integers). Discreet values need intelligent planning, and classical planners may still perform reasonably well. The term reasonable used here is not an absolute measure, more of a guideline for real-time performance time constraints.
- **Continuous**: a practically infinite range of values (E.g. floating-point values). Continuous values makes planning very difficult, and classical planning may no longer be applicable. Partial-specification planning is a likely candidate, aside from specialized planners or approximations.

Our environment does potentially contain continuous attributes (recall that we assume a continuous environment). Because of this, we need to be selective with our approach to interaction. The actions that are available to the agent during planning must work well with partial-specification planning.

3.1.2 Actions

Actions (sometimes called *operators* or *primitive tasks* in planning literature) are the only mean by which an agent can interact with the environment, as opposed to simply observing. Each action is considered in a context (a subset of the environment). As an action may never change anything outside of the context, planning and evaluation can be limited exclusively to instances that are part of the context. Additionally, each action has a set of constraints that must be satisfied before execution, and a set of potential outcomes (see Listing 3.1 on the following page). While the outcomes and

their probabilities are specified in the model, the agent does not know the probabilities. The probabilities may even depend on the current state.

```
1  action Open {
2    context:
3      agent instanceof Agent
4      door  instanceof Door
5    constraints:
6      door.closed = true
7      door.locked = false
8      door.location = agent.location
9    results:
10     outcome 0.99 {
11       door.closed := false
12     }
13     outcome 0.01 {} // door got stuck somehow
14 }
```

Listing 3.1: An example specification of an action.

All the actions in the environment are organized in a hierarchical network, called a Hierarchical Task Network (HTN) (see Figure 3.3 on the next page). In addition to actions (diamonds), there are methods (rectangles) and goals (ellipses) collectively referred to as *tasks* or *subtasks*;

Methods are sequences of actions, other methods and goals. They complete more advanced tasks and must be recursively decomposed into a series of just actions before the agent may execute them. Methods, like actions, also have constraints and results. The difference is that the result is only true when the method is completed, as subtasks may have conflicting results. During the execution of the method, the state of the world can only be determined by analysing the decomposed subtasks.

Goals specify some conditions which should be fulfilled. Goals are potentially fulfilled by one or more methods and/or actions. In continuous environments, goals specifying exact states are not suited for planning. We instead need to specify the intentions of each goal (e.g. instead of a goal `agent.money = 100.0` we could say the goal is `increase agent.money` and give the larger increment higher utility). In order to apply bounds on goals, but not strictly specifying a value, we can use inequalities (e.g. `agent.money > 100.0`) so that the goal can be completed.

Given an agents current knowledge of the environment, a behavioral graph is built

