8

Modeling and Analysis of Artificially Intelligent Planning Systems

K.M. Passino
Dept. of Electrical Engineering
The Ohio State University
2015 Neil Ave.
Columbus, OH 43210

P.J. Antsaklis
Dept. of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

Abstract

In an Artificial Intelligence (AI) planning system the planner generates a sequence of actions to solve a problem. Similarly, the controller in a control system produces inputs to a dynamical system to solve a problem, namely the problem of ensuring that a system's behavior is a desirable one. A mathematical theory of AI planning systems that operate in uncertain, dynamic, and time critical environments is not nearly as well developed as the mathematical theory of systems and control. In this Chapter relationships, and a detailed analogy between AI planning and control system architectures and concepts are developed and discussed. Then, we illustrate how a discrete event system (DES) theoretic framework can be used for the modeling and analysis of AI planning systems. The ideas presented are fundamental to the development of a mathematical theory for the modeling, analysis, and design of AI planning systems for real time environments. It is hoped that the ideas presented here will help to form a foundation for the implementation and verification of AI planning systems.

1. INTRODUCTION

In an Artificial Intelligence (AI) planning system the planner generates a sequence of actions to solve a problem. It is a type of expert system since it emulates how human experts represent, and reason over, abstract uncertain information to solve a problem in a narrow field of expertise. As it is explained below planning systems are, however, specially equipped to be able to use feedback information and cope with uncertainties in real time environments. While several essential ideas in the theory of AI planning on emulation of human planning activities and planning techniques have been developed and reported in the AI literature little attention has been given to engineering aspects of the planning problem. For instance, it is of fundamental importance to study implementation issues and to create a mathematical theory of AI planning systems that operate in dynamic, uncertain, and time-critical environments (real time environments) to lay the foundation for verifying and certifying their dynamical behavior. Without such careful verification of the planning system functions and a full understanding of the implementation issues it is doubtful that they will be trusted in many real-world applications [1].

In a control system the controller produces inputs to a dynamical system to ensure that it behaves in a desirable manner (e.g., to achieve disturbance rejection). In contrast to AI planning, there exists a relatively well developed mathematical systems and control theory for the study of properties of systems represented with, for instance, linear ordinary differential equations. The objective of this Chapter is to point out relationships and to develop and discuss an extensive analogy between AI planning and control system architectures and concepts. In the process, a foundation of fundamental concepts in AI planning systems based on their control theoretic counterparts is built. The second objective of this paper is to investigate the use of a discrete event system (DES) theoretic framwork for the modeling and analysis of plan generation in AI planning systems. This will show that the DES theoretic framework offers a suitable approach for the modeling and analysis of AI planning systems but also indicate that much more work needs to be done. This Chapter represents a synthesis of some of the results reported in [1-7].

Overall, it is hoped that these discussions and analysis will help lead to the development of (i) A variety of quantitative systematic modeling, analysis, and design techniques for AI planning systems, (ii) Methods to analyze planning system performance, and (iii) Empirical and/or analytical methods for AI planning system verification, validation, and certification.

In Section 2 planning systems are classified, their fundamental operation is explained, and the relevant concepts from AI planning system theory are overviewed. In Section 3 the analogy and relationships between AI planning and control theory are developed. This includes a foundation of fundamental concepts for AI planning systems including controllability/reachability, observability, stability, open and closed loop planning, etc. Section 4 contains a DES model that can represent a wide class of AI planning problems. Section 5 provides two applications of the DES theoretic framework to the modeling and analysis of plan generation in AI planning systems. Section 6 provides some concluding remarks and indicates some important research directions.

2. ARTIFICIAL INTELLIGENCE PLANNING SYSTEMS: CLASSIFICATION, FUNCTIONAL OPERATION, AND OVERVIEW

In this section, the essential components and ideas of AI planning systems are briefly outlined and distinctions are drawn between AI planning systems and other similar problem solving systems.

2.1 System Classification

In general, for this Chapter we will adopt the view that we can classify problem solving systems into two categories: conventional and AI. Several distinct characteristics distinguish these two classes of problem solving systems. The conventional problem solving system is numeric-algorithmic; it is somewhat inflexible; it is based on the well developed theory of algorithms or differential/difference equations; and thus it can be studied using a variety of systematic modeling, analysis, and design techniques. Control systems are an example of a conventional problem solving systems.

An AI problem solving system is a symbolic-decision maker, it is flexible with graceful performance degradation, and it is based on formalisms which are not well developed; actually there are very few methodical modeling, analysis, and design techniques for these systems. AI planning systems are examples of AI problem solving systems. When comparing the characteristics of AI and non-AI systems, one can make the following observations: The decision rate in conventional systems is typically higher than that of AI systems. The abstractness and generality of the models used in AI systems is high compared with the fine granularity of models used in conventional systems. Symbolic representations, rather than numeric, are used in AI systems. High level decision making and learning capabilities similar to those of humans exist in AI systems to a much greater extent than in conventional systems. The result is that a higher degree of autonomy sometimes exists in AI systems than in conventional ones [8,9].

