ABSTRACT

CHRISTIAN, DAVID B. Strategic Deception in Agents. (Under the direction of Assistant Professor R. Michael Young).

Despite its negative ethical connotations, deception is a useful tool for human social interaction, and plays an important role in the process of creating the stories that pervade our popular culture. In this paper, we describe the deception planner, an implementation of a model of strategic deception. Strategic deception is deception performed in order to achieve or enable some higher goal, as opposed to deception that is performed for the sake of deceiving, or for an unstated purpose.

Given a model of a deceiver holding ulterior goals, a model of the goals and abilities of a target agent to be deceived, and a model of the relevant pieces of the world, the deception planner generates a set of statements about the current world state which may be either true or false. Those statements are communicated to the target agent, which updates its world state to reflect this new information. The target then performs planning to achieve its own goals, with no knowledge of the deceiver's goals. If the deception planner generates an successful set of statements, the target agent will create a plan that achieves the deceiver's ulterior goals despite no knowledge of those goals.

To find the set of statements that will generate this desired behavior from the target agent, the deception planner models the target agent's planning process. The deception planner searches for a plan that achieves the target agent's goals as well as the deceiver's ulterior goals. When such a plan is found, it is labeled the candidate plan, and the deceiver gives the target agent enough (dis)information so that, given the target agent's knowledge of the world, she can generate that plan. The candidate plan may depend on lies, chosen by the deception planner. The planner ensures that any lies told are not discovered before the target executes enough of the plan to achieve the ulterior goals.

Once a candidate plan has been found, the deception planner the finds and counters competing plans. Competing plans are plan that achieve the target's goals but not the deceiver's goals, but are of equivalent or better quality than the candidate plan according to some metric shared by the target and deceiving agent. Because they are of the same or better quality, the target agent may choose a competing plan instead of the candidate plan. A competing plan is countered by undermining through lying a belief that is necessary for that plan to be executable.

Although the deception planner fits within a body of work on agent deception, the goal of this algorithm is unique in its focus on causing a target agent to act in order to achieve a deceiver's goals, and in its utilization of a model of the goals and planning abilities of the target agent to that end.

Strategic Deception in Agents

by

David Christian

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2004

Approved By:

Dr. James Lester	Dr. Munindar Singh

Dr. R. Michael Young
Chair of Advisory Committee

I dedicate this work to my fiancee, Sarah Goetz, whose encouragement, support, and love mean the world to me.

Biography

David Christian was born in Minneapolis, Minnesota, on May 23, 1979. He spent his formative years in Edina, a suburb of the Twin Cities.

David went to a year of college at Knox College, in Galesburg, Illinois, before deciding that Macalester College in Saint Paul, MN, was a better fit for him, if for no other reason than because the nearest large city wasn't Peoria. He also had met his future wife while visiting the school. At Macalester, he majored in Computer Science and minored in Math. His favorite classes in college were Introductory Poetry, Modern Art, Sociology, Ethics, and Indian Philosophy. Number theory, Artificial Intelligence, and Parallel Programming weren't bad either. David graduated with a Bachelor of Arts in Computer Science and a minor in mathematics. Magna Cum Laude with Departmental Honors and joined the honor societies Phi Beta Kappa, and Pi Mu Epsilon, and three different honors societies, including Phi Beta Kappa. Most importantly, David met his future wife, Sarah Goetz.

David and Sarah moved to North Carolina immediately after graduation, where he embarked upon the journey that led to this document. In graduate school, David joined the Liquid Narrative Lab immediately, attracted by the shining lights and pretty pictures in their demo. There he has worked with Dr. Young on several projects, with a focus on narrative generation. He has been actively involved in the University Graduate Student Association, where he has helped develop the organization's new look through its web server, and is pushing to help make the organization more responsive to graduate students' needs. David will get married this summer.

Acknowledgements

Many people have helped to make this document possible. Thanks go firstly to my advisor, Dr. R. Michael Young, who funded several years of my research and has helped me figure out how to be a graduate student. Thanks also to the members of the Liquid Narrative Lab who listened to all of my crazy ideas past and present, and who make me laugh, including Mark R., Yuna, Arnav, Kevin, Alex, Brian, Justin, Joe, Tommy, and Jim. Also, thanks especially to the lab administrators, who are of a special breed, and who are sorely missed, R.J, Mark B., and Kevin.

Thanks to the University Graduate Student Association, an organization which has helped me discover parts of myself that have little to do with my Master's thesis, but I think will serve me just as well. Thanks especially to Dr. Rufty, Esther, Christina, Elana and the other people who served as representatives and executive board members to serve the graduate student population at NC State.

Thanks to my sisters, who have served as inspiration through their own successes. Now if they would just slow down some so I could catch up... And thanks to my mom, who always makes me feel like I could conquer the world whenever I ask her for advice.

Thanks finally to my fiancee Sarah, whose love and support made this thesis possible.

Contents

List of Figures

Chapter 1

Introduction

Deception is a powerful tool with a capacity for destruction. Despite its potential negative consequences, studies show that we lie almost every day, often to those who are closest to us [8]. Because deception triggers powerful reactions when discovered by humans, developing models and algorithms for computer deception poses a sometimes controversial yet fascinating research area. This thesis develops an algorithm to allow strategic, manipulative lying — lying which can cause a deceived agent or human to behave in a way which is to the benefit of the deceiver. We call the algorithm which determines the manipulative lies the *deception planner*.

Researchers have studied agent communication since the inception of the field of artificial intelligence (AI), but they generally make the simplifying assumption that the agents involved in communication are telling the truth. We will, like others before us, call this assumption the *sincerity assumption* [19]. It is based on a set of maxims for human conversation developed by Grice in 1975 [15]. The value of the sincerity assumption lies in occluding such conversational "anomalies" including sarcasm and deception, which require a more nuanced model of the workings of communication.

However, there are situations where deception practiced by a computer agent is of value. An agent acting as an office assistant might lie by telling a client that its boss is out of the office, or lie to delay a person for a surprise party [9]. Similarly, an agent might ask a criminal to wait while it checks a credit card number while in fact it is alerting the police. Castelfranchi notes that doctors and patients often deceive each other (or are "reticent"

with the truth) in order to steer a conversation away from particular topics [4]. If agents become consultants about matters of health or politics, such a conversational tactic may prove to be useful.

Story-telling, or narrative, is another compelling case for endowing computational agents with deceptive capabilities. We expect and accept that characters in a story lie to each other. Their lies help form the conflict and tension that make a story interesting. Likewise, it is an accepted narrative practice for an author to "deceive" the story's audience by occluding important information or using an unreliable narrator. Thus, narrative provides for both the deception of characters within the narrative and the deception of the narrative's audience.

Deception can be seen as more than just a component of narrative, however. Relating a fictional narrative involves the consensual communication of false information [17]. Like deception, developing a coherent narrative requires a model of the audience's relevant knowledge, as well as what he will be able to observe and deduce from what has been communicated to him. Because both the communicator and the audience know that the information passed is false, there is little, if any, moral ambiguity. Deception, then, may be a useful paradigm for considering the process of creating narratives.

We describe an algorithm, implemented as the Deception Planner, that develops a set of lies intended to cause a target agent or human (from now on denoted as simply the *target* or *target agent*) to perform actions in pursuit of its goals in a way that benefits the deceiver. Specifically, the target acts on false information, and then begins to act out a plan based on that information, and in the process of acting out that plan achieves the ulterior goals without knowing of the existence of these goals. The *deception planner* uses a modified heuristic search planner to create the lies necessary for this process.

The remainder of this thesis is divided into the following chapters: Chapter 2 describes relevant related work. Chapter 3 defines the scope of the problem solved and the properties of its desired solution. Chapter 4 describes the algorithm designed to solve this problem. Chapter 5 gives an example of the deception planner. Chapter 6 maps future work, and chapter 7 concludes the thesis.

Chapter 2

Related Work

The work related to deception planning can be divided into two broad categories: those projects directly related to deception by agents, and those related to anticipating the behavior of an opponent agent.

2.1 Deception by Agents

Researchers developing models of social interaction between agents have often asserted that agents do or should follow a Gricean model of communication [15, 6]. Grice claimed that people communicate cooperatively, following simple maxims to ensure that they are understood and that communication is efficient. The four maxims that Grice posits are the maxims of quality (the information must be accurate), quantity (the information must be no more or less than necessary), relation (the information must be related to the topic), and manner (it must be presented in an appropriate manner).

If, in a given situation, all participants are aware and are obeying these maxims, it is possible to more easily derive the meaning of a speaker's utterances. If a hearer cannot make assumptions about the accuracy and completeness of the utterance spoken, then the utterance loses its value as a source of real information. If the speaker is assumed to be giving only the necessary and relevant information, then a hearer can rely on the information given by the speaker, and even derive information beyond that specifically given in the utterances.

For example, if the hearer is given only a partial description of an event by a speaker, and if the speaker is following Grice's maxims, then the hearer may assume that the speaker wants the hearer to assume reasonable defaults for any missing information. It is the possibility of deriving extra information from utterances that has made Grice's maxims particularly valuable to the study of agent communication.

Despite its value, some researchers have allowed agents to depart from the maxims of quality and quantity. In the most direct case, such a departure implies deception — either through the deliberate falsification or the omission of information. These forays into deception also rely on the existence and presumption of cooperative communication — without some rules for honest communication, deception loses its effectiveness.

Castelfranchi has built a simplified deception model called GOLEM based on the blocks world initially defined in planning research [5]. The two agents in this world have disparate goals and capabilities. For example, Eve may want to build one large tower out of the blocks, while Adam may want to build two smaller ones. Eve can lie or tell the truth to Adam about her abilities, and depending on Adam's attitude towards Eve, Adam will either help her or not. Castelfranchi describes a set of attitudes that affect how agents deceive. For example, a lazy agent might lie in order to avoid doing work that she could do, while a pragmatic agent might lie only when it was necessary to win.

Castelfranchi's work is an important starting point for research exploring deception as opposed to research countering and preventing deception. His model is designed to achieve goals similar to the one proposed in this thesis. In both models, an agent deceives in order to cause another agent to act in the deceiver's favor. Like the deception planner, GOLEM is based inside a world in which the agents reason about action through planning systems. However, Castelfranchi's model of deception is limited in scope: GOLEM does not use the model of the goals and beliefs of the target agent in order to shape the lies told [5].

Carofiglio and de Rosis propose creating deception by taking advantage of a target's uncertainty. They have developed a system, named MOUTH-OF-TRUTH, which picks statements to assert on the basis of how effectively they can undermine the target's relevant beliefs, causing the target to doubt itself, or causing the target to deduce a false fact [7, 3]. MOUTH-OF-TRUTH uses a model of fact implicature to determine the likelihood of the target believing one fact if the other fact is believed. For example, if a person is convinced that it has just rained outside, there is a high likelihood that they will also believe that the ground is wet. These relationships are modeled using Bayesian belief-nets [14]. The

belief-nets model how strongly a target believes a particular fact. The model of implicature and the agent's current beliefs contain enough information to practice "safe" argumentation methods, which preserve a degree of separation between the deceiver and the deception. For example, the deceiver might claim uncertainty where there is certainty, a claim which is very difficult to disprove, or state afterwards that the false information came from an unreliable source.

The goals of Carofiglio *et. al.*'s MOUTH-OF-TRUTH system are complementary to those of the deception planner described in this thesis. While MOUTH-OF-TRUTH focuses on the safe communication of deceptive information to convince a target of the truth of a desired fact, the deception planner focuses on determining which facts a target must believe in order to act in a way that is beneficial to the deceiver. It is possible that the MOUTH-OF-TRUTH could be used as a method to cause a target to believe the lies needed for the deception planner.

Finally, contestant programs in the Loebner Prize Contest can be considered to be deceptive [10]. The Loebner Prize Contest is an implementation of the Turing Test [22]. The Turing Test was devised to answer the question 'If a computer can think, how do we tell?'. The Loebner Prize Contest sets up a judge to communicate through text with both computer programs and humans. The judge must rate how human the entity with which he is communicating seems. The computer program that seems the most human wins that year's contest. A computer that could fool a high enough percentage of humans all of the time could be considered to be intelligent.

Entries into the Loebner Prize Contest continually break Grice's maxims [21]. Loebner Prize Contest programs depart from the Gricean maxims by default: they are trying to assert their humanity, an assertion of low or no quality. Beyond this general violation of the Gricean maxims, Loebner Prize Contest entries break the maxims for pragmatic reasons: they do not have enough world knowledge to behave intelligently in all situations, so they cover by providing mostly irrelevant information, by evading questions, and even by acting angry or paranoid. Although the algorithms are forced to break the maxims because the have limited abilities, a study shows that breaking these maxims can actually improve the program's chances of being seen as human — especially by showing extreme emotion, even in violation of the manner maxim [21].

Although programs in the Loebner Prize Contest attempt to deceive humans, they do so to achieve a specific end determined by the contest they are involved in. It is possible

that the techniques used in the Loebner Prize Contest test may be complementary to the deception planner in the same way that the techniques used in MOUTH-OF-TRUTH are. Their evasion, omission, and deception techniques may be valuable to convince a target agent of the truth of a false fact, which is a task outside of the scope of the deception planner.

The above models focus mostly on the process of communicating a particular deception, without any concern about how the deception will effect the future behavior of the agent being deceived. Castelfranchi's GOLEM does concern itself with the effect of its deception, however it does not take any extra steps to ensure that the deceptions it tells will be successful. Our model uses extra information about the motivators and abilities of a target as well as the context of the deception to help choose which facts are important for the target agent to believe. In this way, we take the deception from the abstract to the purposeful: the deceiver deceives for its own self-interest, and calculates the set of mix of truths and falsehoods it tells in order to ensure that its interests are met.

2.2 Competitive and Opposing Agents

The deception planner focuses on guessing the target agent's potential moves, and using that information to its own ends. In this way, the deception planner is similar competitive systems, which attempt to either explicitly or implicitly discover and counter another agent's moves. Like communication, competition, especially games playing, has been a focus of AI since the field's early days.

Many algorithms for modeling competitive situations assume an opposing agent. The minimax algorithm computes the best move for the agent based on a search of potential moves by both the agent and the competition[20]. The possible combinations of moves is decreased by applying the assumption that each will only choose that action which is best for him.

Bowling *et al* describe an algorithm for Adversarial Planning [2]. In Adversarial Planning, an uncontrollable agent, called the environment agent, challenges the ability of a system-controlled, called the system agent to reach its goals. To counter the environment agent, a preferred action for each potential world state is developed. Each potential action is devised so that it will not trap the agent in a state where the environment agent could

keep the system agent from achieving its goals. By ensuring that the system agent acts to stay within a set of acceptable states that always allow a path to the goal, the system agent can usually ensure its achievement of its goals. In deception planning, the deceiver is similar to the system agent, in that it anticipates the target's moves and uses them to ensure the deceiver's goals are achieved.

Minimax and Adversarial Planning anticipate another agent's moves, as does the deception planner. The deception planner models the potential behavior of the target agent, which could be considered an opponent or environment agent. Unlike the deception planner, minimax and adversarial planning work within a constraint that states that the opponent is most likely acting in direct opposition to the system. They consider the opponent's upcoming actions, but only in so far as they negatively affect the system. In deception planning, there is an "opponent" whose goals are known, but not necessarily counter to the system's. The deceiving agent uses its knowledge to guide the target agent without entering into open competition.

Chapter 3

Problem and Solution Definition

In this chapter we give an informal overview of the problem solved by the deception planner, followed by a formal definition. The formal definition of the deception planner describes the scope of the problem the deception planner solves and the required characteristics of a solution to a deception planning problem.

3.1 Example and Overview

Consider the following example of strategic deception:

> Danielle wants to get to a pay-phone so she can call to get her car fixed. She is walking towards the gas station when a passerby, noticing her situation, tells her that the gas station phone is out of order. "But you can use my phone" says the serial killer, smiling widely. "My house is right over there."

The above example describes a typical situation handled by the deception planner: one agent, the deceiving agent, is considered to be a relative authority about current situation in relation to another agent, called the target agent. In the above example, it is assumed that the serial killer knows the real status of the phone at the gas station, and also about the phone in his house, while Danielle merely guesses that the gas station phone works, and knows nothing about the killer's house. The target agent is trying to achieve its goals, called *planning goals*. The planning goal in this case is the calling of a tow-truck, or perhaps getting to some further destination which requires that the car be fixed.