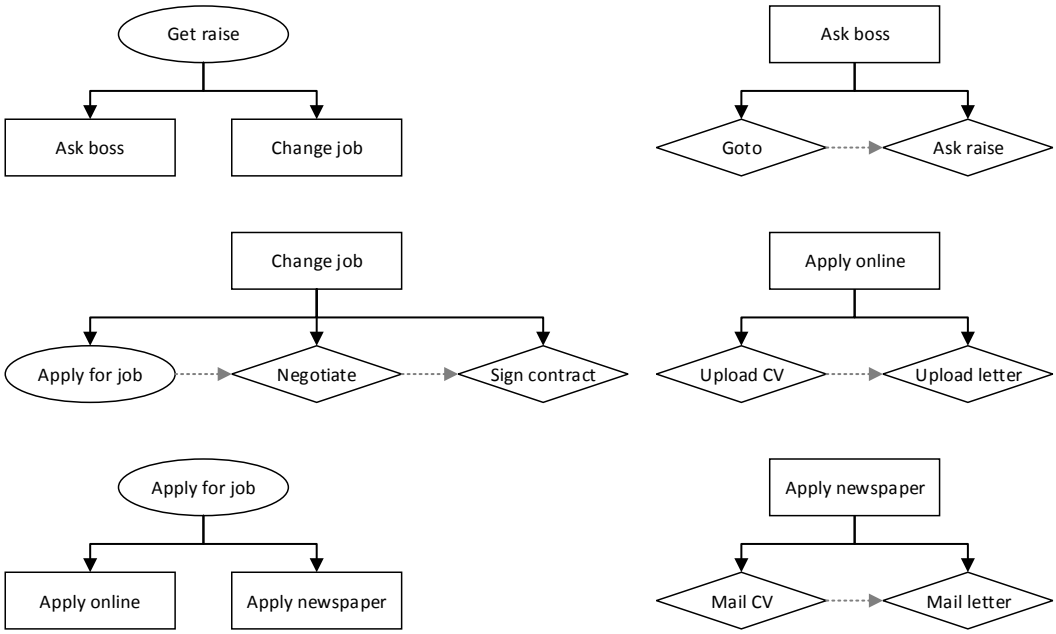

Figure 3.3: Actions, methods and goals in a hierarchical task network

from the task network. The graph is directed and acyclic with only one root node, and describes the complete strategy an agent will use to achieve its goals. Towards the top there are general goals, and further down there are more specific actions (see Figure 3.4 on the following page).

Goals, methods and actions may all be suspended if the constraints are not satisfied at the time of execution. Methods are also suspended once any of its sub-tasks are suspended. Goals are suspended once all of the sub-tasks are suspended. Once the constraints are satisfied the action may be resumed. We assume that the agent has one or more active goals at all times.

When the agent is introduced into the environment, it does not know all the actions it can perform. One of the few things the agent is 'programmed' to know are the primary goals. If an agent satisfies all the primary goals, it has successfully solved the complete problem. As the agent explores the environment, it discovers new concepts and learns new actions. When this happens, the agent can attempt to create a new strategy for achieving its primary goals. An assumption that we make here, is that once the agent learns a new action, it understands where in the task network it

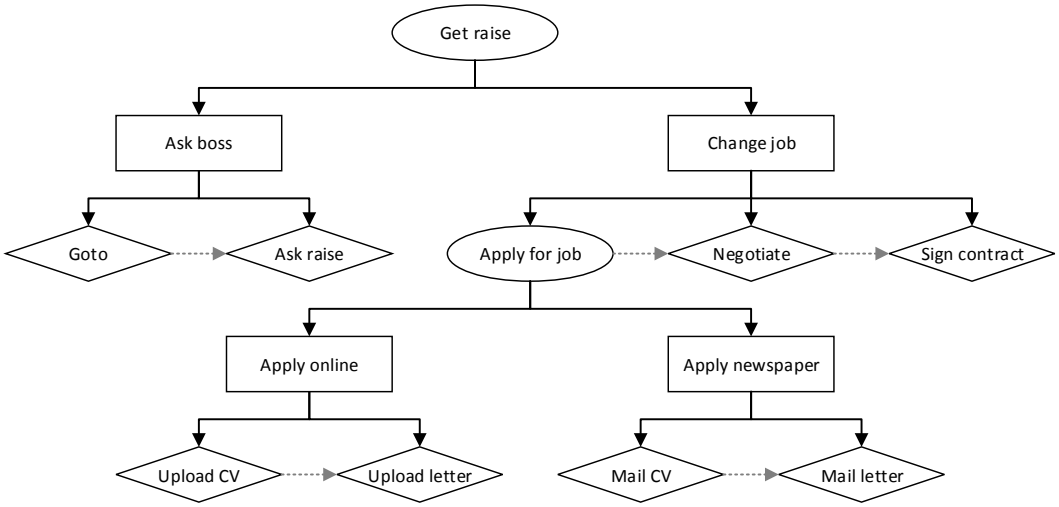

Figure 3.4: One potential strategy for getting a raise

is supposed to go (this is possible since the entire task network is specified at design time). This means that once an action has been learned, the methods and (sub-)goals are discovered as well.

The main motivation behind concept and action discovery is to handle execution-time opportunities that affect the current action or goal. As an example, this allows us to model scenarios in which an agent e.g. discovers the manual for a complex machine; All that is needed is to assign the action to the manuals concept and the agent will learn the action once it encounters an instance of the manual. If the new action(s) triggered an opportunity (see Section 3.2), the agent can learn that discovering manuals is a good way to learn about the environment and may create opportunities.