AI planning systems use models for the problem domain called "problem representations". For instance, in the past predicate or temporal logic has been used. A planner's reasoning methodology is modeled after the way a human expert planner would behave. Therefore, they use heuristics to reason under uncertainty. Conventional expert systems have many of the elements of planning. They use similar representations for knowledge and heuristic inference strategies. The planning systems that are studied here, however, are specifically designed to interface to the real world, whereas conventional expert systems often live in a tightly controlled computer environment. The planning system executes actions dynamically to cause changes in the problem domain state. The planner also monitors the problem domain for information that will be useful in deciding a course of action so that the goal state is reached. There is an explicit loop traversed between planner executed actions, the problem domain, the measured outputs, and the planner that uses the outputs to decide what control actions to execute to achieve the goal. In an expert system there exists an analogous loop which has been recently studied in [10]. The knowledge base is the problem domain and the inference strategy is the planner. For rule based expert systems the premises of rules are matched to current working memory (outputs are measured and interpreted), then a heuristic inference strategy decides which rule to fire, that is, what actions to take to change the state of working memory (the knowledge base) and so on. The expert system has an inherent goal to generate some diagnosis, configure some computer system, etc. Some expert systems have more elements of planning than others. For instance, some consider what will happen several steps ahead, if certain actions are taken.

A further distinction must be made between AI planning and scheduling systems. It is the task of a planner to generate sequences of actions so that some goal is attained without expending too many resources (for specific examples see Section 5). A scheduling system is concerned with *when* the actions should be executed and uses information about the availability of resources to assign resources to times and actions.

2.2 Elements of AI Planning Systems

An *AI planning system* consists of the planner, the problem domain, their interconnections, and the exogenous inputs. The outputs of the planner are the inputs to the problem domain. They are the control actions taken on the domain.

The outputs of the problem domain are inputs to the planner. They are measured by the planner and used to determine the progress in the problem solving process. In addition, there are unmeasured exogenous inputs to the problem domain which are called disturbances. They represent uncertainty in the problem domain. The measured exogenous input to the planner is the goal. It is the task of the planner to examine the problem domain outputs, compare them to the goal, and determine what actions to take so that the goal is met. Not all planners are completely autonomous. Some provide for human interface, through which goals may be generated, and allow varying degrees of human intervention in the planning process.

The *problem domain* is the domain (environment) the planner reasons about and takes actions on. The problem domain is composed of a collection of *problems* that the planner desires to solve. The planner takes actions on the problem domain via the *inputs* to *solve* a particular problem. The planner measures the effect of these actions via the *outputs* of the problem domain. The *disturbances* represent uncertainty in the problem domain. The *solution* of a problem is composed of the sequence of inputs and outputs (possibly states) generated in achieving the goal.

One develops a *model* of the real problem domain to study planning systems. This is called the *problem representation.* The real problem domain (for real-world applications) is in some sense infinite, that is, no model could ever capture all the information in it. The problem representation is necessarily inaccurate. It may even be inaccurate by choice. This occurs when the planning system designer ignores certain problem domain characteristics in favor of using a simpler representation. Simpler models are desirable, since there is an inversely proportional relationship between modeling complexity and analysis power. The characteristics of the problem domain that are ignored or missed are often collectively represented by disturbances in the model. The result is that disturbances in general have non-deterministic character. Clearly, disturbances occur in every realistic problem domain; they can be ignored when they are small, but their effect should always be studied to avoid erroneous designs.

In this section we shall describe the functional components that may be contained in an AI planner. The AI planner's task is to solve problems. To do so, it coordinates several functions: *Plan generation* is the process of synthesizing a set of candidate plans to achieve the current goal. This can be done for the initial plan or for *replanning* if there is a plan failure. In plan generation, the system *projects* (simulates, with a model of the problem domain) into the future, to determine if a developed plan will succeed. The system then uses heuristic *plan decision rules* based on resource utilization, probability of success, etc., to choose which plan to execute. The *plan executor* translates the chosen plan into physical actions to be taken on the problem domain. It may use scheduling techniques to do so.

Situation assessment uses the problem domain inputs, outputs, and problem representation to determine the state of the domain. The estimated domain state is used to update the state of the model that is used for projection in plan generation. The term "situation" is used due to the abstract, global view of the system's state that is taken here. The term "assessment" is used since the value of the state is determined or estimated. *Execution monitoring* uses the estimated domain state and the problem domain inputs and outputs to determine if everything is going as planned. If it isn't, that is if the plan has failed, the plan generator is notified that it must replan. A *World Modeller* produces an update to, or a completely new world model. The world modeler determines the structure of the problem domain rather than just the state of the problem domain like the situation assessor. It also determines what must be modeled for a problem to be solved; hence it partially determines what may be disturbances in the problem domain. The term "world model" is thus used to indicate that it must be cognizant of the entire modeling process. Its final output is a problem representation. A *Planner Designer* uses the problem representation produced by the world modeler and designs, or makes changes to the planner so that it can achieve its goal even though there are structural changes in the problem domain. The planner designer may not need a new problem representation if there are not structural changes in the problem domain. It may decide to change the planner's strategy because some performance level is not being attained or if certain events occur in the problem domain. Situation assessment, exection monitoring, world modeling, and planner design are all treated in more detail in [1].