The deceiver wants the target to achieve another set of goals, called *ulterior goals*. These goals are part of a larger, implied plan held by the deceiving agent. The nature of this larger plan is outside of the scope of this work, but in the above case we can assume that the serial killer wishes to murder Danielle out of sight. The ulterior goal is the state that the target agent must achieve in order to make this plan possible — in the above example, it is simply to get Danielle into the house. The deceiver does not wish to ask the target to explicitly achieve the ulterior goal or goals, because the target agent may not wish to help the deceiver. Instead of asking the target to perform acts to achieve the ulterior goal as a favor or for no reason, the deceiver gives the target agent planning advice, in the form of new information about the current state of the world. Some of these pieces of information may be lies, however, there is no requirement that all of the statements told by the deceiver be false. The information passed from the deceiver to the target is the total output of the deception planner.

The new information given to the target is integrated into the target's understanding of the world. In this work, it is assumed that the deceiver has already determined which facts he can convince the target agent about, and only tells lies about facts that fit in that category. Determining whether a lie is believable or not is outside of the scope of this work, but a process that makes this distinction is assumed to have been previously applied.

After integrating this new information, the target generates a new plan of action to achieve her planning goals. If the deceiver has delivered a successful set of information, the target's plan will achieve the ulterior goals along the way to achieving the planning goals. However, because some of this new information that the target agent incorporated into the new plan may be false, the plan may not actually achieve the planning goals when executed. Imagine, for example, that the serial killer does not really have a phone in his house - if the target does not find this out until after entering the house, the deceiver's ulterior goals are still achieved, but the target's plan is guaranteed to fail.

The deceiver does not care whether the target is actually able to achieve her planning goals. However, the deceiver does care that the target believes that her goals are achievable up to the point where the ulterior goals are achieved. This restraint limits the type of lying that the deceiver can get away with. In this example, the serial killer couldn't tell Danielle any lie that would cause her plan to fail, or that would allow her to see that her plan would eventually fail, before she got into the house. The target's ability to observe facts is modelled in the deception planner.

Finally, in order to ensure that the target agent will use the plan provided, the deceiver should ensure that the target does not have a better alternate solution available, where better is measured by a metric that is specific to the target agent. In the above example, any plan that does not involve relying on a stranger might seem better to Danielle. Thus it is imperative that the deceiver force Danielle to rely on him through making her believe that any plan that does not do so will not succeed. To ensure that the provided plan is the best plan available, the deceiver must consider the alternate solutions to the target's planning goals. These alternate solutions are called *competing plans*, while the plan that the deceiver hopes the target will assume is called a *candidate plan*. The deceiver's preferred plan for the target is called a candidate plan in recognition of the fact that it is one possible solution to a target's problem, and that more lies may have to be told in order to make it the best solution to the problem.

3.2 Formal Model of Standard Planning Input and Output

To complement the informal overview of the deception planner given above, we give a formal model of the inputs and outputs of the deception planner in the next section. Because the deception planner relies heavily on planning, in this section a formal definition of the terms used in planning is given as an aid.

Planning has been formalized into logical representations on multiple occasions. The following definitions of plan components are derived from those earlier formalizations [1, 18, 16]. However, the planning functionality described below is restricted, and not include quantifiers or existence operators, among other modern planning features. The functionality is complete enough for the planning used by the deception planner.

A planning problem can be split into a planning *domain*, and a planning *instance*. A planning domain consists of plan predicates and plan operators. A plan predicate p is a first-order predicate $p(a_1...a_k)$, with a fixed number of parameters, k. Let a substitution $s = \{a_1 = o_1, ...a_k = o_k\}$ be a pairing of each of k predicate parameters with an *object*. Consider a parameter a_k be *bound* in a substitution if the parameter is listed in the substitution. And a_k be *grounded* in a predicate when a substitution that binds a_k has been applied to that predicate. Let a fact f be a predicate where all of its parameters are grounded.

An *operator* consists of sets of parameters, preconditions, add effects, and delete

effects. Preconditions, add effects, and delete effects are sets of predicates. The utility of precondition, add effect, and delete effect sets is apparent only after a substitution has been applied on all these sets in order to transform the predicates into facts, creating an *action*. The operator parameter set is the union of the parameters used in the preconditions and effects, and lists all the variables which must exist in a substitution to convert an operator to an action.

Let s be a substitution that substitutes for all variables in an operator o's parameter set. The application of that substitution to the precondition and effect sets of o creates an *action*. The action preconditions and effects are sets of facts. The action preconditions describe what facts must be true before the action can be executed. Facts in the add effect set are made true, or *achieved* when the action is executed, whereas those actions in the delete effects are made false after the executed is applied. Let a *state* be the complete set of facts which are true about the world at some unnamed time. Let the application of an action a on state S be written as $a(S)$. Given a state S before an action a is applied, the state S' after the action is applied is $S' = (S \cup A) - D$, where A is the add effects of a and D is the delete effects of a.

The planning *instance* contains the set of objects, O, that are available in the current planning problem, an initial state, I, and a goal state, G. The initial state and goal states are states such that the facts are based only on predicates in the planning domain, and the substitutions applied to those predicate only use objects available in O. Objects, predicate parameters, and operator parameters have types associated with them. Objects may only be substituted for predicate parameters of the same type.

Given both the set of objects from the planning instance and the operators from the planning domain, the complete set of actions that are possible in this planning problem can be derived by performing all possible substitutions on all operators.

A *complete plan* is defined as a set of ordered actions $(a_1, a_2, a_3, ..., a_n)$. When these actions are applied in order $a_n(a_{n-1}(...a_3(a_2(a_1(I)))...))$, the following conditions hold. Before the application of each action a_i, a_i's preconditions are true in the current state, and after the application of the final action, the goal state G is true.

3.3 A Formal Model of Deception Planning

A standard planner is given a domain and an instance of that domain, and tries to create a plan that achieves the goals listed in the instance of that domain. In contrast, the deception planner is a meta-planner, or a pre-planner. Its output is not a set of actions, but rather a set of inputs to the target agent's planner. We have informally defined the relationship between the deception planner and the target agent in Section 3.1. We now define the relationship formally.

3.3.1 The Target Agent Model

The deception planner DP takes as input a model of the target agent t. The makeup of this agent model help determine the constraints, complexity, and applicability of the deception planner. The deception planner's model of the target agent consists of enough information to infer the inputs to the target agent's planning process. It is this standard planning process that the deceiver subverts through deception.

The target agent t's planning input consists of a goal set G, an operator set OP, and an initial state that is divided into two sets of facts, a knowledge set K, and a belief set B. The target agent also has a planning metric PM which is used to rank whether one plan is better than another.

The target's understanding of the initial state is incomplete, or at least uncertain — that is one of the underlying requirements for the applicability of the deception planner. If the target knew everything about the current world state, then there would be no way to deceive the target into acting in the deceiver's best interest. The belief set contains those facts which the deceiver is an authority on, or is able to convince the target about. The target agent believes facts in B only until it is told otherwise, whereas the knowledge set represents all knowledge that the target is certain about and will not believe lies about. Note that this distinction between knowledge and belief is pragmatic, not philosophical.

Formally, let K be the target's knowledge set, and B be the target's belief set. Let I be the current world's actual initial state. Then $K \cap B = \emptyset$, and $K \cup B \subseteq I$. If a fact $f \in K$, then t will not believe lies about that fact. If a fact $f \in B$, then t believes that the fact is true, but will believe lies about that fact. If $(\{f, \neg f\} \cap K = \emptyset) \wedge (\{f, \neg f\} \cap B = \emptyset)$, then the target does not have any knowledge about the truth or falseness of f, and will

believe lies about f.

The target agent's plan metric PM allows the target to compare plans in a meaningful way by calculating whether one plan is better than another. For example, assume the target agent's metric measures only the length of a plan. If the deceiving agent gives the target information that allows it to generate a plan with twenty steps, and achieves the ulterior goals, but the target agent can also generate a plan with three steps, the target may decide to ignore the new information given by the deceiver and execute the plan taking three steps. To avoid this result, the deceiver must counter shorter plans that do not achieve the ulterior goals. Alternatively, if the deceiver can offer a plan that takes three steps, the deceiver does not need to worry about the target agent determining that a plan of length twenty is more desirable. The planning metric used by the agents in this thesis is always the length of a plan.

The goal, operator, knowledge and belief sets, and planning metric contain enough input for the standard planning process. However, because the deception planner models process of lying, we also model the process through which the target agent can discover lies. This process is modeled through two additional components of the target agent. The target's observation rules, OR, define how the target agent observes facts about the world. The control cycle, C, defines how the target proceeds in acting out a plan, with particular focus on when the target agent abandons a plan that will not work.

Observation rules describe those facts that the target agent may observe while acting out its plan. They could be used to catch the deceiver in a lie by observing that some statement told by the deceiver is not true. The deception planner uses them to ensure that the target does not catch the deceiver in a lie in that fashion.

Observation rules contain a *trigger*, which contains the conditions under which the observation rule fires, and an *effect*, which contains those conditions which will be observed when the rule is fired.

Observation rules rely on the concept of groundedness. A partially grounded predicate is a predicate in which only some of the substitutions which transform a predicate into a fact have been made. For example, a partially grounded predicate may have an object substituted for its first parameter, while the rest of its parameters are still unbound.

Let an *observation rule* o in OR be a tuple $< TR_o, EF_o >$. Let TR_o be the rule's trigger, and EF_o be the rule's effect. Let TR_o and EF_o be sets of partially grounded predicates. If given some substitution s, all conditions in TR_o/s are true given the current

world state, then all conditions in EF_o/s are observed. If, after applying s to EF_o, there is still some unbound variable in EF_o, then all objects of the correct type may be substituted in turn, and all the resulting facts are observed.

At every step in the execution of the target's plan, the trigger rule is matched against all possible substitutions s that might make all the facts resulting from the substitution true. For example, it is possible to have an observation rule o_1 for an agent *agent* that had $TR(o_1) = at(agent, ?loc)$. If $at(agent, school)$ were true, perhaps after the agent moved from some other location to school, then o_1 would be triggered with the substitution $s = \{?loc = school\}$.

Once a trigger is matched with a substitution s, then that substitution is applied to the effect of the observation rules. If there are any partially grounded predicate in the resulting effect, all possible objects, given type constraints, are substituted for the still ungrounded parameters. The resulting set of facts are observed, meaning that if they are true, the agent knows they are true, and if they are false, the agent knows that they are false.

Continuing the example, let $EF(o_1)$ be $at(?obj, ?loc)$. After applying $s = \{?loc = school\}$, the effect would be $at(?obj, school)$. The predicate $at(?obj, school)$ is only partially grounded. All objects would be substituted in turn into the ?obj variable, meaning that the target would know the truth value of $at(agent, school)$, $at(car, school)$, and so on.

The observation rules, applied in combination with the agent's *control cycle*, ensure that the deceiver cannot lie about a fact if the target would discover its truth value before the target has achieved the ulterior goals. The control cycle C describes how the target agent executes its plan. Once the target has developed a plan, the target will act out its plan, step by step, until all steps have been executed, or until the target agent observes a lie, or until a plan action fails because its preconditions do not hold, due to a lie. The control cycle can be written as the pseudo code shown in Figure 3.1.

An implementation of the described pseudo code is never run by the deception planner. The pseudo code given is the procedure that the target agent is assumed undertake after the deceiver gives the target agent its updated world state information. It is due to the assumption that the target agent will act in this ordered way, and not in some random manner, that the deception planner can discern what information will affect the target agent's behavior in the desired fashion.

```
L = New facts told by deceiving agent
B = B − (B ∩ ¬ L) //remove conflicting beliefs
B = B ∪ L //add new beliefs
Create Plan using G, OP, PM, and (B ∪ K) as inputs
A = plan actions
I = actual world state //(NOT B ∪ K)
RW[1] = I //real world state that updates as actions are performed
AM[1] = B ∪ K //agent's model of world state that updates
                        //as actions are performed
RL[1] = L //relevant facts told by deceiving agent

for i = 1 to n:
   for each observation rule o:
      for every possible substitution s for TR(o)
         if (TR(o)/s ⊆ RW[i]):
            pge = EF(o)/s //pge = partially grounded effects
            for every possible substitution s2 for pge:
               for every fact f ∈ pge/s2
                  if(f ∈ RW[i])
                     AM[i] = (AM[i] ∪ f) − ¬ f
                  else
                     AM[i] = (AM[i] ∪ ¬ f) − f

      for every fact f in RL[i]:
         if (¬ f ∈ AM[i])
                 STOP.

      action = A[i].

      for every precondition f of A
         if(f ∉ AM[i])
            STOP.

      perform_action(A[i]).
      RW[i+1] = (RW[i] ∪ add_effects(action)) - delete_effects(action))
      AM[i+1] = (AM[i] ∪ add_effects(action)) - delete_effects(action))
      RL[i+1] = (RL[i] − ¬ add_effects(action)) -
delete_effects(action))
```

Figure 3.1: Pseudocode for the target agent's behavior (control cycle) after the deceiver has given it the updated information

There are two points to note about the control cycle pseudo code. First, the planning algorithm used by the target agent is not specified. The target agent may use any algorithm, and consider plans in any order, so long as when the planner finds a plan that is better than the current plan according to the planning metric, it adopts that plan. Second, the target agent stops running a plan as soon as it believes that the preconditions of an action it is performing are not true, or it believes that the statement of the deceiving agent is not true.

For the purposes of this algorithm, we assume that the target has access to a perfect planner, and will always pick a plan of optimal length. This assumption puts the deception planner at a disadvantage: it has access only to a real planner, which must choose between optimalty and speed. The deception planner chooses to sacrifice optimality for speed and an ability to improve the plan as time progresses. In some cases, the deception planner may fail to find a shorter plan that the target agent finds, and thus fail to successfully perform deception planning. We believe this potential failure is aceptable, as it mirrors a potential flaw in a human's plan for deception.

3.4 Problem and Solution Definition

Below is a concise definition of the problem solved by the deception planner, given the above definitions.

The Deception Planner is given a set of ulterior goals, the current world state, and a model of the target agent as described in Section 3.3. Given these inputs, the deception planner outputs a set of facts, F, and negations of facts, NF. These sets have the properties that $\forall_{f \in F}(f \notin K) \wedge (f \notin B)$, and $\forall_{f \in NF} f \in B$. Intuitively, the deception planner tells the target agent information that the target agent does not already know (is not in the target's knowledge of belief sets), or actually denies something the target currently believes.

Let the target's updated initial state UI be $(K \cup B \cup F) - NF$. Then we define the *target's planning problem* as the planning problem created by attempting to achieve the target's planning goals G given the updated initial state UI and the target's available actions A.

Given this updated initial state it is possible to create a *candidate plan* CP that solves the target's planning problem, while fitting three extra restrictions.

First, given the ordered execution of the steps in the candidate plan, at some point after the execution of step s in CP, all the facts in the ulterior goal set will be true.

Second, if there is a fact f such that $f \in F : f \notin I$ or a fact in $f \in NF : f \in I$, then f is a lie. As such, the following extra conditions much hold: f must not be used as a precondition for an action in CP before step s, and any observation rules which would trigger the observation of f must not be triggered before step s.

Finally, a successful solution to the deception planning problem will generate a planning problem in which the only optimal solution is the candidate plan as determined by the target's plan metric PM. Any such solution to the above planning problem that does not achieve the ulterior goals or allows the early observation of a lie is a *competing plan.*

Intuitively, the deceiving agent is searching for a set of lies to tell that will cause the target to create a plan that will achieve the ulterior goals. The deceiving agent does not care whether the target agent's planning goals are achieved, but must consider them when crafting the lies that it tells. The target cannot be expected to take any action simply because the action fulfills an ulterior goal — the target agent does not know about those goals. Instead, any action that achieves an ulterior goal must serve some other purpose by helping to achieve the planning goals.

To this end, the deception planner searches for a candidate plan. A candidate plan is a plan that could be executed by the target agent that achieves all of the deceiving agent's ulterior goals in the process of achieving the planning goals. The ulterior goals do not have to be true at the end of the candidate plan. Instead, consider the candidate plan to be fully ordered. To be a candidate plan, all of the ulterior goals must be asserted after the execution of some action a, and no lies must be observed until after the execution of that same action. A candidate plan must also be crafted to ensure that, if the plan is acted out by the target, the target agent will not halt the plan due to the discovery of a lie before the ulterior goals are achieved. A lie about a fact f is discovered if f it is observed: that is, if an observation rule is triggered before or during the execution of the candidate plan, and that observation rule has as an effect the fact f.

Finally, the candidate plan must not add any extraneous actions which do not help achieve the target's planning goals. Such actions would not make sense for the target agent to execute, and so are not acceptable in a candidate plan.