3.2 Opportunities

Whenever an agent attempts to satisfy a goal, the agents beliefs are stored in a case along with the goal and the results of the attempt. Given enough such cases, the agent runs a rule induction algorithm on the cases with similar classification in order to learn what the opportunity might consist of. A possible output of such an algorithm is listed in table 3.3 on page 20.

Given only a limited set of observations, the agent is likely to create sub-optimal

Cases			
	Case 1	Case 11	Case 37
Goal	Buy eggs	Buy eggs	Buy eggs
Result	Success	Failure	Success
World.Time	12:41	19:33	17:07
World.Day	Monday	Saturday	Wednesday
World.Weather	Sunny	Rainy	Cloudy
Store.Location	Building 2	Building 2	Building 2
Store.Open	True	False	True
Store.HasSale	True	False	False
Agent.Location	Building 2	Building 2	Building 2
Agent.Money	100	2560	133
Agent.Mood	Good	Fantastic	Good

Table 3.2: Agent observations as cases.

rules (Zhang et al., 2006). In table 3.4 on the next page we list an example of how the proper rules could be. Another challenge comes from the fact that one goal may have several different associated opportunities. E.g. the goal 'buy eggs' may be satisfied by asking the neighbor if he has some. This way, the store is not involved at all and the opportunity could be indexed on the neighbor as well.

As the agent explores the environment, the observation component of the agent runs through all concepts and attributes, and signals all learned opportunities that index either the concept or the attribute. During observation the agent compares the newly observed data with its beliefs. This allows the agent to signal opportunities that are indexed in terms of increases or decreases of values.

In the example above, the agent would become aware that the goal 'buy eggs' is potentially satisfiable if it observes a change in the attribute `Store.Open` (for any given store it may observe), or an increase in `Agent.Money`.

This type of remembering enables opportunities that are discovered planning-time and execution-time (as the structures are shared between the agents simulation and observation components). It can discover endogenous and exogenous (both changes in the beliefs and in the environment). And it can index the opportunity in terms of tools, resources, time and place (all are represented as attributes).

Opportunity	
Goal	Buy eggs
Cases	1, 11, 37
Triggers	
Change	World.Time
Change	Store.Open
Increase	Agent.Money
Change	Agent.Mood
Conditions	
World.Time	> 12:41
World.Time	< 19:33
Store.Open	True
Agent.Money	> 100
Agent.Mood	Good

Table 3.3: Example of an opportunity learned form observations.

Actual	
Goal	Buy eggs
Episodes	∞
Triggers	
Change	Store.Open
Increase	Agent.Money
Conditions	
Store.Open	True
Agent.Money	> 5

Table 3.4: Actual rules for the opportunity.

Chapter 4

Experiments

Using a very rudimentary implementation of the conceptual framework, we were able to design three very simple experiments. The domain is the same for all three experiments, as well as the room layout (see Figure 4.1 on the following page). The agent starts with the pre-programmed knowledge of the environments as depicted in this figure. The primary goal of the agent is to pick up the coin. The planner en memory were omitted in this implementation. The agent is provided the correct strategy given the observations. The intention of these experiments are to test the opportunity recognition component of the conceptual framework.

The agent can observe all instances in the same location as itself, in this case in the same room. The doors belong to both rooms they are connected to. Furthermore the agent can pick up and drop any item, and it can only hold one item at a time. All doors can be locked and unlocked given the agent has the correct key, and all doors can be opened/closed as long as they are unlocked. Once a door is open, the agent may change its current location by entering the connected room.

4.1 Simple

In the simple experiment, everything went as expected. We engineered a strategy which should be comparable to what a partial-specification planner would create; The goal is to get to the coin. There are two methods: 1) Unlock the locked door and enter the room containing the coin 2) Go around. Since the agent does not have the key, the first method is suspended and indexed in terms of the concept 'Key'. The