2.3 Issues and Techniques in AI Planning Systems

In this section we briefly outline some of the issues and techniques in AI planning systems. A relatively complete summary of planning ideas and an extensive bibliography on planning is given in [11,1]. The goal of this Section is to set the terminology of the Chapter.

Representation is fundamental to all issues and techniques in AI planning. It refers to the methods used to model the problem domain and the planner and it sets the framework and restrictions on the planning system. Often, it amounts to the specification of a formalism for representing the planner and problem domains in special structures in a computer language. Alternatively, it could constitute a mathematical formalism for studying planning problems. Different types of symbolic representations such as finite automata and predicate or temporal logics have been used. Some methods allow for the modeling of different characteristics. Some do not allow for the modeling of non-determinism. One should be very careful in the choice of how much detailed mathematical structure or modeling power is allowed, since too much modeling power can hinder the development of some functional components of the planner, and of the analysis, verification, and validation of planning systems.

The generality of developed planning techniques depends heavily on whether the approach is *domain dependent* or *domain independent.* Techniques developed for one specific problem domain without concern for their applicability to other domains are domain dependent. The ideas in Section 3 of this Chapter are both domain independent and problem representation independent.

Planners can be broadly classified as either being *hierarchical* or *non-hierarchical*. A non-hierarchical planner makes all of its decisions at one level, while in a hierarchical planner there is delegation of duties to lower levels and a layered distribution of decision making. Planners can also be classified as being *linear* or *nonlinear*. A linear planner produces a strict sequence of actions as a plan, while a nonlinear planner produces a partially ordered sequence where coordination between the tasks and subtasks to be executed is carefully considered. There are several types of *interactions* that can occur in planning. One is the interaction between subtasks or subplans that requires their proper coordination. Another is between different goals we might want to achieve. While still another is between different planning modules or with the human interface. *Search* is used in planning systems to, for instance, find a feasible plan. There are many types of search such as the *heuristic search* algorithms called A^* or AO^*. In Sections 4 and 5 we show how A^* search can be used to solve plan generation problems in a DES theoretic framework.

Skeletal plans, *plan schema*, and *scripts* are all representations of plans with varying degrees of abstraction. Skeletal plans are plans that to some extent do not have all the details filled in. A script is a knowledge structure containing a stereotypic sequence of actions. Plan schema are similar. Often planners which use these forms for plans store them in a *plan library*. *Hypothetical planning* is planning where the planner hypothesizes a goal, produces a subsequent plan, and stores it in a plan library for later use all while the current plan is executing.

Replanning is the process by which plans are generated so that the system recovers after a plan has failed. There are two types of plan failures. One occurs in plan generation where the planner fails to generate a plan. In this case, replanning can only be successful if a planner redesigner makes some changes to the planner strategy. The second type of plan failure occurs in the execution of the plan and is due to disturbances in the problem domain. This plan failure can be accommodated for by replanning in the plan generation module, if the failure is not due to a significant structural change in the problem domain. If it is, then the world modeler will produce a new world model and the planner designer will make the appropriate changes to the planner so that it can replan.

Projection is used in plan generation to look into the future so that the feasibility of a candidate plan can be decided. Projection is normally done by performing symbolic simulation with a problem representation. If it is assumed that there are no disturbances in the problem domain, and a plan can be generated, then it can be executed with complete confidence of plan success. Disturbances cannot be ignored in problem domains associated with real world dynamic environments; therefore, complete projection is often futile. The chosen *projection length* (number of steps in symbolic simulation) depends on the type and frequency of disturbances and is therefore domain dependent. Notice that if the projection length is short, plan execution can be interleaved with planning and replanning. This sort of planning has been named *reactive planning*. A completely *proactive planner* always has a plan ready for execution (in a plan library) no matter what the situation is in the problem domain. These could be skeletal or scripts. Some mixture of proactive and reactive planning with varying projection length is probably most appropriate. In *opportunistic planning* one forms a partial plan then begins execution with the hope that during execution opportunities will arise that will allow for the complete specification of the plan and its ultimate success. *Planning with constraints* is a planning methodology where certain constraints on the planning process are set and the planner must ensure that these constraints are not violated. *Distributed planning* occurs when a problem is solved by coordinating the results from several expert planners. It is also called *multi-agent planning*. *Metaplanning* is the process of reasoning about how a planner reasons. It is used with world modeling and changes the planning strategy. Planners can also be made to *learn*. For example, a simple form of learning is to save successful plans in a plan library for later use in the same situation. Again, we emphasize that the full details on the main ideas in planning theory can be found in [11,1].