The goal of this process is to communicate knowledge to the target agent that will

cause the target agent to create a plan to achieve the ulterior goals. If the target agent does not know anything of the surrounding environment, the simplest way to do that is to communicate only those facts (or lies) needed to develop the candidate plan. If the target agent has initial knowledge (expressed as facts in $B(a)$ or $K(a)$), or if the candidate plan contains within it a shorter plan that achieves the planning goals but not the ulterior goals, the target agent may learn the facts stated by the deceiving agent, go through the planning process, and develop a different plan, a *competing plan*.

There may be an almost limitless number of competing plans, depending on the knowledge held by the target agent. As the knowledge of the target agent increases, deception becomes harder and harder, as there may be more competing plans to discover and counter. In the limit case, there is no possible successful deception planner, because the target agent has enough knowledge to keep the deceiver from ever successfully leading her astray.

In order to find as many competing plans as possible, the algorithm continually tries to find competing plans that are better by some metric than the candidate plan and do not achieve the ulterior goals. As it finds a better plan, it searches for a lie that would not be discovered and could void the competing plan. This lie is then added into the set of facts to communicate to the target agent, and new competing plans are sought with this new lie in place.

A successful deception planner is one that crafts a set of statements that allow for a candidate plan to be discovered by the target planner while disallowing the discovery of competing plans that are shorter or equal in length.

Chapter 4

The Deception Planner Algorithm

In this chapter we describe the deception planner algorithm in detail. The deception planner algorithm solves the problem set out in Chapter 3, by finding a candidate plan and countering its competing plans. We first describe the standard planning algorithm that serves as the backbone for the deception planner. We then describe the modifications to this planning algorithm that are necessary to search for candidate plans, followed by the process used to search for competing plans. Finally, we describe how the search for candidate and competing plans is translated into a set of statements which are communicated to the target agent.

4.1 The LPG Planner

The deception planner used in this work uses a modified version of the LPG (Local Search for Planning Graphs) planner [12, 13]. The LPG algorithm is a complete implementation of the latest standards in planning research, as listed in the Planning Domain Definition Language 2.1 (PDDL)[11]. Its support for numerical and time based plans, resource maximizing and user-defined plan-quality metrics make it state-of-the-art, while its ability for plan repair make it relatively unique in the planning world.

The LPG planner was chosen as a starting point for the deception planner because it is a local search planner, has a relatively informed heuristic and it performs plan repair

(it also happens to be freely available). Its other features are unused by the current version of the deception planner. LPG's plan search algorithm is described in the following section, followed by LPG's plan repair algorithm.

4.1.1 LPG Plan Search Overview

The LPG planner, like many other planners, treats planning as a exploration of a search space, where the nodes in the search space are partial or complete plans [18]. A partial plan may have zero or more actions in it, and must at least one flaw in it, meaning that the actions in the plan cannot be executed in order and achieve the goal state, given that they are executed in the context of the initial state. The most common plan flaw is an *open preconditions*. Recall that a precondition is a fact that must be true when an action is executed for that action to execute successfully. An open precondition p in plan P is a precondition for some action in P such that p is not asserted by the actions before p or the initial state. Conditions in the goal state can be considered preconditions of a special "goal action". Most planners treat planning as a search problem, and are distinguished by how they search that space.

Most search algorithms depend on a heuristic to help them determine which node in the search space to search first. The search algorithms for planners in the late 80s and early 90s such as UCPOP were *systematic*, meaning that they were guaranteed to find a solution eventually, given that such a solution exists.

Unfortunately, the heuristic used by such planners to find the solution quickly were generally not very informed. A heuristic is informed if its estimate of the value of a node (in this case, the likelihood that the partial plan being examined will lead to a good solution to the planning problem) is accurate. For example, the default UCPOP heuristic counts the number of steps currently in a plan plus the number of flaws in the plan (usually open preconditions), and considers that total to be the partial plan's heuristic value. This value can be thought of as an estimate of the number of steps that will be in the final plan that could be created from this partial plan. The number of steps currently in the plan accounts for the current length of the plan, and the number of open preconditions gives an estimate of the number of steps that will need to be added to create a complete solution to the planning problem.

However, the UCPOP heuristic grossly underestimates the number of steps some

preconditions will take to achieve. In general, not all open preconditions are alike. Nothing about a precondition indicates how many actions it will take to achieve that precondition. Indeed, the number of actions needed to achieve a precondition is highly context dependent. For example, if a precondition states that you must be at the top of the Eiffel Tower, that precondition will be much easier to achieve if you are already in Paris. Although the default UCPOP heuristic is not very informed, UCPOP and other planners used it because of its low calculation cost, which allowed for the fast examination of many search nodes. Designers were also encouraged to design their own heuristics which could take into account domain specific information about the difficulty of a precondition.

In 1996, McDermot developed a heuristic that traded the ability to quickly search a large number of nodes for the ability to accurately estimate the quality of a partial plan, and thus more quickly arrive at a soltuion. His planning system, named UNPOP, uses *relaxed planning* in order to calculate the estimated number of actions needed to complete a partial plan. Relaxed planning achieves the current open preconditions while ignoring the delete lists of all actions added to the plan. The number of actions used in this relaxed plan is used as the cost of completing the current partial plan. The calculation is guaranteed to underestimate the number of steps needed to complete the plan. McDermot's more accurate heuristic costs more to calculate, but actually leads a solution more quickly. His planners and others with more informed heuristics are called Heuristic Search Planners (HSP).

HSPs not only use a more advanced heuristic, they also generally are not systematic. A systematic planner must explore each node currently visible in a search space within a finite amount of time. This is only possible if a queue of all the partial plans seen so far is kept. These plans are ranked along with all plans expanded by generating children of the current node, and the best plan from the queue is expanded next.

In a HSP, no such queue of older partial plans is kept. The heuristic is often good enough to make such a queue unnecessary. Instead, heuristic planners perform local search. Local search algorithms, such as hill-climbing, consider only the neighbors of the current search node. The neighbors are created by considering a set of possible alterations to the current plan, based on solving a flaw with the current plan. The behavior of HSPs upon reaching a node with no better neighbors is varied. Some algorithms allow the removal a previous addition to the plan, if that change makes a positive difference in the evaluation of the algorithm.

The LPG planning algorithm uses local search as described above. The basic

```
function search(Initial Plan)
{
        current plan = Initial Plan

        while current plan is not finished:
             get neighbors(current plan)
           evaluate neighbors(current plan)
      current plan = best neighbor

        final plan = current plan
}
```

Figure 4.1: Pseudocode for the LPG (Local Planning Graphs) planner

search algorithm is displayed in Figure 4.1. An empty plan with only the preconditions from the "goal state action" is passed in to the search function. The LPG search algorithm can be divided into two parts: creating the neighborhood of plans for the current plan, ranking the neighborhood plans. The next two sections describe these parts.

4.1.2 Neighbor Generation

LPG's generation of neighbors for a partial plan is straightforward. LPG picks an open precondition in the current plan. By default, LPG picks an open precondition from the action was most recently added. Picking open preconditions that are recently added leads to a search of the plan-space where one particular goal condition is solved entirely before the next is considered.

For each action a that can be added to the plan P in order to achieve this open precondition, a copy of the current plan with the additional action a is added to the partial plan's neighborhood. If a asserts some fact which is deleted by some other action, or deletes another action's assertion, then then multiple neighbors are created for a, one before and after each point in P where a's effects interact with other actions in P.

If p is a precondition to the action a', then a neighbor plan is created with a' removed. This removal counts as a resolution to the precondition, since the precondition is no longer a relevant part of the plan. Such a removal is not possible if the precondition is a goal condition. An example of neighborhood generation is shown in Figure 4.2.

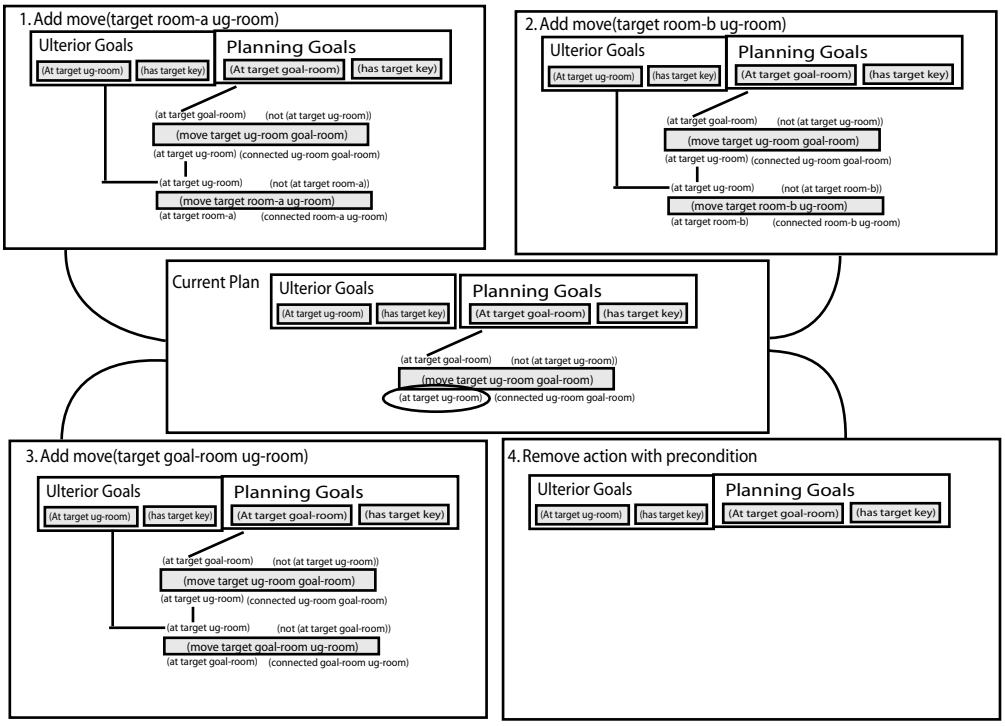


Figure 4.2: Neighborhood generation example. A precondition is chosen from the current partial plan and solved in 4 ways. Three ways involve adding a new action to assert the precondition. The fourth way removes the action which the selected precondition.

4.1.3 Neighbor Evaluation

LPG evaluates its neighbors using a complex heuristic that, given a partial plan, attempts to estimate accurately the cost of the complete plan that would be derived from that partial plan. The details of the heuristic are complex, and somewhat tangential to the focus of this paper. For completeness, the detailed heuristic used in LPG is included in Appendix A. Broadly, the heuristic for altering a plan P by adding an action a is as follows:

$$\begin{aligned} H(P,a) &= (1 + \lambda_m^a * CostToFixMutexes(P,a) \\ &\quad +\lambda_p^a * CostToFixOpen(P,a)) \end{aligned} \tag{4.1}$$

The LPG heuristic is based on the open preconditions in plan P as well as the mutexes in P. A *mutex* is created when the action a_{mid} negates a fact as its effect that was asserted earlier in the plan by action a_{before}, and is required to be true for a precondition p of an action a_{after} later in the plan. The addition of action a_{mid} in between these two actions requires that p be reasserted after a_{mid}. The cost of reasserting this precondition is what is calculated in CostToFixMutexes. The weights λ_m^a and λ_p^a are dynamically calculated stochastic weights that take into account the empirical difficulty of solving a particular precondition within a particular planning problem. Their exact calculation is beyond the scope of this thesis.

CostToFixOpen(P,a) estimates the cost to assert each fact that is an open precondition in the plan $P + a$. Asserting a fact f requires inserting an action a that has f as an effect. This action may have unfulfilled preconditions, and so on. Ideally, an estimate of the cost of asserting a fact f would include the cost of satisfying all of the preconditions of a as well. The actions used to satisfy those preconditions may have open preconditions as well, leading to a chain of actions that eventually should rely only on the initial state. Finding the size of this chain of actions is as difficult as solving the planning problem in its entirety, so LPG uses simplifying assumptions to create a polynomial-order heuristic.

LPG estimates the number of actions in this chain through performing relaxed planning to complete the current partial plan. In relaxed planning, the delete lists of added actions are ignored. Because the deletion of facts is ignored, once an action is added to solve a precondition, its effects can always be used to solve preconditions using the same fact elsewhere in the plan. The total number of actions that must be added to ensure that all the necessary preconditions are asserted using relaxed planning is used as the estimate of the number of actions needed when using real planning.

LPG's heuristic is used to find a complete plan quickly. The heuristic is modified for the deception planner to also find a plan that reaches the more stringent criteria set for plans that achieve the deceiver's ulterior goals as well as the target's planning goals.

4.1.4 Plan Repair

Once a complete plan is created, LPG has the ability to improve upon the plan, if possible, through plan repair. This ability is helpful to the working of the deception planner, which must not only find a plan which achieves the deceiver's goals, but also ensure that the target will not find a better plan on his own.

```
function LPGPlanner(initial_state,goal_state,actions):
        initial plan = empty plan

        do forever:
                current plan = initial plan

                while current plan is not finished:
                                get neighbors(current plan)
                                evaluate neighbors(current plan)
                                current plan = best neighbor

                final plan = current plan

                if(final plan is good enough):
                               return final plan
                else:
                               initial plan = add_errors(final plan)

                repeat
```

Figure 4.3: pseudocode for LPG planner

LPG is capable of plan repair because its search algorithm is local: given a starting partial plan P, it can consider all possible modifications to the plan, including removing actions already in the plan. When improving a plan, the LPG algorithm executes code similar to that in Figure 4.3.

The function `add_errors(final plan)` takes a complete plan, and removes several actions from the plan in order to create open preconditions that need to be fixed. The actions that are removed are chosen based on the cost that they add to the plan. By removing these actions, the LPG achieves these open preconditions in a different context than they were considered before. It is possible that there is a better way to solve these preconditions that is available now because the partial plan that is the context for resolving the precondition has changed. This partial plan is taken as an input for the heuristic search. The deception planner uses this ability of LPG to consider whether the candidate plan it has created is actually optimal.

4.2 The Deception Planner

The deception planner is implemented as a modification and extension of the LPG planner. Like LPG, its input includes the plan actions, initial state, and planning goals held by the target. Beyond this standard planning input, the deception planner takes as input the ulterior goals held by the deceiver, the target's beliefs and knowledge, and observation rules. Its output consists of the set of possibly false statements about the initial state that the deceiver should tell the target to cause the target to achieve the ulterior goals.

The deception planner finds the plan it wishes the target to execute by performing its own type of planning. Unlike a standard planner, the deception planner is tuned to highly favor plans that also achieve the ulterior goals, and . Also the deception planner is allowed to "modify" the initial state through lies. These lies must fit stringent requirements about when they may be told and when they may be discovered.

Planning Goal:
(towed danielles-car)

Ulterior Goal:
(at danielle house)

Initial State:
(at car road)
(at phn1 gas-stn)
(working g-phn)
(at h-phn house)
(connected road house)
(not (working h-phn)

Actions:
(move ?agent ?from ?to):
Preconditions:
(at ?agent ?from) ∧
(connected ?from ?to)

Effects:
(at ?agent ?to) ∧
(not (at ?agent ?from))

(call ?agent ?phn ?loc):
Preconditions:
(at ?agent ?loc) ∧
(at ?phn ?loc) ∧
(working ?phn)

Effects:
(towed danielles-car)

Danielle's beliefs
(at g-phn gas-stn)
(working g-phn)
(connected road gas-stn)
(connected road house)

Resulting Candidate Plan
(move danielle house)
(dial danielle h-phn house)

Statements told by deceiver
(at h-phn house)
* (working h-phn)
* not (working g-phn)

Figure 4.4: Deception Planning Example. Column 1 shows the planning goals, ulterior goals, and initial state. Column 2 shows the actions allowed in this world. Column 3 shows the beliefs of the target, the plan the deceiver would like the target to develop, and the statements the deceiver gives to get the target agent to act accordingly. Statements with *s next to them are lies.

For example, Figure 4.4 shows an initial state, candidate plan, and resulting statements for the serial killer example used in Chapter 3. The following subsections below give more detail on how the deception planner generates its output, through modifying the deception planner heuristic to handle ulterior goals and effective deception, and through countering alternative solutions.