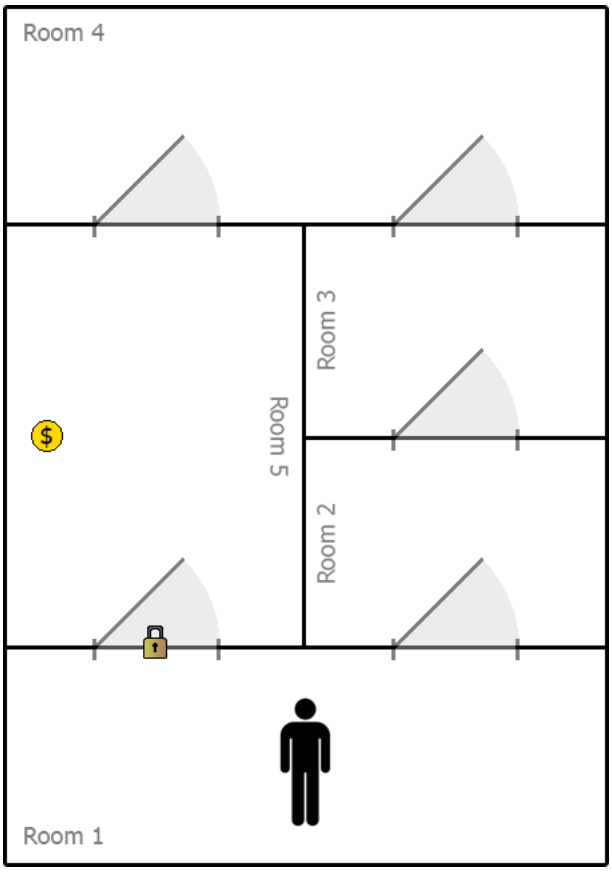

Figure 4.1: The initial programmed knowledge of the agent

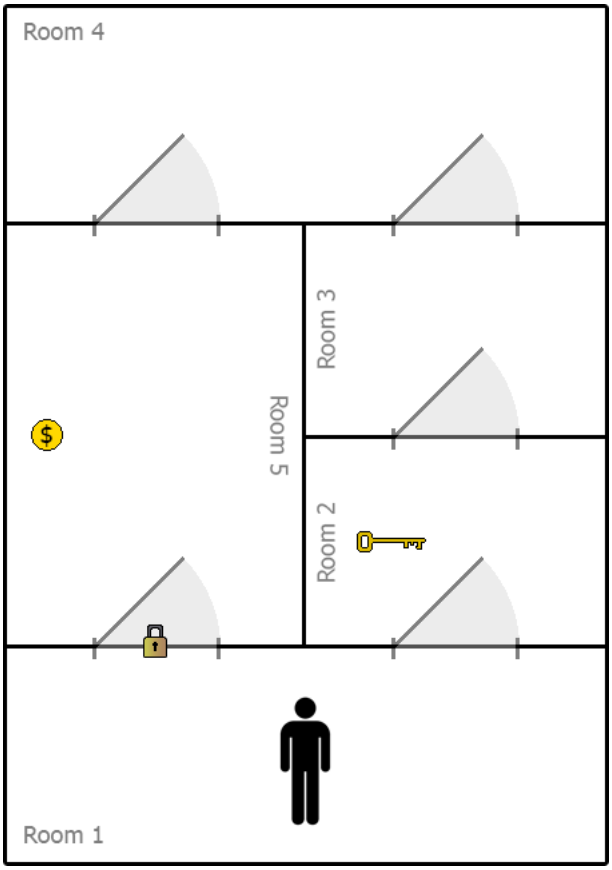

Figure 4.2: The actual environment of the simple experiment

agent should be able to resume the first method once it finds a key.

Upon entering the next room (Room 2) the agent discovers the key, and the first method may be resumed (see Figure 4.2). Since the first method has higher utility (in this case, shorter plan length), the agent chooses to continue with the first method and reaches the goal with 1 less action. Observations revealed underway did not trigger the agent to create a new strategy, as only the key was unexpected.