3. ARTIFICIAL INTELLIGENCE PLANNING AND CONTROL THEORY: ANALOGY AND RELATIONSHIPS

Relationships and an extensive analogy between AI planning and control system architectures and concepts are developed in this section based on the work in [1]. This is possible since both are problem solving systems (as described in Section 2.1), with different problem domains. It is useful to draw the analogy since conventional problem solving systems, such as control systems, are very well studied. They have a well developed set of fundamental concepts and formal mathematical modeling, analysis, and design techniques. The analogy is used to derive a corresponding foundation of fundamental concepts for AI planning systems that can be used to develop modeling, analysis, and design techniques.

The discussions below are meant to motivate the utility of using general systems theory for the study of AI planning systems. In particular, it is hoped that it is made clear that the general concepts of controllability/reachability, observability, stability, etc. as defined in systems and control theory will be useful in the quantitative study of AI planning systems. The results here will probably need to be revised and expanded before a careful formulation of a mathematical theory of AI planning via control theory is possible.

3.1 The Problem Domain / Plant Analogy

In this section we shall give the structural analogy (functional analogy) between the problem domain and the plant. In conventional control, the *plant* is a dynamical system whose behavior we wish to study and alter. It is generally described with linear or nonlinear differential/difference equations, and is either deterministic or non-deterministic. The *problem domain* is the domain (environment) the AI planner reasons about and takes actions on. It can be modeled using predicate or temporal logic, or other symbolic techniques such as finite automata. We develop the analogy further using Figure 3.1.

As it is often done, we adopt the convention that actuators and sensors are part of the plant, and thus part of the problem domain description. Plant *actuators* are hardware devices (transducers) which translate commands u(t) from the controller into actions taken on the physical system. The variable t represents time.

In a problem domain, we take a more macroscopic view of an actuator, a view which depends on available hardware and the type of inputs generated by the planner. For example, in a robotic system a manipulator may be quite dexterous; one may be able to just send the command "pick up object", and it will know how to move to the object, grip it, and pick it up. Such a manipulator can be seen as an actuator, although simpler ones also exist. Clearly, the inputs to the problem domain can be more abstract than those of a plant; consequently, we describe them with symbols u_i rather than numbers. The index i represents time in the problem domain. The symbols are quite general and allow for the representation of all possible actions that <u>any</u> planner can take on the problem domain. For example, u_1="pick up object", or u_2="move manipulator from position 3 to position 7". Rather than an input u(t) for the plant, the problem domain input u_i is a time sequence of symbols.

The *physical system* for both the problem domain and the plant is some portion of the real world which we wish to study and alter. The difference between the two is the type of systems that are normally considered, and thus the modeling techniques that are used (See discussion in Section 2.2). Aspects of the dynamical behavior of plants such as cars, antennas, satellites, or submarines can be modeled by differential equations. Problem domains studied in the AI planning literature include simple robot problems, games, experiments in molecular genetics, or running errands. Notice that problem domains cannot always be described by differential equations. Consequently, certain conventional control techniques are often inappropriate for AI planning problems.

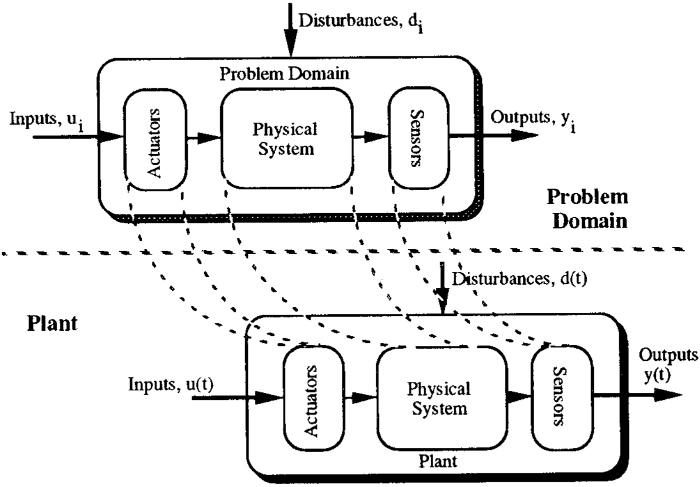

Figure 3.1. Problem Domain / Plant Structure

The *sensors* in the plant and problem domain are used to measure variables of the physical system and translate this information to y(t) for the controller and y_i for the planner. The symbols y_i provide for the representation of all possible measured values of outputs of the problem domain. As with the actuators in the problem domain, we take a more macroscopic view of sensors. They can combine various data to form an aggregate representation of dynamic problem domain information. This necessitates the use of symbolic representations of the measured outputs; consequently, y_i is a time sequence of symbols. For example, in the robot problem the position of some of the objects to be moved could be represented with y_i. The outputs could be y_1="object 1 in position 5" and y_2="object 1 in position 3". The inputs u_i can affect the physical system so that the outputs y_i can change over time.