4.2.1 The Ulterior Goal Heuristic

While the LPG heuristic searches for plans of good quality, the deception planner must search for plans that are of good quality and achieve the ulterior goals in the process of achieving the planning goals. Fortunately, heuristic planning makes it simple to modify the characteristics of the preferred plan. By modifying the function used to rank each plan, the deception planner will to prefer plans that are more likely to achieve the ulterior goals; as a result the deception planner can skew the search towards those plans. In order to prefer plans that will achieve ulterior goals, we add a fourth element to the abstract LPG heuristic given in Formula 4.2.

$$\begin{aligned} H_{DP}(P,a) &= (1+\lambda_m^a * CostToFixMutexes(P,a) \\ &\quad +\lambda_p^a * CostToFixOpen(P,a)) \\ &\quad +\upsilon_{ug} CostToAchieveUltGoals(P,a) \end{aligned} \tag{4.2}$$

CostToAchieveUltGoals (abbreviated CUG) estimates how many steps are needed to modify the current plan so that it achieves the ulterior goals. An ulterior goal must be achieved in the process of achieving the planning goals, meaning that an action that achieves an ulterior goal must also achieve a precondition for another action that is used to achieve the planning goals (or must achieve a planning goal). By definition, if an ulterior goal is achievable and not already asserted in the current plan, then there must be some chain of actions possible that connect from a current open precondition in the plan to an action that achieves the ulterior goal, where every action in the chain asserts a fact that is a precondition for the next action. Intuitively, we define the fact distance between two facts as the minimum number of actions that form such a chain that asserts one fact p and depends on another fact u.

Formally, let $FD(A, q, g)$ define the *fact distance* between q and g using A, where A is the target's action library and q and g are facts. The fact distance from a fact q to another fact g is the minimum number of actions in an ordered list $a_1, a_2, ..., a_n$ required to create a link such that q is asserted by a_1, a_n asserts g, and a_n has a precondition which is asserted by a_{n-1}, and so on. The fact distance of a fact to another fact where no such path exists is infinite.

If the stipulation of a particular precondition p is no longer required and instead the use of any open precondition in a plan P is allowed, the *minimum fact distance* for a fact q in the plan P can be defined.

Let $MFD(A, P, q)$ define the *minimum fact distance* to q from P using A, where action library A is the action library used to create P, P is a partial plan, and q is a fact expressible in the domain used to create P. The minimum fact distance to q from P is the minimum of all the distances between q and the open preconditions in P. $MFD(A, P, q)$ where q is already asserted by some step in P is 0, except as noted below.

Intuitively, $CUG(P)$ is simply the sum of the minimum fact distances for all the ulterior goals. Formally, $CUG(P) = \sum_{i=1}^{n} MFD(A, P, ug_i)$ where $\{ug_1, ug_2, ..., ug_n\}$ is the set of ulterior goals for this deception problem.

The ulterior goal heuristic leads the planner towards solving the ulterior goals opportunistically. Choosing to solve an ulterior goal may mean finding a suboptimal solution to a planning goal, and therefore increasing the `CostToAchievePreconds` weight. Because of these conflicting inputs to the deception planner heuristic, the modified plan heuristic favors solving ulterior goals when it does not add a significant cost to the rest of the planning process. However, achieving the ulterior goals is not optional: the heuristic immediately eliminates plans that cannot possibly be extended to achieve the ulterior goals, because the MFD for that ulterior goal will be infinite.

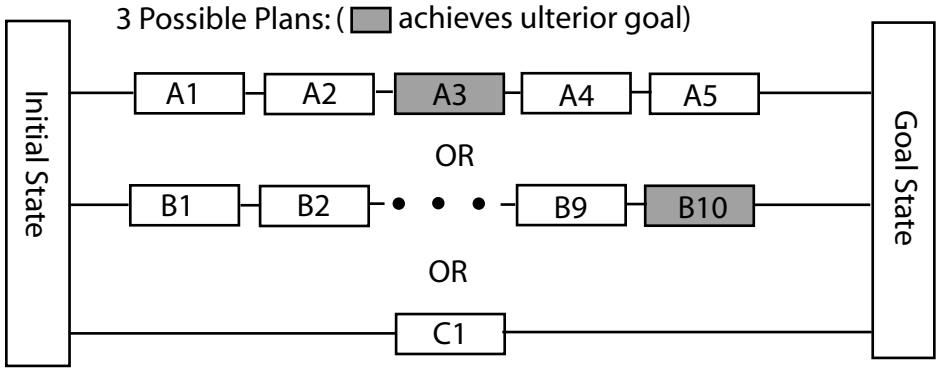


Figure 4.5: Ulterior goal cost example. There are three ways to solve a particular planning problem which differ both in how many total steps they have and how easily they achieve the ulterior goal.

Consider the following example, shown in Figure 4.5. In this domain there are three ways to achieve a planning goal, through chaining actions $A1$ through $A5$ together in order, or through chaining actions $B1$-$B10$ together, or simply applying action $C1$. Assume that action $B10$ achieves the single ulterior goal, as does action $A3$, while action C does not.

At the beginning of the planning process, the planner considers the heuristic values of the neighbors of the empty plan. The neighbors are a plan P_A with the action A5 in it, a plan P_B with the action B10, and a plan P_C with the action C. P_C has a CUG of infinity, and can be thrown out without consideration[1]. P_A has a CUG of 3 since the length of the action chain that links an action that asserts the ulterior goal, $A3$, to an open precondition is 3: $A3$-$A4$-$A5$. The number of steps needed for the plan is 5, making the total cost of this plan $5+3=8$. P_B has a CUG of 1, and a step cost of 10, making the total cost of this plan 11. The deception planner will make P_A its current plan, and start looking at neighbors to this plan.

[1]Actually, plans that have high heuristic values do on occasion get considered by LPG. For some percentage of the choices made by LPG, a random plan is selected from the neighborhood without regard to its heuristic value. This additional check is to help assist in the movement from local minima. It is possible that the planner would then next choose to consider removing some alternate action that asserts some precondition which, once open, allows the ulterior goal to be considered.

Note that it is trivial to construct an example slightly different than the above one that will choose the suboptimal planning solution, using the B actions and thereby creating a plan that is twice as long as neccesary. This possibility is unavoidable, due to the conflicting goals of the deception planner. Since ensuring that ulterior goal's achievement is a necessary precondition for successfully completing the plan, we boost plans that achieve ulterior goals quickly, despite the fact that this additional weight may cause the planer behave less optimally. Indeed, even without this extra weight, the planner is not guaranteed to behave optimally, because its heuristic is not completely accurate.

The above ulterior goal heuristic favors plans that achieve all of the ulterior goals, but it does not specify whether they are all true at the same time. All the ulterior goals must be asserted at a particular point in the execution of a plan P for P to be considered a valid candidate plan. It is possible that the execution of a plan may accomplish one ulterior goal u_1 at time t_1, and assert the negation of that goal at time t_2, and assert a second ulterior goal u_2 at time t_3, where $t_1 < t_2 < t_3$. In this case, one of the ulterior goals must be reasserted — either u_1 must be asserted after t_2, or u_2 must be asserted before t_2. However, the naive CUG of a plan with this configuration would be 0.

This problem stems from the delete effects of actions, and can be even more complicated. Imagine that there is one point in the plan where u_1 is asserted, but not u_2 or u_3, and another where u_2 is asserted, but not u_1 or u_3, and another where u_3 is asserted. The ulterior goal cost would still be 0, and now there are three potential levels at which all the ulterior goals could be asserted.

This problem can be eliminated by modifying CUG so that it recognizes the possibility for multiple potential time points where all of the ulterior goals are asserted at the same time. We call this new cost function CUG^+. The modified ulterior goal heuristic is as follows:

Let S be a group of connected steps in A such that the step before the first step in S deletes an ulterior goal or is the initial state and the next step after the last step in S deletes an ulterior goal or is the goal state. For every such group S, calculate $CUG_r(P, S)$ (restricted CUG).

$CUG_r(P, S)$, where S runs from time t_b to t_e, sums the MFD for each ulterior goal with the following restriction:

1. If the ulterior goal u is deleted at $t_d < t_b$, then any assertion of u before t_d is ignored, and MFD may not consider any open preconditions before t_d.

2. Any assertion of any ulterior goal after t_e shall be ignored.

Note that there is no restriction about whether MFD may consider open preconditions after t_e. This is because a precondition may be filled at any time, so long as it is not deleted by any intervening action. So a precondition for the final action in a plan could be asserted by the first action in the plan.

CUG^+ is equal to the minimum value of CUG_r for all S. In simple cases, this calculation is equivalent to the original CUG.

Consider the previous example using the actions A1-A5 and B1-B10. Assume now that there are two ulterior goals, UG1 and UG2. Assume that A1 asserts UG1, A5 asserts UG2, but A3 deletes UG1. Also, let B1 assert UG1 and UG2. As before, P_A is a plan in the neighborhood that has one action, $A5$ in it, and P_B is a parallel plan with $B10$ in it. Note that since no action currently in the plan deletes an ulterior goal, the calculation of CUG^+ is equal to CUG. $CUG(P_A) = MFD(UG1, P_A) + MFD(UG2, P_A) = 1 + 5 = 6$, and $CUG(P_B) = 10$, so P_A's neighborhood will be expanded.

Eventually, the search will evaluate the partial plan with the actions A3-A4-A5. As noted, action A3 deletes UG1. The addition of A3 will create two alternate positions where the ulterior goals could be achieved, before A3 and after. Correspondingly, there are two S values, one called S_1 including A3's precondition and the other one containing the open preconditions of the steps which are after A3. In this case there are no open preconditions after A3. $CUG_R(P, S_1)$ will be infinity, because while $A1$ asserts $UG1$, there is no way to achieve $UG2$ with the current open preconditions. Similarly, $CUG_R(P, S_2)$ will be infinity, because while $UG2$ is already achieved, MFD may not consider using the one open precondition for $UG1$, since it is before $A3$, which deletes $UG1$.

The ulterior goal heuristic must also handle the effects of lies on the achievement of ulterior goals. As noted in the control cycle for the target agent, shown in Figure 3.1, the target will stop executing a plan if its preconditions are false. To handle the possibility of lie discovery stopping the achievement of an ulterior goal, $CUG^{+}(P)$ does not consider an ulterior goal u achieved if u is only asserted by an action that depends upon a lie, or by an action which occurs after a action that depends upon a lie. This restriction can be stated as requiring that S groups cannot include actions that have a precondition which is asserted by a lie, and cannot come after any action that depends on a lie.

4.2.2 The Lie Heuristic

The ulterior goal heuristic ensures that the ulterior goals are all asserted simultaneously at some point during the execution of the candidate plan. A similar heuristic is added to the deception planner to regulate the use of lies in the deception planner. After its addition, the deception planner heuristic is as follows:

$$\begin{aligned} H_{DP}(P,a) \quad = \quad & (1 + \lambda_m^a * CostToFixMutexes(P,a) \\ & + \lambda_p^a * CostToFixOpen(P,a)) \\ & + \upsilon_{ug} CostToAchieveUltGoals(P,a) \\ & + \upsilon_{lie} CostToFixLies(P,a) \end{aligned} \tag{4.3}$$

CostToFixLies (abbreviated CFL) is 0 when the deception planner has told no lies which will be observed before all of the ulterior goals are achieved. Therefore, the deception planner heuristic reaches 0 when all of the planning goals and ulterior goals are achieved in a complete plan, and the lies used in this plan are not observed before the ulterior goals are achieved. Such a plan is a complete plan.

Lies may only be used in specific ways in generating a deception plan. Lying to achieve ulterior goals is only practical if the lie is not discovered or is only discovered after the ulterior goals are achieved. Within the model used by the deception planner, a lie is discovered when an observation rule for the condition being lied about is triggered. Keeping the target agent from observing a lie is critical to successfully deceiving the target. A lie can also be discovered when a lie is used as a precondition for an action, and the action is attempted. The action will fail because its precondition is not true.

```
Trigger: (a ∧ b ∧ c)
Effect: (p1 ∧ p2)

Trigger: (d ∧ e ∧ f)
Effect: (p2)
```

Figure 4.6: Sample Observation Rule

The algorithm must ensure that the observation triggers for a lie l are false for as long as necessary — that is, until a point in the plan where all of the ulterior goals are met. For example, consider the observation rule in Figure 4.6. If the deception planner uses a lie about the fact $p1$ (by stating $p1$ or (not $p1$), whichever is false), the deception planner must ensure that $\neg(a \wedge b \wedge c)$ holds until all of the ulterior goals are satisfied. Similarly, if the deception planner depends on a lie about p2, then both observation triggers listed must be kept from firing, meaning that at every point in the plan up until the ulterior goals are achieved, $\neg(a \wedge b \wedge c) \wedge \neg(d \wedge e \wedge f)$ must hold.

These negated observation triggers, collated by the condition they observe, form a *lie precondition* for a telling a lie. This precondition must hold at every step of the candidate plan until all of the ulterior goals are achieved. Doing so will ensure that the lie will not be observed by the target agent.

It is reasonable to allow the way the a lie precondition is held to change through time. For example, if you do not want an agent to see inside a box, you may at one point keep the agent out of the room the box is in, and at another point close the box. Using the above abstract example, it is acceptable to have $\neg a \wedge b \wedge c \wedge \neg d \wedge e \wedge f$ satisfy the preconditions for $p2$ at one point in the plan, and $a \wedge \neg b \wedge c \wedge d \wedge \neg e \wedge f$ at another point, since both sets of states keep the observation rules from being triggered. In order to ease these calculations, lie preconditions are converted into disjunctive normal form (DNF). The DNF of the above observation rule, for example, would list $(\neg a \wedge b \wedge c \wedge \neg d \wedge e \wedge f) \vee (a \wedge \neg b \wedge c \wedge \neg d \wedge e \wedge f)$ $\vee (a \wedge b \wedge \neg c \wedge \neg d \wedge e \wedge f)$, and so on, with all possible combinations of negating the first observation trigger combined with all possible ways to negate the second observation trigger, creating a total of 7 * 7 = 49 possible ways to keep $p2$ from being observed.

Formally, let a *lie precondition* for a fact f be the conjunction of the negated observation triggers which list the fact f in the effect of the observation rule.

The lie preconditions are used by $CostToFixLies(P)$ to determine how hard it is to fix P so that a lie which is currently observable is not observable. The CFL calculates the *minimum observation negation distance* (MOND) for each lie that is observed. MOND is an estimate of how many steps must be added to keep a lie from being observed.

$$CFL(p) = \begin{cases} 0 & \text{if no ulterior goal has yet been achieved} \\ 0 & \text{if no lie has been observed before an ulterior goal} \\ \sum_{l \in Lies(P)} MOND(P, l) & \text{otherwise} \end{cases}$$

The $MOND$ for a lie l in P depends on how many times during the execution of P the lie l would be observed. We let $Q(P, l)$ be the set of sets of actions $q_1 = \{a_n, a_{n+1}, ..., a_m\}$, such that before a_n and after a_m, l is not observable, or is observable due to a different trigger, and any achievement of an ulterior goal is after a_m.

Let $LP(l) = C1 \vee C2 \vee C3$ be the lie precondition for l in disjunctive normal form. Let $\{f_1, f_2, ..., f_n\}$ be the facts that make up one clause in the lie precondition.

$$MOND(P, l) = \sum_{q \in Q(P,l)} MIN(\forall_{C \in LP(l)} \sum_{i=1}^{n} MFD(f_i, P)) \tag{4.4}$$

Intuitively, the MOND for a lie is the sum of the number of steps that must be added to remove the possibility of observation of the lie at each step in the plan.

Although this definition of CFL is close to the actual heuristic used in the deception planner implementation, some optimizations have been made. In fact all lie observations are fixed simultaneously, starting by adding those facts that are necessary to deny the observation of the lie at level one, and then adding those facts that are necessary to deny the observation of the lie at level two, and so on. At each level, the facts asserted to deny the observation of the lie at one level are carried to the next level, unless some action in the plan specifically deletes a fact. This way of calculating the CFL, while estimating the same potential cost as the above algorithm, deletes the duplication that takes place in the above serial calculation of $MOND$.

```
Initial State:  (INIT O1)

A1: Pre: (INIT) Add: (P2) Delete: (O1)
A2: Pre: (P2) Add: (P3 O2)
A3: Pre: (P3) Add: (P4 ULTGOAL O2)
A4: Pre: (P4 L) Add: (P5 O2)
A5: Pre: (P5) Add: (GOAL ULTGOAL)

Observation Rule:
Trigger: (O1 ∧ O2)
Effect: (L)
```

Figure 4.7: A sample initial state and set of actions

For example, consider a domain where there are is a single solution to a planning problem that uses actions $A1$-$A5$, as described in Figure 4.7.

The only possible standard plan that can be created is one that contains the actions $A1$-$A2$-$A3$-$A4$-$A5$ in that order. $A5$ and $A2$ both assert the ulterior goal ULTGOAL. $A4$ relies on a false fact, L. The target agent has a single observation rule, which observes L and requires that $O1$ and $O2$ be true to be triggered. Actions $A3$ and $A4$ assert one half of the observation trigger, and the other half is asserted in the initial state.

When the deception planner is given this domain, it will first add the action $A5$, since that is the only action that solves the planning goal $GOAL$[2] . After this action is added the CUG^+ value will be 0 as well, since A5 achieves the ulterior goal.