4.2 Noisy

The only difference between the noisy experiment and the simple one, is the addition of several randomly generated items (red). These red items can not be used to reduce the effort of the agent, and exist only to create noise in the environment (see Fig-

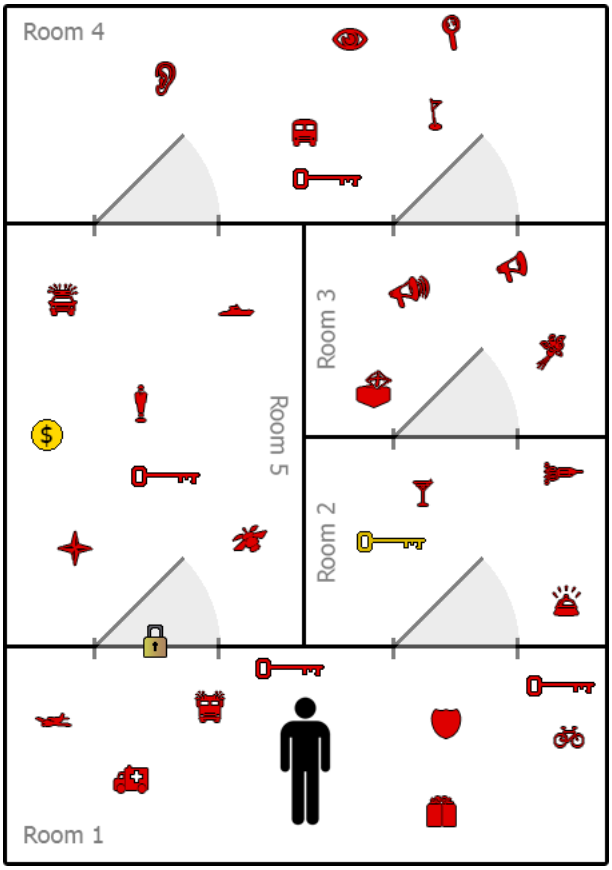

Figure 4.3: The actual noisy environment

ure 4.3). This requires a much larger initial effort when creating a strategy, however, the strategy in this example is exactly the same as in the simple experiment.

Once entering the next room (Room 2) the agent discovers the key, and a 3 other items. While the key allows the agent to resume the first method, as in the simple experiment, the agent performs some additional reasoning to see if any of the other items are usable[1]. Except for this, the scenario played exactly as in the simple experiment. The agent reached the goal with 1 less action.

[1]This scenario was already too large for classical planning. See 6.2 on page 31

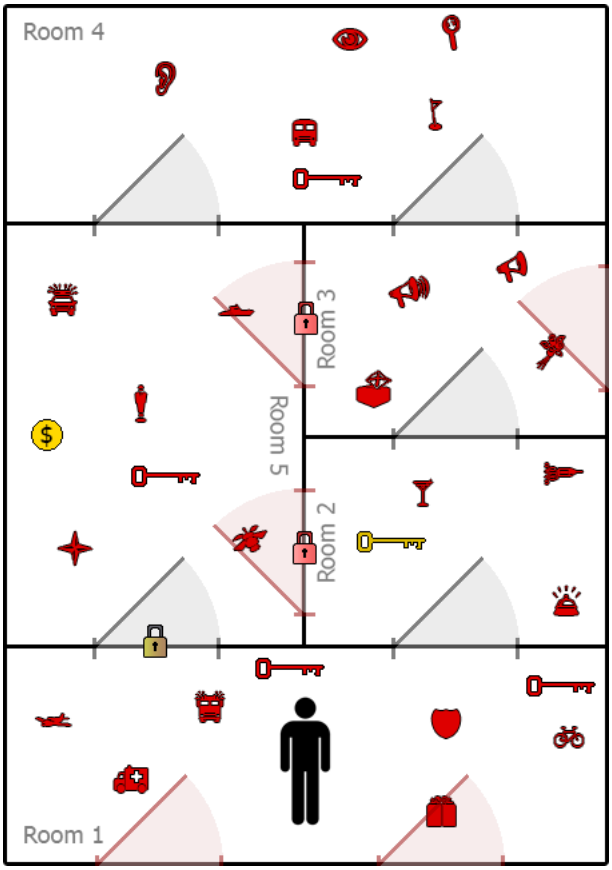

Figure 4.4: The tricky environment which requires more complicated strategy

4.3 Tricky

The tricky experiment illustrated the importance of filtering incoming observations (see section 6.3 on page 34 3rd paragraph). In this experiment the extra doors and keys (see Figure 4.4), while no combination except the yellow one works, were estimated to take a significant hit in the reasoning performance. Additionally, the strategy would have to increase significantly in size. As this experiment first becomes interesting when an actual planner is running, we did not implement this experiment and therefore we do not have any actual results. We note, however, that such an environment would impose a much higher workload on the agent until the agent has learned that the (red) items are not useful.