The *state* of the plant or problem domain (or any dynamical system) is the information necessary to uniquely predict the future behavior of the system given the present and future system inputs. A particular state is a "snapshot" of the system's behavior. The *initial state* is the initial condition on the differential/difference equation that describes the plant, or the initial situation in the problem domain prior to the first time a plan is executed. We shall denote the state of the plant with x(t) and the problem domain with x_i. The set of all possible states is referred to as the "state space". In our robot problem domain, the initial state can be the initial positions of the manipulator and objects. For two objects, the initial state might be x_0="object 1 in position 3 and object 2 in position 7 and manipulator in position 5". Notice that part of the state is directly contained in the output for our example. The state describes the current actuation and sensing situation in addition to the physical system, since the sensors and actuators are considered part of the problem domain.

The plant and problem domain are necessarily affected by *disturbances* d(t) or symbols d_i respectively (See discussion in Section 2.2). These can appear as modeling inaccuracies, parameter variations, or noise in the actuators, physical system, and sensors. In our robotics problem domain a disturbance might be some external, unmodeled agent, who also moves the objects. Next we show how the functional analogy, between the plant and problem domain, extends to a mathematical analogy.

3.2 The Plant / Problem Domain Model Analogy

Due to their strong structural similarities it is not surprising that we can develop an analogy between the models that we use for the plant and problem domain and between fundamental systems concepts. Essentially this involves a discussion of the application of a general systems theory described in [12] (and many stanard control texts) to planning systems. We extract the essential control theoretic ideas, and adapt them to planning theory, without providing lengthy explanations of conventional control theory. The interested reader can find the relevant control theoretic ideas presented below in one of many standard texts on control. For the sake of discussion we assume that we can describe the dynamics of the problem domain by a set of symbolic equations such as those used to describe finite state automata [13]. Alternatively, one could use one of many "logical" DES models (See Section 4 or [14]).

The mathematical analogy continues by studying certain properties of systems that have been found to be of utmost importance in conventional control theory.

Controllability / Reachability

In control theory, and thus in planning theory, *controllability* (or similarly, reachability) refers to the ability of a system's inputs to change the state of the system. It is convenient to consider a deterministic system for the discussion. A sequence of inputs u_i can *transfer* or *steer* a state from one value to another. In the robot example, a sequence of input actions transfers the state from x_0="object 1 in position 3 and object 2 in position 7 and manipulator in position 5" to x_7="object 1 in position 5 and object 2 in position 10 and manipulator in position 1".

A system is said to be *completely controllable* at time i if there exists a finite time j>i such that for any state x_i and any state x there exists an input sequence $u_i, \dots ,u_j$ that will transfer the state x_i to the state x at time j, that is $x_j=x$.

Intuitively, this means that a problem domain is completely controllable at some time if and only if for every two state values in the state space of the problem representation, there exists a finite sequence of inputs (that some planner could produce) which will move the states from one value to the other (one state to the other). Also notice that the time j-i is not necessarily the minimum possible. There might be another sequence of inputs which will bring one state to the other in fewer steps. In the robot example, the problem domain is completely controllable if for any position of the manipulator and objects, there exist actions (inputs) which can change to any other position of the objects and manipulator.

If a problem domain is completely controllable, then for any state there exists a planner that can achieve any specified goal state. Sometimes complete controllability is not a property of the system, but it may possess a weaker form of controllability which we discuss next. To discuss a more realistic, weaker form of controllability we assume that the state space can be partitioned into disjoint sets of controllable and uncontrollable states.

A system is said to be *weakly controllable* at time i if there exists a finite time j>i such that for any states x_i and x, both in the set of controllable states, there exists an input sequence $u_i, \ldots, u_j$ that will transfer the state x_i to the state x at time j, that is $x_j=x$.

If the initial state and the goal state are given and contained in the set of controllable states and the problem representation is weakly controllable, then there exists a planner which can move the initial state to the goal state. That is, there exists a planner which can solve the problem. In the robot example, if the problem representation is weakly controllable and the initial state begins in the set of controllable states, then there are actions (inputs) that can move the manipulator and objects to a certain set of positions in the set of controllable states, the ones one might want to move them to. Note that there are corresponding definitions for *output controllability.* Notions of controllability and reachability applicable to AI planning systems have been studied in the DES literature [15,16,5].

Observability

In control theory, and thus in planning theory, *observability* of the problem domain refers to the ability to determine the state of a system from the inputs, outputs, and model of the system.

A system is said to be *completely observable* at time i if there exists a finite time j>i, such that for any state x_i, the problem representation, the sequence of inputs, and the corresponding sequence of outputs over the time interval [i,j], uniquely determines the state x_i.