The planner calculates CFL(P) once it has added the action A4, since A4 depends on a precondition which is false, but can only be asserted by lying. If it were possible to assert L either by lying or by achieving L through some action, the algorithm would wait to consider a lie as having been told until the precondition L had been filled by a specific examination of the alternatives for filling that precondition.

[2]In reality, when given only one choice of action, the deception planner does not calculate the heuristic values, since there is no point. For pedagogical purposes, we calculate the values anyway.

$CFL(P)$ when $P = A4$-$A5$ is 0, however, for two reasons, either of which would be enough to ensure that the CFL was 0. First, the lie cannot currently be discovered by observation. The preconditions for the observation trigger, $O1 \wedge O2$, are not true at any point before the use of L. When L is used, it will be discovered anyway by the target when the target tries to perform some action whose preconditions are false.

Second, the ulterior goal ULTGOAL is no longer asserted at a valid point in the plan and therefore there is no cost associated with lying. Note that the CUG^+ value for this plan must be recalculated, since $A5$, which previously achieved the ulterior goal, is after an action which depends on a lie, and thus will not be executed by the target agent. The new value for CUG^+ is 1, because $FD(P4, ULTGOAL) = 1$.

The action $A3$ is added next. $A3$ reasserts $ULTGOAL$, and also asserts $O2$. The lie cost heuristic value is still 0, because the assertion of $O2$ takes place *after* the execution of the action $A3$, which asserts ULTGOAL. Although the ulterior goal and the observation occur at the same time, the algorithm assumes that the execution of the action $A3$ is impossible to retract once started.

When $A2$ is added, it asserts $O2$. As this point, both $O1$ and $O2$ are true at time point 2, which means that L can be observed before the execution of the action that achieves the ulterior goal. The lie cost heuristic is finally greater than 0. The minimum observation negation distance for L is one - the precondition $P2$ can be linked to $(notO1)$ through $A1$. When $A1$ is added, the lie cost is 0, $CUG^+(P)$ is 0, and the LPG heuristic value is 0, meaning that the plan in question is complete.

4.2.3 Negating Competing Plans

Given this plan, $A1$-$A2$-$A3$-$A4$-$A5$, what is the output at this point from the deception planner? Assuming that the target agent has no previous knowledge or beliefs, the deceiver needs to pass on enough information to allow the target to develop the desired plan. However, it may be that, given this information, there is still a better plan that the target agent can develop which does not achieve the ulterior goals. For example, if there were an action B that achieves the planning goal and relied on the initial state, the target would prefer a plan consisting solely of action B. Such a plan is called a competing plan. The final stage of the deception planner negates competing plans.

If there are no competing plans, the deceiving agent passes along the minimal amount of information necessary in order to allow the target agent to create the candidate plan. This minimal communication follows common sense, and it has positive benefits: the planning environment of the target agent is limited, which means there are fewer competing plans to consider. Because they are built from the target agent's limited knowledge of the planning world, competing plans may only rely on knowledge that the target agent knows beforehand (as stated in $B(a)$ and $K(a)$) or that the deceiving agent gives the agent in order to make the candidate plan seem viable.

When finding competing plans, the deception planner starts with an initial state containing only this bare minimum of information needed to permit the target to develop the candidate plan. This information constitutes the *competing initial state*. The competing initial state is divided into a *static competing initial state* and a *dynamic competing initial state*. The static portion of the initial state are those facts which cannot be lied about to negate competing plans. The dynamic portion of the initial state can be lied about and may change during the process of countering competing plans.

The static competing initial state contains the lies used in the candidate plan, those facts in the initial state that are depended upon by the candidate plan, and K, the relevant knowledge of the target agent. The dynamic competing initial state consists of those facts *believed* by the target that are not contradicted by the static competing initial state.

Potential competing plans will be countered by lying about some belief that is in the dynamic initial state. It is possible to determine which beliefs can be lied about by checking to see whether the lie will be observed before the ulterior goals will be achieved during the execution of the candidate plan. If this is not the case, then the lie can be told.

LPG is run with the competing initial state, planning goals, and initial actions as input. If the planner returns a plan P_c that is shorter than the candidate plan, the new plan is a competing plan. Those facts in the initial state that P_c depends on for which there is a possible lie can be lied about to counter P_c. A lie from this set is added to the static initial state, and the corresponding belief is removed from the dynamic initial state.

This process continues until the time alloted is exhausted, or a competing plan is discovered which cannot be countered by lying about some piece of initial state. If a competing plan is found which is impossible to counter, the candidate plan must be repaired, or an entirely new candidate plan must be discovered. The deception planner uses plan repair to improve the existing candidate plan [13]. Plan repair works as it does in LPG and is described in section 4.1.4.

The full pseudocode for generating and countering competing plans is given in Figure 4.8. The competing plan generation depends on two helper functions, `CreateCompetingInitialState`, shown in 4.9, and `DeterminePossibleLies`, shown in Figure 4.10.

For an example of countering competing plans, consider plan situation described in Figure 4.11. The candidate plan is A1-A2. The static initial competing state consists of (INIT), since that is the only fact relied on by the candidate plan, and no lies were needed. The dynamic competing initial state includes BINIT, BINIT2, and CINIT. By comparing the observation rules against the candidate plan, it can be seen that a lie about BI2 will be discovered before the ulterior goal can be achieved in the candidate plan. So, BINIT2 is moved from the dynamic to static initial state. Lies about BINIT1 and CINIT are allowed.

Given the competing initial state (INIT BINIT1 BINIT2 CINIT), the LPG planner is able to develop a competing plan consisting of the action C. Since we can lie about the truth value of C's precondition CINIT, the deception planner counters the competing plan by doing so. (not CINIT) is added to the static competing initial state, and CINIT is removed from the dynamic initial state. Now, the competing initial state is (INIT BINIT1 BINIT2 (not CINIT)). The competing plan B1-B2 can be found. As mentioned, no lies about BINIT2 are possible, however, a lie can be told about BINIT1, making the plan B1-B2 fail. No further competing plans can be found, and thus the statements (INIT, not(BINIT2), not(CINIT)) will be passed as output from the deception planner.

```
p = CreateCandidatePlan(Domain,Agent)

do forever:
   (possibleLies,dynInit,staticInit) = getCompetingInit(p,possibleLies)
   competingInit = staticInit ∪ dynInit
   cp = createCompeting(competingInit,PlanningGoals,Actions)

   //AllAvailLies lists all the possible lies
   //that could be told, ignoring the candidate plan
   //possibleLies are those lies that can be told
   //given the candidate plan

   possibleLies = AllAvailLies ∩ Not(dynInit)
   (possibleLies,dynInit,staticInit) =
   determinePossibleLies(possibleLies,dynInit,staticInit)

   do forever:
      if(could not find cp)
         if(not enough time spent):
               cp = createCompeting(competingInit,PlanningGoals,Actions)
               repeat inner do loop
         else
               return statements

      //The resulting competing plan is not better than the candidate plan.
      if(p is shorter than cp)
         cp = repairCompeting(competingInit,PlanningGoals,Actions)
      //The competing plan is actually a better candidate plan
      if(cp is shorter than p and achieves ultgoals)
         p = cp
         repeat outer do loop
      //Try to counter competing plan through counter a current belief:
      (dynInit,staticInit,statements,possibleLies) =
      CounterCompetingPlan(cp,P)
      if(CounteredCompetingPlan(cp)
            cp = createCompeting(competingInit,PlanningGoals,Actions)
            repeate inner loop
      //Could not find counter to shorter competing plan!
      P = RepairCandidatePlan(P,Domain,Agent)
      repeat outer do loop
```

Figure 4.8: Pseudocode for the entire deception planner

```
function CreateCompetingInitialState
            (dynInit,realInit,possibleLies,statements):
      for each belief in dynInit:
         if(cp depends on belief)
              if((belief ∉ realInit) or ((not belief) ∈ possibleLies))
               staticInit = staticInit ∪ (not belief)
               statements = statements ∪ (not belief)
               dynInit = dynInit - belief

               possibleLies = possibleLies - (not belief) - belief
               competingInit = staticInit ∪ dynInit
```

Figure 4.9: Pseudocode for countering a competing plan

```
function DeterminePossibleLies(possibleLies,dynInit,statcInit)
   staticInit = Lies(P) ∪ NeededInitialState(P) ∪ Knowledge(agent)
   dynInit = (Beliefs(P) - staticInit) - Not(staticInit)
   possibleLies = PossibleLies ∩ Not(dynInit)

   for i = 0 to ultgoalLevel:
      for each lie in possibleLies
         if(observable(lie,i))
            possibleLies = possibleLies - lie
             dynInit = dynInit - not (lie)
             staticInit = staticInit ∪ not (lie)

   statements = Lies(P) ∪ NeededInitialState(P) - Knowledge(agent)
```

Figure 4.10: Pseudocode for creating the competing initial state

```
Initial State: (INIT BINIT1 BINIT2 CINIT)

Actions:
A1: pre: (INIT) add: (PA2)
A2: pre: (PA2) add: (GOAL ULTGOAL)
B1: pre: (INIT BINIT1) add: (PB2)
B2: pre: (PB2 BINIT2) add: (GOAL)
C: pre: (INIT CINIT) add: (GOAL)

Observation Rules:
(PA2) -> (BINIT2)

Agent Beliefs:
(INIT BINIT1 BINIT2 CINIT)

Candidate Plan:
A1-A2
```

Figure 4.11: Example for countering competing plans. Two shorter competing plans can be found and countered.

Chapter 5

Example

In the following section we give an example of the deception planner from the input given to its output. The example given was developed in order to illuminate the properties and capabilities of the deception planner. The numbers and choices given in this section mirror those produced by an actual run of the deception planner algorithm. This example with actual the program input files, outputs, and choices, is given as a printout in Appendix B and C.

5.1 Initial Setup

In this example, we use a setting that might be appropriate in a very simple narrative. The world described is a small building with several connected rooms, and a key. The target desires to take the key to the goal room. The ulterior goal of the deceiving agent is to get the target agent into another room, called ug-room, carrying the key. There are two actions possible in this world: move, and take. The initial world state, the planning goal, the ulterior goals, and the actions possible in this world are all listed in Figure 5.1.

In this example there are two observation rules. The first rule lists those facts that can never be lied about, because their state is always observable to the agent. An observation rule shows it will always be triggered by having an empty trigger, implying that the observation does not require any fact to be true before the rule is triggered. In this case, the target agent always knows his own location, and always knows what he is carrying. The second observation rule states that the target agent knows about everything in the room he is in, including what rooms are connected to that room. The observation rules are shown in the third column of Figure 5.1.

Finally, we state that the agent *believes* he understands the layout of the world, represented in the current example as a list of connections between rooms but will accept new knowledge about the connections. The listing of the agent's beliefs is equivalent to the list of facts about the *connected* predicate in Figure 5.1. The target agent has no absolute knowledge about the current situation beyond those necessitated by the empty observation rule: his current location, and his current belongings.

Figure 5.2 shows the empty plan created by the example. The initial state is abbreviated; displaying the entire initial state would require too large a diagram. We will show figures of the development of this plan while progressing through the deception planner's planning process.

5.2 Preprocessing

The deception planner performs some preprocessing in order to prepare for the execution of the deception planner's main algorithm. The preprocessing consists of transforming observation rules into lie preconditions, as discussed in Section 4.2.2, and calculating the fact distances that are needed for both the ulterior goal and lie heuristic.

Planning Goal:
(at target goal-room) ∧
(has target key)

Ulterior Goal:
(at target ug-room) ∧
(has target key)

Initial State:
(at target startroom)(K)
(at key room-a)(U)
(connected start-room room-a)(B)
(connected room-a start-room)(B)
(connected room-a room-b)(B)
(connected room-b room-a)(B)
(connected room-a ug-room)(B)
(connected ug-room room-a)(B)
(connected ug-room room-a)(B)

Actions:
(move ?agt ?frm ?to):
Preconditions:
(at ?agt ?frm) ∧
(connected ?frm ?to)

Effects:
(at ?agt ?to) ∧
(not (at ?agt ?frm))

(take ?agt ?obj ?loc):
Preconditions:
(at ?agt ?loc) ∧
(at ?obj ?loc)

Effects:
(has ?agt ?obj)
(not (at ?agt ?loc))

Observation Rules
Null Obs. Rule:
Trigger: Empty

Effects:
(at target ?loc) ∧
(has target ?obj)

Obs. Rule #2:
Trigger:
(at target ?loc)

Effects:
(at obj ?loc) ∧
(connected ?loc ?loc2)
(connected ?loc2 ?loc)

Figure 5.1: Deception Planning Example. Column 1 shows the planning goals, ulterior goals, and initial state. States that the agents believes or knows to be true are marked with a (B) or (K). States unknown to the agent are marked with a (U). Column 2 shows the actions allowed in this world. Column 3 shows the observation rules of the target agent.

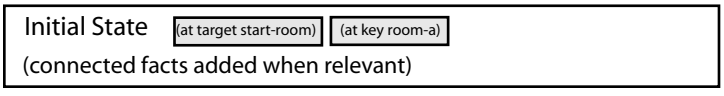


Figure 5.2: Initial State before any actions added

Lie preconditions are created by considering each false fact f in turn, and gathering all the observation rules whose effect contains a partially grounded condition that unifies with f[1]. These gathered observation rules, when triggered, could cause f to be observed. In the example given in this chapter, all facts that use the predicate *has* will unify with the effect of the null observation rule, meaning that they will always be observed. The lie precondition for all facts involving the *has* predicate can be considered to be false, meaning that such lies can never be told.

Facts using the *at* predicate could unify in two ways. If the fact is of the sort *(at target ?loc)*, then it will unify with the null observation rule, and its lie precondition will be false. Otherwise, it will unify only with the effect of the second observation rule. Given a fact *(at key some-loc)*, the trigger that will cause this fact to be observed is *(at target some-loc)*. The lie precondition for this fact is the negation of this trigger, which is *(not (at target some-loc))*.

Facts involving the *connected* predicate are slightly more complicated. They do not ever unify with the null observation rule. However, they all unify with two effects of the second observation rule, both *(connected ?loc1 ?loc2)* and *(connected ?loc2 ?loc1)*. The double listing of the connected predicate in this observation effect reflects the fact that the connected predicate is reflexive: if point a is connected to point b, then the reverse is also true, at least within this example. When the agent observes one of the these facts, it is guaranteed to observe its reverse. However, these two observations have no special relation.

For example, let f be *(connected ug-room goal-room)*. The second observation rule o_2 has effects that can translate to f through the application of a substitution s. The first effect from the o_2 will translate to f using $s = \{?loc1 = ug - room, ?loc2 = goal - room\}$. When s is applied to the observation trigger for , the trigger *(at target ?loc1)* is translated to *(at target ug-room)*. In the second effect, *(at target ?loc1)* is translated to *(at target goal-room)*. Thus the observation trigger for this fact is *(at target ug-room)* $\vee$ *(at target goal-room)*. The lie precondition for this fact is the negation of its observation trigger, and is therefore *(not (at target ug-room)* $\wedge$ *(not (at target goal-room))*.

[1]In the implementation, a more efficient process is used that gathers facts together in groups according to which observation rules affect them.

During initialization, the fact distances described in Section 4.2.1 are also calculated for all fact pairs. The algorithm for calculating these values is a simple variation on the all-pairs shortest path algorithm. The distance between a fact and itself is 0. The distance between two facts that are effects of the same action is 1, because only that one action needs to be added to achieve the second fact if the first fact is a precondition. The distance from a fact f_e that is an effect of some action a_1 to a precondition f_p of a_1 is 2, assuming that there is some other action a_2 that asserts f_p, and there is no action that asserts both f_e and f_p. The action chain a_2,a_1 fills asserts f_p on its way to assert f_e. Finally, for every additional action that must be added in the chain from the effect of one action to the precondition of another action, 1 is added to the fact distance. As described in Section 4.2.1, if there is no such chain, the distance is infinity. Note that this function is not symmetric — the distance *from* some fact f_1 *to* some fact f_2 is not necessarily equivalent to the distance from f_2 to f_1.

Consider the distances *to* the ulterior goal (at target ug-room) from potential preconditions in the plans the deception planner could create. These distances will be used repeatedly during the planning process. The distance from *(at target goal-room)* is 2, because there is an action *(move target ug-room goal-room)* which has the fact we are calculating the distance from *(at target goal-room)*, in its effects, and the fact we are calculating the distance to in its preconditions. In fact, for any two facts of type *(at target ?loc)*, the distance will likely be 2, because it is possible to craft an action that will move from one location to the other.