Chapter 5

Discussion

Below we go through the research goals from section 1.1 on page 2, and quickly discuss our results given each goal. The results from the experiments are questionable at best, as the scenarios were engineered to demonstrate specific features of the conceptual framework.

1. *Creating a framework for identifying opportunities in complex environments*

   As detailed under section 3.2 on page 18, there are several types of opportunities we can identify in complex environments. The rudimentary implementations indicate that our conceptual framework is possibly a sound approach. This is merely an indicator, however, as the implementation simplifies planning and memory to such an extent. While the results look Several more detailed experiments must be conducted before we can reach a conclusion.

2. *Creating a framework for recognizing opportunities in complex environments*

   The recognition mechanism works well, and other research which uses the same mechanism confirms these findings (Hammond et al., 1993; Hayes-Roth, 1993; Pryor and Collins, 1994). We extended it to allow the agent to remember several different types of conditions for reasoning.

3. *Finding a representation of opportunities for reasoning*

   Given the lack of prior information about the domain the agent is to operate in, we were required to find a flexible representation. Using Case-Based Reasoning (CBR) as a foundation allowed us to use previous research when generalizing

and finding similar cases. With some additional work, we believe rule induction/extraction to be useful tools together with a memory that stores observations in order to take advantage of opportunities.

4. *Specifying what an agent should learn in order to recognize future opportunities*

   Much of the work on this research goal has been based on reinforcement learning. We attempt to narrow down which features we store when observing opportunities related to specific goals. But we also need to keep remembering some odd features in case they become relevant in the future. In other words, we must avoid using a stabilising filter. The experiments did unfortunately not cover this aspect of opportunity recognition, and therefore a conclusion would be premature. This is probably one of the most important points under section 6.3 on page 34 (Further work).

5. *Specifying when an agent should create a new case in order to recognize future opportunities*

   The model we used in this thesis is just one of several possibilities. However, if we consider the circumstance where an opportunity may arise as an opportunity, using recursion we could potentially learn chains of circumstances that eventually lead to opportunities.

6. *Specifying how an agent should decide whether to pursue the recognized opportunity*

   This research goal, if slightly rephrased, has been the focus of much research. We preferred an approach using an indefinite horizon with discounted rewards. This approach has been formalized for (partially observable) Markov decision processes. While we can not expect to find an optimal policy for complex environments, we can analyse the potential future reward given the current strategy and horizon. Our thesis has not contributed anything new to this research goal.

All things considering, we believe we have a solid conceptual framework, but we do not have enough data to confidently back this claim. While much of our work is based on previously proven research, the combination itself is not guaranteed to work as we intended. Much less once the real-time constraints of the real world are considered.

5.1 Limitations

- *The agent does not properly understand causation*

  Consider the following scenario: The agent is in a room with a locked chest. No matter how many times the agent unlocks the chest and opens it in order to find the treasure inside, the agent will not learn to associate the chest with the treasure. It will, at best, be able to learn that the action of opening chests has a certain probability of revealing a treasure. This is because the agent does not consider chains of actions. The agent we propose can only see a relationship between the last performed action and the results of the state change following that action.

- *The agent does not properly understand complex attribute relationships*

  If an opportunity could be reliably predicted by the presence of the car owned by the person married to the agents boss, and the agent understood this relationship, once the agent observed the car it could trigger the indexed goal. However, in our framework there is no way to currently discover such relationships. More formally, while the agent may understand that $a \rightarrow b$ and $b \rightarrow c$ where c is an opportunity, it does not realize that $a \rightarrow c$. As a result, even though the opportunity c could be reliably predicted from a, the agent can only index the opportunity under the attribute b.