Intuitively, this means that a problem domain is completely observable at some time if and only if for every sequence of domain inputs and their corresponding outputs, the model of the domain and the input and output sequences are all that is necessary to determine the state that the domain began in. A problem domain that is completely observable on some long time interval may not be completely observable on a shorter interval. It may take a longer sequence of inputs and outputs to determine the state.

In the robot example, if the problem domain is completely observable, then for every sequence of actions (inputs) there exists a situation assessor that can determine the position of the objects and manipulator from the input sequence, output sequence, and model of the problem domain.

If a problem domain is completely observable, then for any initial state, there exists a situation assessor that can determine the state of the problem domain. This situation assessor needs both the inputs and the outputs of the problem domain, and there is the assumption that there are no disturbances in the domain. Sometimes complete observability is not a property of systems, but they may possess a weaker form of observability which is defined next. To discuss a more realistic, weaker form of observability, we will assume that the state space can be partitioned into disjoint sets of observable and unobservable states.

A system is said to be *weakly observable* at time i if there exists a finite time j>i, such that for any state x_i in the set of observable states, the problem representation, the sequence of inputs, and the corresponding sequence of outputs over the interval [i,j], uniquely determines the state x_i.

If the problem domain is weakly observable there exists a situation assessor that can determine the state of the problem domain given that the system state begins in the set of observable states. In the robot example, if the problem domain is weakly observable, then for any initial observable state and every sequence of actions (inputs) that any planner can produce, there exists a situation assessor that can determine the position of the objects and manipulator from the planner input sequence, output sequence, and model of the problem domain.

As with controllability, observability is a property of systems in general, therefore it has meaning for the problem domain, planner, and the planning system. Notions of observability applicable to AI planning systems have been studied in the DES literature (See the references in [14]).

Stability

In control, and thus in planning theory, we say that a system is *stable* if with no inputs, when the system begins in some particular set of states and the state is perturbed, it will always return to that set of states. For the discussion we partition the state space into disjoint sets of "good" states, and "bad" states. Also we define the *null input* for all problem domains as the input that has no effect on the problem domain. Assume that the input to the system is the null input for all time. A system is said to be *stable* if when it begins in a good state and is perturbed into any other state it will always return to a good state.

To clarify the definition, a specific example is given. Suppose that we have the same robot manipulator described above. Suppose further that the set of positions the manipulator can be in can be broken into two sets, the good positions and the bad positions. A good position might be one in some envelope of its reach, while a bad one might be where it would be dangerously close to some human operator. If such a system were stable, then if when the manipulator was in the good envelope and was bumped by something, then it may become dangerously close to the human operator but it would resituate itself back into the good envelope without any external intervention.

We make the following definitions to make one more definition of stability. We assume that we can partition the set of possible input and output symbols into disjoint sets of "good" and "bad" inputs and outputs. A system is said to be *input-output stable* if for all good input sequences the corresponding output sequences are good.

In the robot example, suppose that the inputs to the manipulator can be broken into two sets, the good ones and the bad ones. A bad input might be one that takes a lot of resources or time to execute or it might be an input that takes some unreasonable action on the problem domain. Let the output of the robot problem domain be the position of the objects that the manipulator is to move. A bad output position would be to have an object obstruct the operation of some other machine or to have the objects stacked so that one would crush the other. The robot problem domain is input-output stable if for all reasonable actions (good inputs) the manipulator is asked to perform, it produces a good positioning of the objects (good outputs) in the problem domain. These stability definitions and ideas also apply to the planner and the planning system. Notions of stability applicable to AI planning systems have been studied in the DES literature [17].

Stabilizability refers to the ability to make a system stable. For a planning system it may, for instance, refer to the ability of any planner to stabilize the problem domain. A system is said to be *stabilizable* if the set of controllable states contains the bad states. For the robot example, the problem domain is stabilizable if for all states that represent bad positions of the manipulator arm, there are inputs that can move the arm to its good (state) operating envelope. Detectability refers to the ability to detect instabilities in a system. For a planning system it may, for instance, refer to the ability of the situation assessor to determine if there are instabilities in the problem domain. A system is said to be *detectable* if the set of observable states contains the bad states. For the robot example, if the problem domain is detectable, then for all input sequences that place the manipulator arm in a bad position, there exists a situation assessor that can determine the state. These definitions also apply to the planner and the planning system.

3.3 Open Loop AI Planning Systems

In this section we define open loop planning systems and outline some of their characteristics. They are named "open loop" because they use no feedback information from the problem domain. Here we develop a structural analogy between open loop conventional control systems and open loop planning systems beginning with Figure 3.2. In conventional control theory, the open loop control system has the structure shown at the bottom of Figure 3.2. The outputs of the controller are connected to the inputs of the plant so that they can change the behavior of the plant. The input to the controller is the *reference input* r(t), and it is what we desire the output of the plant to be. The controller is supposed to select the inputs of the plant u(t) so that $y(t) \rightarrow r(t)$, or $y(t) - r(t)$ is appropriately small for all times greater than some value of t. Specifications on the performance of control systems speak of the quality of the response of y(t). For example, we might want some type of transient response or we may want to reduce the effect of the disturbance on the output y(t). However, an open loop control system cannot reduce the effect of disturbances in any way; notice that, by definition, the disturbances cannot be measured.