The fact distance will not always be 2 for all such facts, however, because there are some actions of type *(move target ?loc1 ?loc2)* that are disallowed because they rely on preconditions that can never be true. For example, in this plan there are no lies allowed about the connections to the start-room — they are all disallowed by the null observation hypothesis, which states that the target will observe immediately the connections at its current location. Thus, the fact distance from *(at target goal-room)* to *(at target start-room)* will be 3, because the target must first move to another room from the start-room before moving to the goal room.

For other pairs of facts, the distance will be infinity. For example, there is no action which has as its effect *(connected ?loc1 ?loc2)*, so the fact distance from all such facts to other facts will be infinity. Similarly, there are no actions which require fact *(has target key)*, so the distance to this fact from all other facts is infinity. These infinite distances relate the fact that there is no way to make having the key a precondition for some other action, and thus no way to have an ulterior goal requiring the target to get the key, unless the target already wants to get the key for its own reasons.

5.3 Neighborhood Creation

After the preprocessing is completed, the deception planner begins running the LPG search algorithm with its modified heuristic. During each round of the search, an open precondition is picked, and the current plan's neighborhood is created based on that precondition. The plan has two flaws, *(at target goal-room)* and *(has target key)*, as shown in Figure 5.2. The deception planner picks a flaw, in this case *(at target goal)*, and generates the plan neighborhood based on this flaw. The LPG algorithm creates each neighbor plan by picking an action which asserts the open precondition, and adding that action to the current plan to create a new partial plan. If the plan currently had any actions in it, the planner would also consider removing the action that holds the precondition. The actions which achieve this precondition are *(move target start-room goal-room)*, *(move target room-a goal-room)*, *(move target room-b goal-room)*, and *(move target ug-room goal-room)*, so the neighborhood consists of plans that contain these single actions.

5.4 Neighborhood Ranking

Once the neighborhood has been created, each partial plan in the neighborhood is ranked. The ranking is consistent with that described in Section 4.2.2, 1 + CostToFixPreconds() + CostToFixMutexes() + CostToAchieveUltGoals() + CostToFixLies(). As in that section, the full LPG heuristic is beyond the scope of this thesis. The numbers given for the LPG planner in this example match those given in actual execution of the planner, however.

CostToAchieveUltGoals is the minimum fact distance needed to achieve all the ulterior goals. The fact distance for the ulterior goal *(has target key)* is 0, because that fact is a precondition of the current plan. The minimum fact distance for *(at target ug-room)* differs, depending on the neighbor examined. If the action added is *(move target ug-room goal-room)*, then the cost is 1, because the ulterior goal is a precondition in the current plan, meaning that only one action needs to be added to assert the ulterior goal. Otherwise, the cost will be two, for the reasons described in the preprocessing section. CostToAchieveLies is always 0, because no lies have been told yet.

The ulterior goal heuristic affects the rankings of the neighborhood plans. The unmodified LPG cost of the plan with action *(move ug-room goal-room)* is 12.50, while the unmodified LPG cost for the other plans 12.38. Going through ug-room is an extra step not required by the other plans, which makes the LPG cost relatively higher. However, the value of CostToFixUltGoals for these plans is 2, while the value for the plan with *(move ug-room goal-room)* is only 1. This extra boost makes the plan with action *(move ug-room goal-room)* the plan with the lowest cost.

The *(move ug-room goal-room)* action is inserted into the plan, and the process is repeated. The plan after the first action is inserted is shown in Figure 5.4. The neighbors for this new plan are shown in Figure 5.3.

5.5 Lies

As shown in Figure 5.4, adding the first action adds in two preconditions: *(at target ug-room)* and *(connected ug-room goal-room)*. As in the previous iteration of the search algorithm, the deception planner picks a precondition and chooses to solve it. In this round, the deception planner chooses to solve *(at target ug-room)*. The actions in the neighborhood consist of *(move target goal-room ug-room)*, *(move target room-a ug-room)*, and *(move target room-b ug-room)*. Note that adding any of these actions will assert the ulterior goal *(at target ug-room)*.

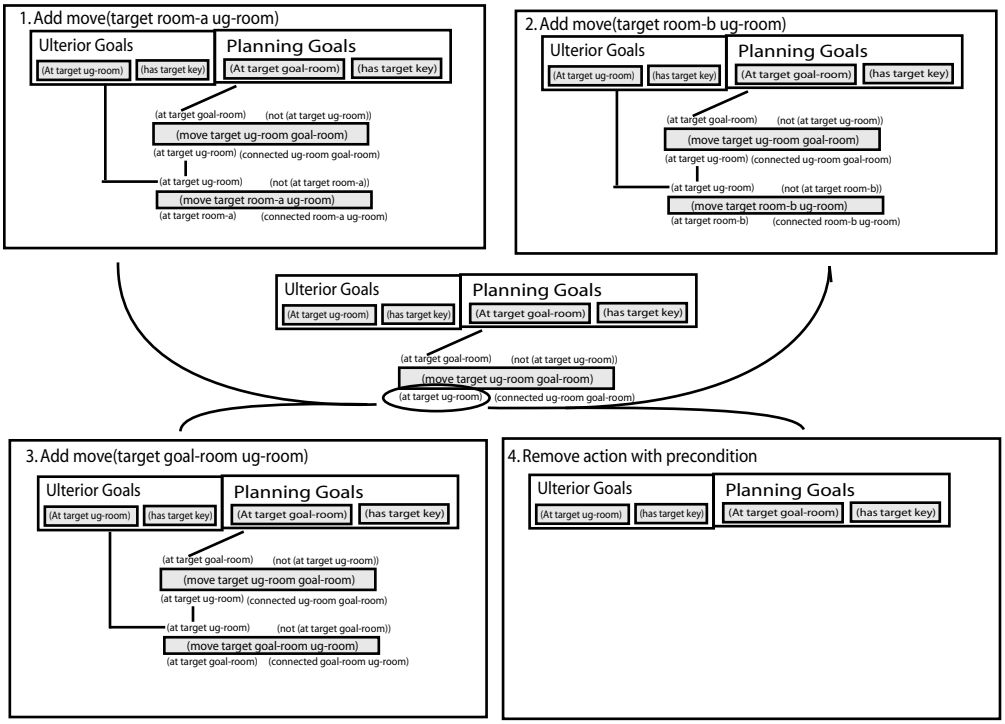
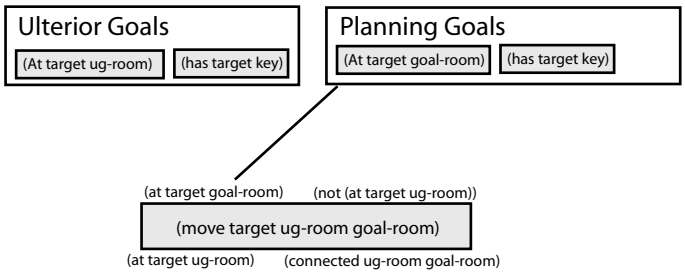


Figure 5.3: Example Neighborhood



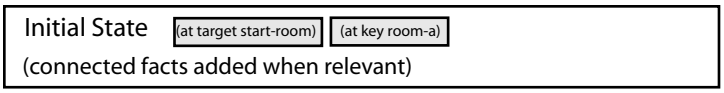


Figure 5.4: Plan after first action added

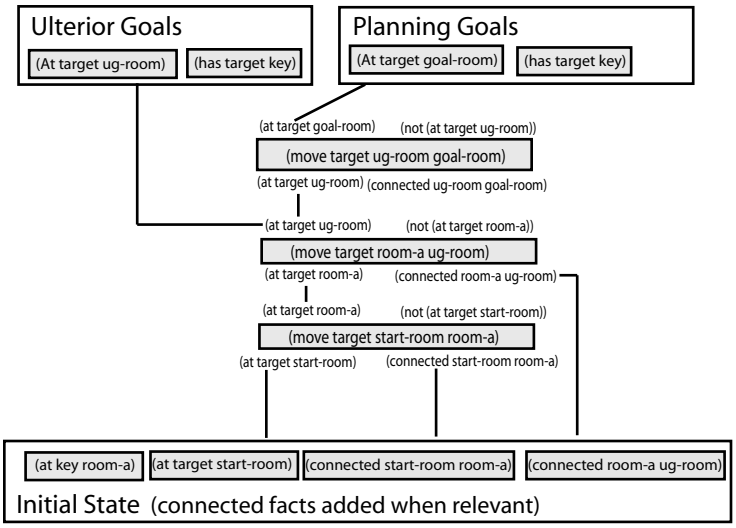


Figure 5.5: The developing candidate plan after actions 2 and 3 are added

However, only the action *(move target room-a ug-room)* does not have a precondition that is a lie. If another action, such as *(move target room-b ug-room)* were added, then the deceiver would have to convince the target that *(connected room-b ug-room)*. Telling a lie at a time that is before the achievement of an ulterior goal would mean that the achievement of that ulterior goal is invalidated — the target is never going to execute a target plan past the point of a lie. For plans that rely on this action cases, the ulterior goal cost rises to 2, based on fact distances from the open *at* precondition. The planner therefore chooses to add in the step *(move target a-room ug-room)* and, for similar reasons, *(move target start-room ug-room)*. The current partial plan has the open preconditions *(has target key)* and *(connected ug-room goal-room)*. The partial plan at this point is shown in Figure 5.5.

At this point, the planner picks *(connected ug-room goal-room)* as its goal. There is only one possible way to solve this precondition: a lie must be added to the plan. The resulting plan is shown in Figure 5.6. After adding the lie, the plan is now ranked considering the lie preconditions for this lie. In this case *(at target ug-room)* and *(at target goal-room)* must be false until all of the ulterior goals are achieved. Currently, *(at target ug-room)* is true *as* the ulterior goal is achieved, which is acceptable. Because there is no lie that is discovered too early, the lie cost of this plan is still 0.

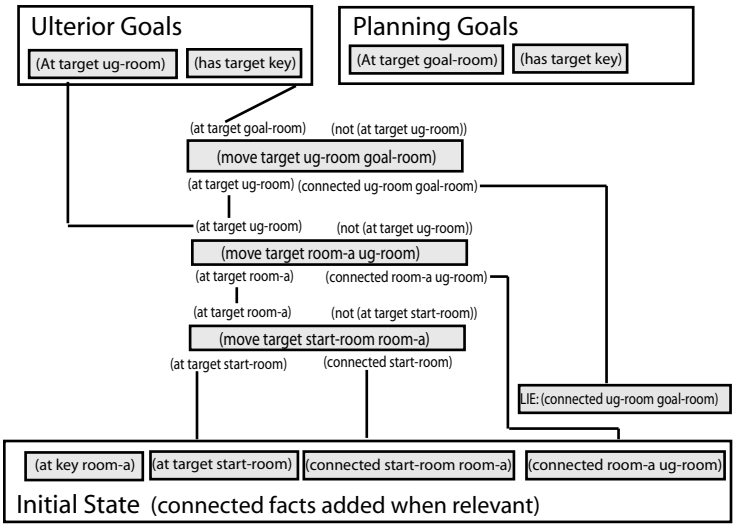


Figure 5.6: Add lie

5.6 The Final Ulterior Goal

The final ulterior and planning goal is *(has target key)*. The deception planner creates a neighborhood of eight actions that achieve this goal. These eight actions show how the LPG planner considers adding actions to a plan at multiple positions in that plan, if the preconditions of the action interact with the effects of actions currently in the plan. In this case, the actions, such as *(take target key room-a)*, interact with effects of other effects in the plan, which at one point assert the fact *(at target room-a)* and at a later point deny it again. The total list of eight actions considers taking the key in all four available rooms, both after the target has moved into room-a, and after the target has left that room. *(take target key room-a)* is chosen as the best action, completing the plan. The final candidate plan is shown in Figure 5.7.

The above candidate plan succeeds in achieving the ulterior goals and not getting caught in a lie before the ulterior goals are achieved. At some point during the execution of the above plan, the target agent will be in the ulterior goal room with the key.

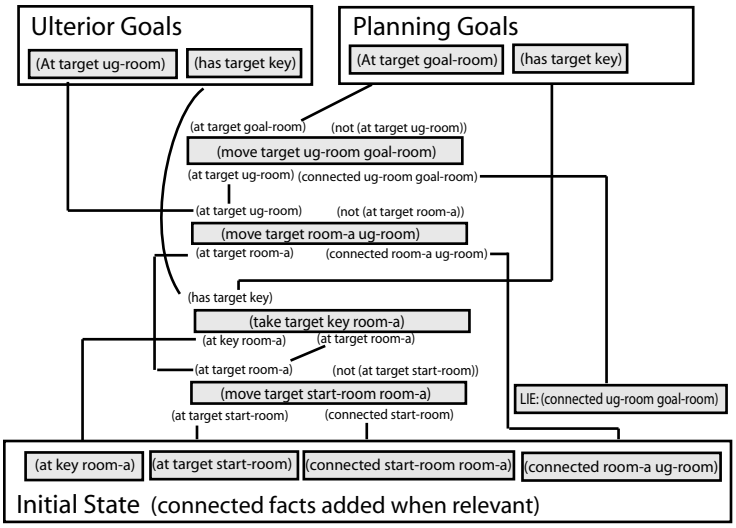


Figure 5.7: Take key and complete plan

5.7 The Competing Plans

To ensure that this candidate is chosen, the deception planner rules out any competing plans. The competing plans are plans that the target agent might choose as alternatives for the candidate plan. They are only created using information that the target agent already knows or believes, or information that the deceiving agent gives the target agent so that the target agent can create the candidate plan.

In this case, the deceptive planner will use the following as the static competing initial state: {*(at target start-room) (connected start-room room-a) (connected room-a ug-room) (connected ug-room goal-room) (at key room-a)*}. These facts are depended on by actions in the candidate plan.

The dynamic competing initial state includes all those facts in the target agent's belief set that are not either asserted or denied in the static initial state. In this case this dynamic initial state consists of {*(connected room-a room-b) (connected room-b goal-room) (connected room-b room-a) (connected goal-room room-b) (connected room-a start-room)*}.

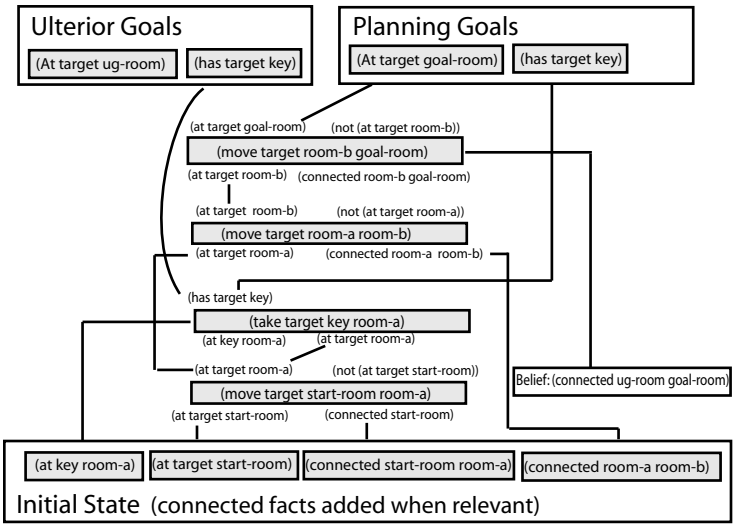


Figure 5.8: A competing plan for the chapter 5 example. This plan achieves all the planning goals but not all the ulterior goals.

Facts are moved from the dynamic initial state to the static initial state by calculating whether a lie about the fact would be observed during the execution of the candidate plan before the achievement of the ulterior goals. If the deceiving agent is caught in a lie, even if the lie simply denied some fact that is irrelevant to the candidate plan, the target agent is assumed to no longer trust any of the information given by the deceiver. In the current candidate plan, the dynamic fact *(connected room-b room-a)* is observed when the second observation rule is triggered by *(at target room-a)*. The other dynamic beliefs are observed during the executing of the candidate plan as well. However, the facts *(connected room-b goal-room)* and *(connected goal-room room-b)* are observed only *after* the achievement of the ulterior goals, making their observation moot. After the pruning of observable facts, the dynamic initial state contains only these two actions. All of the other facts about the connected predicate are moved into the static competing initial state.

The competing initial state is the union of the dynamic and static competing initial state. A standard planner is run using the competing initial state, the available actions, and the planning goals as input. Given the current initial state, a plan that moves through room-b instead of ug-room to get to the goal room could be created. This plan is equal in worth to the candidate plan (it contains the same number of steps), so it is a dangerous competing plan. The competing plan is shown in Figure 5.8. Initial state that is static is shown in gray boxes. Dynamic initial state is in white boxes.