Chapter 6

Conclusions and further work

6.1 Conclusion

In summary; we set out to create a conceptual framework for an opportunistic agent that identifies, learns and recognized opportunities in complex environments. Classical planning and case-based reasoning are not tractable in such environments, and some simplification must be performed. With respect to planning, we are confident that partial-specification planning is the most promising candidate in the environments we considered. Case-based reasoning worked for short simulations, but as the scenarios grow bigger there is a need to filter and prune the case-base (see section 6.3 on page 34 3rd paragraph). Otherwise, we believe the conceptual framework shows promise, and that opportunity identification and recognition in complex environments is tractable. Unfortunately we lack results from proper experiments to confidently claim usefulness of our approach.

6.2 Additional work

Unfortunately, we spent much time on development that never bore any fruit. We underestimated the problem of planning in complex environments, and our results suffered significantly because of this. Below we quickly outline some of the failed attempts at producing conclusive and valuable results using our conceptual framework. All in all, we ended up with over 441 code files totalling 1,23MB which ended up yielding nothing of value.

6.2.1 Domain modelling software

Since we are interested in large and complex environments, it was natural to desire a tool that would allow fast modelling of such environments. We started on an implementation in Java which would allow modeling of concepts, actions and attributes, and save those to a file for running in the simulation software. While the software allowed definition of concepts, attributes and actions, as well as the underlying data models for the attributes, there was not enough time to write software that would load and simulate those domain models. As we realized this fact, we started development of a system that would generate large and complex environments, but this task proved to be just as time consuming considering the other challenges with planning. Factoring in the extra time needed to interpret the output of a generated environment, we realized this was also a dead end. The software has been released to the public domain, but it is doubtful that it will be of any practical value.

6.2.2 Simulation software

Testing the conceptual model was naturally of high priority. Getting all the different libraries working together was too big a challenge, especially with much time spent on the modelling software. Integration of myCBR 3 and JSHOP2 went as expected, but we soon realised that JSHOP2 would not be able to plan in the environments we were interested in. Since both those libraries were written in Java, we decided to do the same with out library. However, memory constraints on the Java VM ended up destroying any hope of simulating even large deterministic environments. Once we scaled down enough that the memory limit (12GB) was no longer an issue, the garbage collection (GC) took over as a critical problem. The application spent over 98% of its running time in the GC, which throws an exception (even as we tested on a high-end 12 core i7, the application made no progress). Finally, after changing the GC strategy, we still could not get results within a day of running. The Partially observable Markov decision process (POMPD) scenarios were now reduced to 5 concepts (4 attributes each), 15 instances and 5 actions, but we kept running out of memory[1].

[1] Although it is difficult to admit, at this point we began to suspect incompetent development from our side. Somewhere in the code we must have been giving the garbage collector a run for the money. It just seemed to unreal that such small environments could strain a modern desktop computer to such an extent. All unit tests were working as expected. There was, however, not enough time to investigate properly

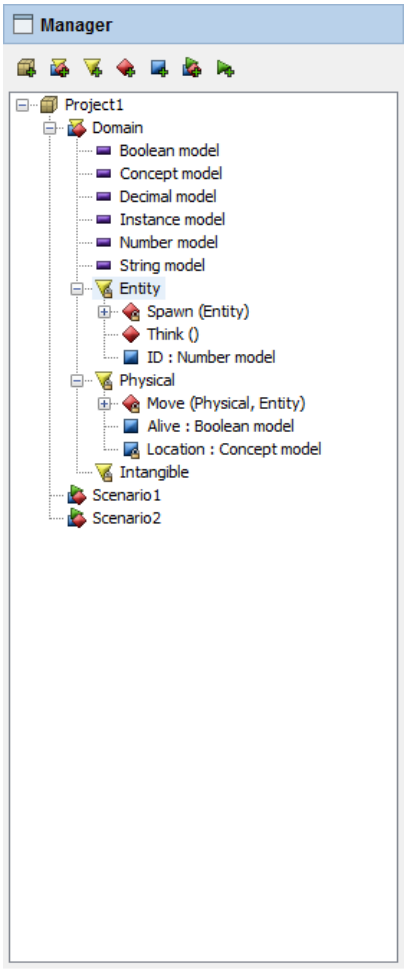

Figure 6.1: Screenshot from modeling software with a very simple domain

Classical planning was off the table. We started looking for partial-specification planners and approximation algorithms for solving POMPDs. Add on top of that libraries for machine learning (Weka), and libraries for rule induction, and we were stuck writing type converters and layers upon layers of wrappers in order to get the systems working together. We realised we had to simplify significantly.