In the open loop planner, plan generation is the process of synthesizing a set of candidate plans to achieve the goal at step i which we denote by g_i. The goals g_i may remain fixed, or change in time. In plan generation, the system projects (simulates, with a model of the problem domain) into the future, to determine if a developed plan will succeed. The system then uses heuristic plan decision rules based on resource utilization, probability of success, etc., to choose which plan to execute. The plan executor translates the chosen plan into actions (inputs u_i) to be taken on the problem domain. Many AI planners discussed in the AI literature and implemented to date are open loop planners.

Next, we examine some basic characteristics of AI open loop planning systems. We first consider the characteristics of the planner itself (not connected to the problem domain) by interpreting the results above. Then we outline the characteristics of open loop planning systems. It is useful to consider the planner to be a model of some human expert planner. The state of the planner is the situation describing the planner's problem solving strategy at a particular instant.

Planner controllability refers to the ability of the goal inputs to affect the state of the planner. Planner observability refers to the ability to determine the planner state using the goal inputs, planner outputs u_i, and the model of the planner. Stability of the planner refers to its ability to stay in a problem solving state of mind (ready to solve problems) when there are no goals to achieve (a null input). Input-output stability of the planner is attained if for all reasonable, admissible goals input the planner produces reasonable, acceptable outputs u_i. The planner is stabilizable if there exists a sequence of goals that will keep the planner properly focused on the problem. The planner is detectable if for all goal sequences that cause the planner to loose its focus of attention, the inputs, outputs, and model of the planner can be used to determine where the focus of attention is.

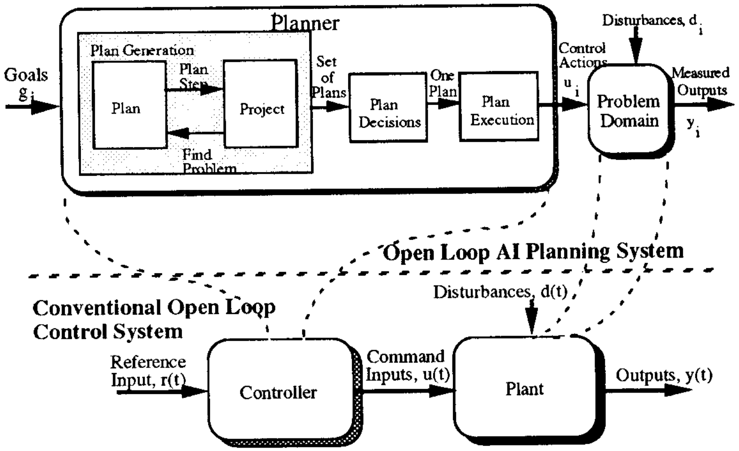

Figure 3.2. Open Loop Structural Analogy

All of the interpretations given for the planner are valid here, the difference being that since we cascade the problem domain we are thinking of solving a particular problem. If the problem domain is uncontrollable, then there may not exist a planner capable of solving the problem. If it is controllable, then a planner does exist. This does not mean that if the problem domain is completely controllable, we can choose just any planner and it will solve the problem. It just says that one exists. Situation assessment and execution monitoring cannot be done since there is no connection to the outputs of the problem domain. Consequently, there cannot be any replanning. If there are any disturbances in the problem domain, the planner may become totally lost in its problem solving process, because it has no ability to recover from plan failure; it is even unaware that there was a failure. We say that the planning system is sensitive to problem domain variations and open loop planners cannot reduce this sensitivity. This is closely related to the idea of sensitivity reduction in conventional control theory. Since, as explained in Section 2.2, there will always be some disturbances in the real world, open loop planners will necessarily fail at their task. However, they can work if the problem domain is well modeled and the disturbances are quite insignificant. This generally requires the use of a very complex, detailed model of the problem domain in the case of significant real world problems. Notice that since the outputs are not sensed, if the problem domain is unstable (input-output or internally), then it is never stabilizable in open loop planning. Open loop stabilization requires absolutely exact knowledge of the problem domain. Since often this cannot be obtained, even insignificant disturbances can be catastrophic.

The length of projection in plan generation can be quite long since, if one is using open loop planning then the disturbances must be assumed non-existent or insignificant. The only reason for making the projection length shorter would be to begin plan execution. If the projection length is too short for the plan generator to specify a set of plans that will work, then there will be uncertainty in the plan execution which may lead to ultimate plan failure. This is why current open loop planners build the complete plan, then execute it.

Open loop planners do have the advantage of simplicity. If the problem domain is stable and disturbances are insignificant, then they should certainly be considered. They are cheaper to implement than the closed loop planners described in the next two sections, since one does not need to buy sensors to gather information about the states and outputs of the problem domain.