To counter a competing plan, the deception planner must find a piece of dynamic initial state that is that is used to achieve a precondition by the competing plan. In this case, there is only one such fact: *(connected room-b goal-room)*. Thus, the deception planner removes the fact *(connected room-b goal-room)* from the dynamic initial state, and inserts *(not (connected room-b goal-room))* into the static initial state, while marking it as a lie.

The standard planner is run again with the updated initial state. It will return the candidate plan, since that is now the shortest way to achieve the planning goals, given the competing initial state. In order to try to improve upon this plan, random flaws will be inserted into the candidate plan, and the standard planner will be run again (this process is the plan repair and optimization process described in Section 4.1.4). When the process has run for enough time, the planner will accept the candidate plan as the best plan available.

At this point, the competing initial state contains those statements that must be communicated to the target agent. Only the information not already known or believed by the target agent will be communicated. In this case, that will be *(connected ug-room goal-room) (at key room-a) (not (connected room-b goal-room))*. This information should influence the target agent to act out the candidate plan and thus achieve the ulterior goals.

Chapter 6

Future Work

No research project is ever finished, and that is doubly true for research projects in artificial intelligence. It is therefore always a useful exercise to list the directions in which a project could be taken as time progresses. Below we list several such potential developments, experiments, and integrations from which the deception planner would benefit.

6.1 Improvements to the Target Agent Model

The complexity of the deception planner is directly related to the complexity of the target agent model. Although the scope of the agent in the deception planner was picked with an eye towards modularity, a more complex agent would allow for more ways to deceive and more ways to be stymied in deception. Below are some of the possibilities we see as the most interesting.

The current deception planner's implementation of the agent's thought process is limited in some ways. For example, the observation rules fail where the logical ties between world states are not explicitly stated. For example, if the agent sees a unique object he is looking for in one room, he could deduce that the object is not in any other room. Similarly, if someone states that point a is connected to point b, the reverse can be deduced and used in planning. One way to give the target this capability is to implement deduction rules in a similar manner as observation rules. Where observation rules are triggered based on the state of the world as the target agent executes its plan, deduction rules would fire based on the facts that the agent believes as the plan is executed.

Another possible extension of the current target model would remove the requirement that the target agent be single-minded both before and after interacting with the deceiving agent. A model of the development of planning goals could be added to the target model. These planning goals would be triggered by motivation rules, that, like deduction rules, would trigger based on the agent's current beliefs. The effect of triggering a motivation rule would be the consideration of another goal. In this case, the deception planner might choose to purposefully trigger the motivation rule that is most likely to cause the target to act in a way the is beneficial to the deceiver.

Where it is useful to lie about the facts of the world, it is often also be useful to lie about the effects or preconditions of actions. A model of the target's understanding of the effects and preconditions of actions could create interesting additional constraints and solutions for the deception planner. However, doing so effectively creates another modelling challenge – when is it acceptable to lie about the effects of an action, and when is such a deception not credible?

The deception planner is presumed to have prescreened the possible lies that the deceiver can tell, and put as knowledge or something that the target will immediately observe those facts that the deceiver cannot lie about. However, there is a potential for a "lie detection" capability of the target agent. Such an examination of the potential for lying would imply that the target agent has a model of the deceiving agent [19].

6.2 Improvements to the Deceiver Agent Model

The ways to solve a deception problem would be increased by allowing the the deceiving agent to act before speaking with the target agent. For example, imagine a deceiving agent slipping poison into a cup before giving it to a target agent to give to a king. In this way, the deceiving agent might avoid a direct lie while still manipulating the situation to his advantage.

Also, the deception implies action by the deceiver upon the achievement of the ulterior goals, but this action is not within the scope of the deception planner. The planning done by the deceiving agent could be modeled as well. For example, if the serial killer wants to kill Danielle in a private place, then that plan could be modelled as a reason, giving the input to the current module which just wants Danielle to get the house.

6.3 Verification and Improvement of the Deception Planning Algorithm

Running a test of the validity of this algorithm is difficult for multiple reasons. Firstly, the situation it deals with is hypothetical: there is no group of target agents to deceive. Thus, testing the algorithm's efficacy on real agents is not a possibility.

Secondly, any attempt to convert a current example planning problem used in the standard planning literature to a problem acceptable for deception planning requires a significant amount of specialization during the conversion process. The development of observation rules, target knowledge and belief sets, and ulterior goals are all interrelated and the way they are connected directly affects the likelihood of creating a deception plan.

Thirdly, it is not clear that the successful creation of a deception plan is in any way equivalent to the successful creation of a lie that would convince a more complex agent, such as a human. Any complete verification method should at least acknowledge the importance of creating a set of lies that seem plausible from an external perspective.

We believe that even without verification, the algorithm and model are sufficiently new in purpose and technique to merit interest and further research. As research progresses and different techniques are compared and algorithmic improvements are suggested, a method of empirical testing and evaluation will become critical. Although we haven't implemented them, we propose the following future experiments to help validate of the deception planner model.

1. Efficiency and effectiveness can be judged through creating a randomized test suite of deception planner examples. These examples could vary the ratio of deception plan solutions to competing plans, number of poor (long) candidate plans to shorter ones, size of plan search space, the knowledge held by the target agent, the number of ulterior goals, and so on. The purpose of such tests would be to expose the limits of the deception planner. Hand crafted cases meant to fool the deception planner's heuristics might show interesting results as well.

2. The value of the set of lies created by the deception planner could be determined by comparing its output to a human subject's. A study participant could be given the role of the deception planner by coming up with a set of manipulative lies to take advantage of a situation. The deception planner could be given a formal representation of that same situation. Their answers could be compared for the number and type of information that is given. Compare the performance of the deception planner to the human.

3. Another way the deception planner could be evaluated is by giving participants the explicit role of evaluating the deception planner. A participant would be informed of the situation the deception planner is planning for and the statements it came up with. The participant could evaluate whether they thought the statements were going to effectively cause the target agent to act as desired.

6.4 Integration of the deception planner in a larger project

In this thesis, we assume that there is some larger system that keeps track of the target agent, and posits the model of the target agent used by the deception planner. Similarly, we assume there is a system that determines in general whether a target agent will believe a lie about a fact — if not, then that piece of knowledge is shown as target agent knowledge instead of belief. De Rosis suggests heuristics for determining the advisability of a lie[7].

Beyond the direct integration of the deception planner within an autonomous agent, there are several potential ways to integrate the deception planner within a larger system that interacts with agents. As mentioned in the introduction, deception is a useful paradigm for considering narrative. A deception planner could have as an ulterior goal to cause two agents to meet, with the understanding that putting these two agents together would trigger further action. Instead of lies, the deception planner would actually control the state of the virtual world in order to enable or alter the targets' plans to coincide. Such a venture implies adding a new level of complexity to an NP-hard problem. One might posit that it is the seeming impossibility of such projects that makes them alluring.

Chapter 7

Conclusion

We set out in this thesis to create an algorithm, called the deception planner, that creates a set of statements that can be told to a target agent in order to convince that agent to perform particular actions. The presumed situation is one in which a deceiver is giving advice to a target agent, which the target agent will believe and act on as long as it fits within the constraints on the target's previous knowledge.

The deception planner is implemented as a modified planning algorithm which searches for plans that pass through a desired set of states, called the ulterior goals, on its way to achieving the target's planning goals. The deception planner eases the search for these plans by allowing the manipulation of the initial state through adding lies, although the use of these lies is mitigated by a consideration of the target agent's knowledge and observation capabilities. After finding a plan which achieves the ulterior goals on its way to achieving the planning goals, the deception planner ensures that this plan is the one found by the target agent by countering any other plans that the target might develop.

Although the deception planner fits within a body of work on agent deception, the goal of this algorithm is unique in that it focuses on causing a target agent to act through an understanding and exploitation of the mental state of the agent. The deception planner does not lead itself to a deep comparison because of its unique focus. If the measure of success is limited to succeeding in achieving its goals, the algorithm is successful in creating the desired output in a significant number of cases. It is expected that with refinement of the deception planner heuristics, the ratio of successes to attempts can be increased. Since deception planning is an NP-hard problem, however, it is not expected that the deception planner should ever be successful in finding a solution to every deception planning problem.

In order to increase its usefulness, the deception planner was built with modularity in mind. The world it depends on is represented in first-order logic, as dictated by the Planning Domain Definition Language (PDDL), the current standard for representing planning domains[11]. Its output is also in first-order logic. The algorithm would fit within a system built for communicating lies effectively (like the MOUTH-OF-TRUTH), or within a system for generic discourse. In the long run, these larger communication systems may be fit within models of computer agency.

The deception planner is the solution to a formalized version of a problem – how can an agent cause another agent to perform an action through deception? This problem has applications to the future of computer agents and artificial intelligence. By considering the meta-planning necessary for manipulating another agent, an agent could cause a thief to misstep, or help pull off a surprise party. Beyond these capabilities, deception can be a part of negotiating, teasing, joke telling, and story telling. In fact, deception is a tool used by humans every day, and is deeply embedded in our art and culture. Despite its negative connotations, deception is a critical part of humanity, and may someday be used by computer agents for our benefit.

Bibliography

[1] Anthony Barrett and Daniel S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, May 1994.

[2] N. H. Bowling, R. M Jensen, and M. M. Veloso. *Intelligent Planning*, chapter Multiagent Planning in the Presence of Multiple Goals. Wiley, To appear.

[3] Valeria Carofiglio, Fiorella de Rosis, and Cristiano Castelfranchi. Ascribing and weighting beliefs in deceptive information exchanges. *Lecture Notes in Computer Science*, 2109:222–??, 2001.

[4] C. Castelfranchi. Artificial liars: Why computers will (necessarily) deceive us and each other. *Ethics and Information Technology 2*, 2000.

[5] C. Castelfranchi, R. Faclone, and F. de Rosis. *Deceiving in GOLEM: How to Strategically Pilfer Help*, chapter 4. Kluwer, 2001.

[6] R. Dale and E. Reiter. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 19:233–263, 1995.

[7] F. de Rosis, V. Carofiglio, G. Grassano, and C. Castelfranchi. Can computers deliberately deceive? A simulation tool and its application to turing's imitation game. *Computational Intelligence*, 2003.

[8] Bella M. Depaulo and Katherine L. Bell. Truth and investment: Lies are told to those who care. *Journal of Personality and Social Psychology*, 1996.

[9] Toby Donaldson. A position paper on collaborative deceit. In *AAAI-94 Workshop on Planning for Interagent Communication*, 1994.

[10] R. Epstein. The quest for the thinking computer. *AI Magazine*, 13(2):303–319.

[11] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

[12] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *to appear in Journal of Artificial Intelligence Research (JAIR)*, 2003.

[13] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs. In *Proceedings of the Sixth Int. Conference on AI Planning and Sceduling (AIPS'02)*, pages 13–22.

[14] I. J. Good. A causal calculus. *British Journal of the Philosophy of Science*, 11:305–318.

[15] H. Grice. *Logic and Conversation*. 1975.

[16] J. Hoffman and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

[17] Edward Jayne. *Negative Poetics*. University of Iowa Press, Chicago, 1992.

[18] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167–238, 1995.

[19] Mark Lee. The ethics of deception: Why ai must study selfish behavior. *Procs. AISB'00 Symposium on AI, Ethics and (Quasi-)Human rights*, 2000.

[20] J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, New Jersy.

[21] Ayse Pinar Saygin and Ilyas Cicekli. Pragmatics in human-computer conversations. *Journal of Pragmatics*, 34(3):227–258, 2002.

[22] Alan Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460.

Appendix A

LPG Plan Evaluation Heuristic

Below is a (still incomplete) description of the LPG algorithm. Among other capabilities that are left out of the following heuristic, LPG algorithm manages temporal planning. See their papers for more details [12]. All of these figures are very close (where they are not exactly the same) to those in [13]. The descriptions of the functions are my own.

A is the set of actions that make up the current plan.

$pre(a, A)$ = the unsupported preconditions of a in A

$me(a, A)$ = the mutexes of a with other actions in A

A $no-op$ is an action that has its sole precondition and effect some fact f. They are used to carry forward the effects of actions through time in some representations of planning, including LPG's.

And $E(a, A)^i = \lambda_p^a * pre(a, A) + \lambda_m^a * me(a, A)$

$E(a, A)$ is a very rough estimate of the cost of adding an action – it counts the number of preconditions of a that are not supported in A plus the number of mutexes caused by adding a to A.

λ_p^a and λ_m^a are stochastic weights for the preconditions and mutexes associated with each action. They are updated every time the heuristic planner gets to a neighborhood in which all choices are worse than the current plan. If an action has an open precondition when this local minimum is reached, then it is considered "hard", and is given more weight. Similarly, if the action has no open preconditions, then its weight is decreased.

$a_f = MIN_{a' \in A_f}\{E(a', A)^i\}$

a_f is the best choice of action for filling a precondition, based on the value of E.

$$H(f, A) = \begin{cases} 0 & \text{if } f \text{ is supported} \\ H(f', A) & \text{if } a_f \text{ is a no-op with precond. } f' \\ SUM_{f' \in pre(a_f)} H(f', A) + me(a_f, A) + 1 & \text{otherwise} \end{cases}$$

$H(f, A)$ is the cost of supporting some fact. In the generic case, the cost is 1 plus the cost of any mutexes created by adding this action, plus the cost of supporting its preconditions. The actions chosen to support the preconditions are chosen using $E(a, A)$.

$E_H(a, A) = \lambda_p^a * SUM_{f \in pre(a)} H(f, A) + \lambda_m^a * me(a, A)$

$E_H(a, A)$ is a more nuanced cost of supporting an action. It is calculated as the cost of supporting the open preconditions of a (using H) plus the cost of supporting the mutexes caused by adding a.

The final heuristic $F(a, A)$ also counts the number of plan steps in the relaxed plan created to solve the partial plan $A + a$. We call this number $Steps(f, A)$.

$\kappa = \sum_{a \in A} pre(a, A) + me(a, A)$

$F(a, A) = \frac{1}{2 * max_S * \kappa} * (1 + MAX_{f \in pre(a)} MAX(steps(f, A))) +$
$\frac{1}{max_E} * E_H(a, A)$

$F(a, A)$ is the actual value for ranking a neighbor which adds a to plan A.

max_E is the maximum E_H value for all plans in the plan neighborhood. max_S is the maximum number of steps added using relaxed planning by any plan in the neighborhood. Both values are used to average out the weight of these components so that they are relatively equal in weight. Note that in the actual implementation, instead of dividing the step value by max_S, the heuristic value is multipled by max_S. This has the same overall effect, except that the planner avoids very small values. This multiplication explains why, even after these numbers are, according to the algorithm, reduced to being between 0 and 1, the heuristic values seen in Appendix B and C are larger than 1.

Not mentioned in this Appendix is the values of the LPG heuristic when removing actions. The details are similar, but not important to the workings of the deception planner as given in this thesis. The values of the utlerior and lie costs are calculated in exactly the same way when removing an action as when adding it.

Appendix B

Input For The Deception Planner

```
### Input File 1: Plan Domain
### This file describes the domain for the example used in Chapter 5.
(define (domain test-domain)
    (:requirements :strips :equality :typing)
    (:types location - object
             holdable agent - movable )
    (:predicates
     (at ?obj - movable ?loc - location)
     (has ?agent - agent ?obj - holdable )
     (connected ?loc - location ?loc2 - location ))
    (:observation-rules
     (obs1
      :parameters (?obj - holdable ?loc - location )
      :trigger ()
      :observe
      (and (has target ?obj) (at target ?loc)))
     (obs2
      :parameters (?loc - location)
      :trigger (at target ?loc)
      :observe (and (forall (?loc2 - location)
                      (and (connected ?loc ?loc2) (connected ?loc2 ?loc)))
                    (forall (?obj - movable)
                        (at ?obj ?loc)))))
```

```
(:action move
 :parameters (?a - agent ?from ?to  - location )
 :precondition (and (at ?a ?from) (connected ?from ?to))
 :effect (and (at ?a ?to) (not (at ?a ?from))))
(:action take
 :parameters (?a - agent ?obj - holdable ?loc - location)
 :precondition (and (at ?a ?loc) (at ?obj ?loc))
 :effect
 (and (has ?a ?obj) (not (at ?obj ?loc)))
))
```

```
#####################################################
## Input File 2
## This input file describes the specific deception
## planner problem within the described domain
#####################################################
(define (problem test)
 (:domain test-domain)
 (:objects target - agent key - holdable
start-room room-a room-b goal-room ug-room - location)
 (:init  (at target start-room)
   (at key room-a)
   (connected start-room room-a)
   (connected room-a start-room)
   (connected room-a ug-room)
   (connected ug-room room-a)
   (connected room-a room-b)
   (connected room-b room-a)
(connected room-b goal-room)
(connected goal-room room-b))
 (:believes
(connected start-room room-a)
   (connected room-a start-room)
   (connected room-a ug-room)
   (connected ug-room room-a)
   (connected room-a room-b)
   (connected room-b room-a)
(connected room-b goal-room)
(connected goal-room room-b))
 (:goal (and (has target key) (at target goal-room)))
 (:ulterior-goal (and (has target key) (at target ug-room)))
)
```

Appendix C

Output From The Deception Planner

Below is the output given from the deception planner when running the program with the inputs shown in appendix B (edited for clarity).