6.2.3 Planner-less implementation

Finally, as time was truly running out, we implemented a rudimentary version of our conceptual framework. This very limited implementation started out with hardcoded knowledge about the opportunities in the environment (there was no point on build-

ing a case-base and learning without a planner which could use that knowledge in order to act). As a substitute to a planner (which was just a black box to the agent anyways), we created a planner that would generate the correct plan based on the current state. Since the environment was much smaller than intended, this was tractable by hand. We engineered scenarios specifically for testing out the opportunity recognition aspects of the conceptual model, and those were a success. Although, the validity of the results may be discussed.

6.3 Further work

Opportunity recognition and exploitation is limited by how precise the opportunity structures are. The agents ability to combine, generalize, remove and create these structures can be explored further. Triggers could have different weights, allowing the agent to suppress select opportunities when there already is a high workload. Conditions could be optimized to cover the broadest possible set of opportunities, while still giving few false-positives. Further work on rule induction or rule extraction on the observed cases might allow much more precise opportunity descriptions. Finally, one could experiment with several layers of conditions; one for unblocking the task so that the agent may resume it at will, and one layer signaling that the opportunity is now worth chasing (e.g. when an opportunity guarantees a high reward).

Active opportunity search was not explored in this thesis. Although a planner which handles stochastic environments could likely take advantage of the potential opportunities, this was not considered while designing the conceptual framework. This aspect of opportunism is somewhat related to internal opportunities. Internal (reasoning) opportunities were also not considered at any stage during the development. Internal opportunities could potentially skip expensive reasoning steps when evaluating possible actions (as happened even in the very limited experiments we performed). Actively looking for opportunities (chasing opportunities), and internal opportunities are potentially very interesting in complex environments.

One of the decisions we made early, was that we wanted the noise from the observations the agent made. This is because the agent does not know which features or concepts are relevant for carrying out its plan. However, analysing all the noise takes a lot of time and resources. One potential approach to reduce such noise is to filter the incoming observations before the agent begins reasoning. This requires some in-

teraction between the memory, the opportunities and the filter. This idea could be further expanded to include the memory itself. Instead of storing all observations, filter cases based on the expected amount of contributed information and even prune older observations that are no longer as relevant.

References

Francis, A. (1995). *Memory-Based Opportunistic Reasoning*. PhD thesis, College of Computing, Georgia Institute of Technology.

Hammond, K., Converse, T., and Marks, M. (1993). Opportunism and learning. *Machine Learning*, 10:279–309.

Hauskrecht, M. (2000). Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33—-94.

Hayes-Roth, B. (1993). Opportunistic control of action in intelligent agents. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 23(6):1575–1587.

Mancarella, P., Sadri, F., Terreni, G., and Toni, F. (2005). Planning partially for situated agents. In *Computational Logic in Multi-Agent Systems*, volume 3487 of *Lecture Notes in Computer Science*, pages 230–248.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404.

Pryor, L. (1996). Opportunity recognition in complex environments. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1147–1152.

Pryor, L. and Collins, G. (1994). Opportunities: A unifying framework for planning and execution. In *Proceedings of the Second Artificial Intelligence Planning Systems Conference*, pages 329–334.

Schank, R. C. and Leake, D. B. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40:353–385.

Simina, M. D. and Kolodner, J. L. (1995). Opportunistic reasoning: A design perspective. In *Proceedings of the Seventeenth Annual Cognitive Science Conference*, pages 78–83.

Weld, D. S. (1994). An introduction to least commitment planning. *Artificial Intelligence Magazine*.

Zhang, D., Yang, T., Wang, Z., and Fan, Y. (2006). A new approach to symbolic classification rule extraction based on svm. In *9th Pacific Rim International Conference on Artificial Intelligence*, volume 4099, pages 261–270.