The formula used to calculate the cost is Scaling Ratio * (Steps Needed) + ActWeight, where StepsNeeded is the number of steps added in relaxed planning (counting a lie as a step), and ActWeight is OpenPreconds + CostToFixOpenPreconds + CostToFix-Mutexes + CostToUltGoals + CostToFixLies. The scaling ratio is the maximum ActWeight over the maximum StepsNeeded. This formula is repeated when it is calculated.

```
Parsing domain file:  domain 'TEST-DOMAIN' defined ... done.
Parsing problem file:  problem 'TEST' defined ... done.


***Creating lies...
Number of actions   :      50
Number of facts     :      41
Evaluation function weights:
     Action duration 0.00; Action cost 0.50

Temporal flag: OFF

Computing mutex... done

Preprocessing total time: 0.00 seconds
```

```
Searching ('.' = every 50 search steps):

Considering Adding Action (MOVE TARGET ROOM-A GOAL-ROOM) at level 3
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET ROOM-B GOAL-ROOM) at level 3
Precondition Cost: 1.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET START-ROOM GOAL-ROOM) at level 3
Precondition Cost: 1.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET UG-ROOM GOAL-ROOM) at level 3
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 0.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)


****** Actual Choosing Done Here:
(MOVE TARGET UG-ROOM GOAL-ROOM)
Value = (7.00/4.00) * (4.00/2) + 4.00 =   7.50

(MOVE TARGET START-ROOM GOAL-ROOM)
Value = (7.00/4.00) * (3.00/2) + 7.00 =   9.62

(MOVE TARGET ROOM-B GOAL-ROOM)
Value = (7.00/4.00) * (3.00/2) + 7.00 =   9.62

(MOVE TARGET ROOM-A GOAL-ROOM)
Value = (7.00/4.00) * (3.00/2) + 7.00 =   9.62

>>> Chose option #0, (MOVE TARGET UG-ROOM GOAL-ROOM) at level 3
```

```
Considering Adding Action (MOVE TARGET ROOM-B UG-ROOM) at level 3
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET ROOM-A UG-ROOM) at level 3
Precondition Cost: 1.00 (weight = 1.00)
Ultgoal cost: 0.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET START-ROOM UG-ROOM) at level 3
Precondition Cost: 1.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET GOAL-ROOM UG-ROOM) at level 3
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Removing Action (MOVE TARGET UG-ROOM GOAL-ROOM) at level 3
Ultgoal cost: 2.00
Lie Cost: 0.00
Add effect cost: 20000000.00 (weight 1.00)


****** Actual Choosing Done Here:
REMOVE (MOVE TARGET UG-ROOM GOAL-ROOM)
Value = (9.00/7.50) * (0.00/2) + 10000003.00 =   10000003.00

(MOVE TARGET GOAL-ROOM UG-ROOM)
Value = (9.00/7.50) * (5.00/2) + 9.00 =   12.00

(MOVE TARGET START-ROOM UG-ROOM)
Value = (9.00/7.50) * (3.00/2) + 7.00 =   8.80

(MOVE TARGET ROOM-A UG-ROOM)
Value = (9.00/7.50) * (2.00/2) + 2.00 =   3.20
```

```
(MOVE TARGET ROOM-B UG-ROOM)
Value = (9.00/7.50) * (4.00/2) + 8.00 =   10.40

>>> Chose option #3, (MOVE TARGET ROOM-A UG-ROOM) at level 3


Considering Adding Action (MOVE TARGET UG-ROOM ROOM-A) at level 2
Precondition Cost: 1.00 (weight = 1.00)
Ultgoal cost: 0.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET START-ROOM ROOM-A) at level 2
Precondition Cost: 0.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET ROOM-B ROOM-A) at level 2
Precondition Cost: 1.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET GOAL-ROOM ROOM-A) at level 2
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Removing Action (MOVE TARGET ROOM-A UG-ROOM) at level 2
Ultgoal cost: 0.00
Lie Cost: 0.00
Add effect cost: 20000000.00 (weight 1.00)


****** Actual Choosing Done Here:
REMOVE (MOVE TARGET ROOM-A UG-ROOM)
Value = (5.00/1.50) * (0.00/2) + 10000001.00 =   10000001.00

(MOVE TARGET GOAL-ROOM ROOM-A)
Value = (5.00/1.50) * (10000001.00/2) + 10000008.00 =   26666676.00
```

```
(MOVE TARGET ROOM-B ROOM-A)
Value = (5.00/1.50) * (10000001.00/2) + 10000006.00 =   26666674.00

(MOVE TARGET START-ROOM ROOM-A)
Value = (5.00/1.50) * (1.00/2) + 5.00 =   6.67

(MOVE TARGET UG-ROOM ROOM-A)
Value = (5.00/1.50) * (10000001.00/2) + 10000002.00 =   26666670.00

>>> Chose option #3, (MOVE TARGET START-ROOM ROOM-A) at level 2


Considering Adding Action (#lie 1:CONNECTED UG-ROOM GOAL-ROOM) at level 3
Ultgoal cost: 0.00
Lie Cost: 0.00

Considering Adding Action (#lie 1:CONNECTED UG-ROOM GOAL-ROOM) at level 1
Ultgoal cost: 1.00
Lie Cost: 0.00

Considering Adding Action (#lie 1:CONNECTED UG-ROOM GOAL-ROOM) at level 2
Ultgoal cost: 1.00
Lie Cost: 0.00

Considering Removing Action (MOVE TARGET UG-ROOM GOAL-ROOM) at level 3
Ultgoal cost: 0.00
Lie Cost: 0.00
Add effect cost: 20000000.00 (weight 1.00)


****** Actual Choosing Done Here:
REMOVE (MOVE TARGET UG-ROOM GOAL-ROOM)
Value = (3.00/1.00) * (0.00/2) + 10000001.00 =   10000001.00

(#lie 1:CONNECTED UG-ROOM GOAL-ROOM)
Value = (3.00/1.00) * (1.00/2) + 3.00 =   4.50

(#lie 1:CONNECTED UG-ROOM GOAL-ROOM)
Value = (3.00/1.00) * (1.00/2) + 3.00 =   4.50

(#lie 1:CONNECTED UG-ROOM GOAL-ROOM)
Value = (3.00/1.00) * (1.00/2) + 1.00 =   2.50

>>> Chose option #3, (#lie 1:CONNECTED UG-ROOM GOAL-ROOM) at level 3
```

```
Considering Adding Action (TAKE TARGET KEY ROOM-A) at level 4
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 2.00 (weight 1.00)

Considering Adding Action (TAKE TARGET KEY ROOM-A) at level 2
Precondition Cost: 0.00 (weight = 1.00)
Ultgoal cost: 0.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (TAKE TARGET KEY START-ROOM) at level 2
Precondition Cost: 3.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 2.00 (weight 1.00)

Considering Adding Action (TAKE TARGET KEY START-ROOM) at level 3
Precondition Cost: 3.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 2.00 (weight 1.00)

Considering Adding Action (TAKE TARGET KEY ROOM-B) at level 2
Precondition Cost: 3.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 2.00 (weight 1.00)

Considering Adding Action (TAKE TARGET KEY ROOM-B) at level 4
Precondition Cost: 3.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 2.00 (weight 1.00)

Considering Adding Action (TAKE TARGET KEY GOAL-ROOM) at level 4
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 2.00 (weight 1.00)
```

```
Considering Adding Action (TAKE TARGET KEY UG-ROOM) at level 3
Precondition Cost: 2.00 (weight = 1.00)
Ultgoal cost: 2147483648.00
Lie Cost: 0.00
Mutex Cost: 0.00 (weight 1.00)


****** Actual Choosing Done Here:
(TAKE TARGET KEY UG-ROOM)
Value = (1.00/2.00) * (3.00/2) + 4294967296.00 =   4294967296.00

(TAKE TARGET KEY GOAL-ROOM)
Value = (1.00/2.00) * (3.00/2) + 4294967296.00 =   4294967296.00

(TAKE TARGET KEY ROOM-B)
Value = (1.00/2.00) * (3.00/2) + 4294967296.00 =   4294967296.00

(TAKE TARGET KEY ROOM-B)
Value = (1.00/2.00) * (3.00/2) + 4294967296.00 =   4294967296.00

(TAKE TARGET KEY START-ROOM)
Value = (1.00/2.00) * (4.00/2) + 4294967296.00 =   4294967296.00

(TAKE TARGET KEY START-ROOM)
Value = (1.00/2.00) * (3.00/2) + 4294967296.00 =   4294967296.00

(TAKE TARGET KEY ROOM-A)
Value = (1.00/2.00) * (1.00/2) + 1.00 =   1.25

(TAKE TARGET KEY ROOM-A)
Value = (1.00/2.00) * (4.00/2) + 4294967296.00 =   4294967296.00

>>> Chose option #6, (TAKE TARGET KEY ROOM-A) at level 2

 solution found:

Last plan computed:
   Time: (ACTION) [action Duration; action Cost]
 0.000: (MOVE TARGET START-ROOM ROOM-A) [D:1.000; C:1.000]
 1.000: (TAKE TARGET KEY ROOM-A) [D:1.000; C:1.000]
 2.000: (MOVE TARGET ROOM-A UG-ROOM) [D:1.000; C:1.000]
 3.000: (MOVE TARGET UG-ROOM GOAL-ROOM) [D:1.000; C:1.000]
```

```
*** Solution DOES achieve all ultgoals

*** Solution DOES lie successfully

Solution number: 1
Total time:      0.02
Search time:     0.02
Actions:         4
Execution cost:  4.00
Duration:        4.000
Plan quality:    2.000
     Plan file:       plan_test7_1.SOL


Available Lies: (we can negate these facts):
CONNECTED(ROOM-B GOAL-ROOM)
CONNECTED(GOAL-ROOM ROOM-B)

Competing static initial state:
AT(TARGET START-ROOM)
AT(KEY ROOM-A)
CONNECTED(ROOM-A UG-ROOM)
CONNECTED(START-ROOM ROOM-A)
CONNECTED(UG-ROOM GOAL-ROOM)
CONNECTED(ROOM-A START-ROOM)
CONNECTED(UG-ROOM ROOM-A)
CONNECTED(ROOM-A ROOM-B)
CONNECTED(ROOM-B ROOM-A)
CONNECTED(ROOM-B GOAL-ROOM)
CONNECTED(GOAL-ROOM ROOM-B)
Number of actions   :      50
Number of facts     :      41

Searching ('.' = every 50 search steps):
>>> Chose option #0, (MOVE TARGET ROOM-B GOAL-ROOM) at level 6


Considering Adding Action (MOVE TARGET GOAL-ROOM ROOM-B) at level 6
Precondition Cost: 1.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET ROOM-A ROOM-B) at level 6
Precondition Cost: 1.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)
```

```
Considering Removing Action (MOVE TARGET ROOM-B GOAL-ROOM) at level 6
Add effect cost: 20000000.00 (weight 1.00)


****** Actual Choosing Done Here:
REMOVE (MOVE TARGET ROOM-B GOAL-ROOM)
Value = (4.00/1.00) * (0.00/2) + 10000001.00 =   10000001.00

(MOVE TARGET ROOM-A ROOM-B)
Value = (4.00/1.00) * (0.00/2) + 2.00 =   2.00

(MOVE TARGET GOAL-ROOM ROOM-B)
Value = (4.00/1.00) * (0.00/2) + 4.00 =   4.00

>>> Chose option #1, (MOVE TARGET ROOM-A ROOM-B) at level 6


Considering Adding Action (MOVE TARGET ROOM-B ROOM-A) at level 5
Precondition Cost: 1.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET START-ROOM ROOM-A) at level 5
Precondition Cost: 0.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET UG-ROOM ROOM-A) at level 5
Precondition Cost: 1.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Removing Action (MOVE TARGET ROOM-A ROOM-B) at level 5
Add effect cost: 20000000.00 (weight 1.00)


****** Actual Choosing Done Here:
REMOVE (MOVE TARGET ROOM-A ROOM-B)
Value = (1.00/1.00) * (0.00/2) + 10000001.00 =   10000001.00

(MOVE TARGET UG-ROOM ROOM-A)
Value = (1.00/1.00) * (10000000.00/2) + 10000002.00 =   15000002.00

(MOVE TARGET START-ROOM ROOM-A)
Value = (1.00/1.00) * (0.00/2) + 1.00 =   1.00

(MOVE TARGET ROOM-B ROOM-A)
Value = (1.00/1.00) * (10000000.00/2) + 10000002.00 =   15000002.00
```

```
>>> Chose option #2, (MOVE TARGET START-ROOM ROOM-A) at level 5

>>> Chose option #0, (TAKE TARGET KEY ROOM-A) at level 5

 solution found:

Last plan computed:
   Time: (ACTION) [action Duration; action Cost]
 0.000: (MOVE TARGET START-ROOM ROOM-A) [D:1.000; C:0.000]
 1.000: (TAKE TARGET KEY ROOM-A) [D:1.000; C:0.000]
 2.000: (MOVE TARGET ROOM-A ROOM-B) [D:1.000; C:0.000]
 3.000: (MOVE TARGET ROOM-B GOAL-ROOM) [D:1.000; C:0.000]

!!! Solution does NOT achieve all ultgoals

*** Solution DOES lie successfully

Solution number: 1
Total time:      0.03
Search time:     0.01
Actions:         4
Execution cost:  0.00
Duration:        4.000
Plan quality:    0.000
     Plan file:       plan_test7_1.SOL

*****

Found competing plan!
Length: 4
Achieves ultgoals: NO -- finding lie to tell for this plan

******
Looking for disabling lie at level 7...Found lie!
 Found lie to add to the mix.  Restarting
 Lie = CONNECTED(ROOM-B GOAL-ROOM)
Removing CONNECTED(ROOM-B GOAL-ROOM) from initial state.
Found fact, removing...
Updated competing initial state:
AT(TARGET START-ROOM)
AT(KEY ROOM-A)
CONNECTED(ROOM-A UG-ROOM)
CONNECTED(START-ROOM ROOM-A)
CONNECTED(UG-ROOM GOAL-ROOM)
```

```
CONNECTED(ROOM-A START-ROOM)
CONNECTED(UG-ROOM ROOM-A)
CONNECTED(ROOM-A ROOM-B)
CONNECTED(ROOM-B ROOM-A)
CONNECTED(GOAL-ROOM ROOM-B)
Number of actions   :      50
Number of facts     :      41

Searching ('.' = every 50 search steps):
>>> Chose option #0, (TAKE TARGET KEY ROOM-A) at level 7


Considering Adding Action (MOVE TARGET START-ROOM ROOM-A) at level 7
Precondition Cost: 0.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET ROOM-B ROOM-A) at level 7
Precondition Cost: 1.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Adding Action (MOVE TARGET UG-ROOM ROOM-A) at level 7
Precondition Cost: 1.00 (weight = 1.00)
Mutex Cost: 0.00 (weight 1.00)

Considering Removing Action (TAKE TARGET KEY ROOM-A) at level 7
Add effect cost: 20000000.00 (weight 1.00)


****** Actual Choosing Done Here:
REMOVE (TAKE TARGET KEY ROOM-A)
Value = (1.00/1.00) * (0.00/2) + 10000001.00 =   10000001.00

(MOVE TARGET UG-ROOM ROOM-A)
Value = (1.00/1.00) * (10000000.00/2) + 10000002.00 =   15000002.00

(MOVE TARGET ROOM-B ROOM-A)
Value = (1.00/1.00) * (10000000.00/2) + 10000002.00 =   15000002.00

(MOVE TARGET START-ROOM ROOM-A)
Value = (1.00/1.00) * (0.00/2) + 1.00 =   1.00
```

```
>>> Chose option #3, (MOVE TARGET START-ROOM ROOM-A) at level 7

.......... search limit exceeded.No competing plan found!
Current solution is successful.
Facts (* = lies) to tell to target:
AT(TARGET START-ROOM)
AT(KEY ROOM-A)
CONNECTED(ROOM-A UG-ROOM)
CONNECTED(START-ROOM ROOM-A)
 * CONNECTED(UG-ROOM GOAL-ROOM)
 * NOT(CONNECTED(ROOM-B GOAL-ROOM))
```