3.4 AI Feedback Planning Systems

AI feedback planning systems are analogous to conventional feedback control systems that do not use state estimation; they do not use situation assessment. In [18, Chapter 9] the authors point out the inherent feedback in the planning process. They do not, however, make a clear distinction of the separation between the planner and problem domain and their interconnections. In this section the distinctions will be clarified.

The analogy between the feedback structures emerges from Figure 3.3. The structure is the same as for the open loop system except there is the feedback connection. This allows the planner to perform execution monitoring and replanning. The feedback planning system can recover from plan failures that occurred due to significant disturbances in the problem domain. The execution monitoring system uses the measured outputs, the inputs, and domain model to determine if the current plan has failed. If a plan fails, it informs the plan generator that it must replan.

If the problem domain is not completely controllable, then one can perhaps use more elaborate actuation systems that will properly affect the state of the domain; or perhaps rederive the model of the problem domain since controllability is a property of the mathematical model used. Therefore, controllability studies can be used for design guidelines for the problem domain. Likewise for observability. Situation assessment is not needed in a planner, if the full state of the problem domain is measurable. This is analogous to full state feedback in conventional control theory. If observability studies show that some states of the domain are unobservable, then one can design and implement additional sensors which can provide the necessary information about the state. We see that there is a tradeoff between expense of implementation of a planning system and planner complexity. It may be expensive to implement sensors to sense the whole state, but then situation assessment is not necessary thus making the planner simpler.

In order for the controller to force the plant output to *track* or *follow* the reference input, it compares the output to the current reference input, and decides what to input into the plant. In comparing r(t) and y(t), the controller simply uses the difference r(t)-y(t) to determine how well it is meeting its objective at any instant and it takes appropriate actions. The difference r(t)-y(t), called the *error*, is a control measure.

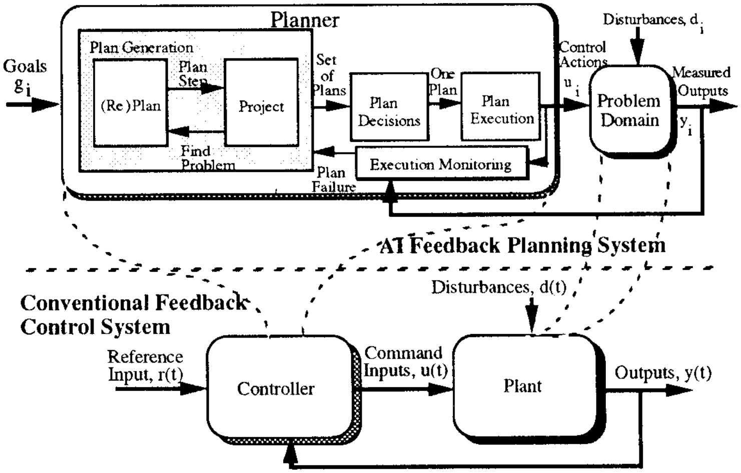

Figure 3.3. Closed Loop Structural Analogy

The planning system examines the difference between the current output situation and the goal to be achieved, and takes subsequent actions. The error in the planning system is not as easy to form as in the conventional control case. This is because distance between symbols is more difficult to quantify. One could, however, say that a problem domain output is closer to the goal if the components that make up the output are closer to satisfying the goal. If the goal is a conjunction of several subgoals, it is closer to the output if the outputs make more of the subgoals true.

Suppose we fix the goal input to the feedback planning system to be the same for all time, i.e., $g_i=g_0$ for all i. The feedback planning system is then considered to be a *regulatory planning system.* It achieves the goal state and regulates the inputs to the problem domain to ensure that the goals are met for all time even in the face of problem domain disturbances.

If the sequence of goals g_i, the exogenous inputs to the planner, change over time and the planner sequentially achieves the goals, the planning system is said to be a *goal following* or *goal tracking planning system.* Notice that if the goals change too quickly, the planner will not be able to keep up, and there will be some *tracking error.*

When one designs a feedback planning system, there are certain properties that are desirable for the closed loop planning system. We refer to these properties collectively as the "closed loop specifications". These could be stability, rate, performance measures, etc.

Normally, stability is always a closed loop specification. The planner is designed so that stability is present in the closed loop system. Take special note that even though feedback planning systems have the ability to stabilize any system that is stabilizable, they can also destabilize an otherwise stable problem domain. As an example, consider the case when because of some delay in receiving feedback information, the planner applies the right plan but at the wrong time. One must be careful in the design so that this is avoided.

A very important advantage of feedback planning systems over their open loop counterparts is their ability to reject problem domain disturbances (reach and maintain a goal even with disturbances) and to be insensitive to problem domain variations (reach a goal even though the model is inaccurate). In conventional control theory these objectives are designed for, using techniques which will produce optimal disturbance rejection and sensitivity reduction. Systems that meet these objectives are said to be *robust*. The theory of robust control addresses these questions.

4. A DISCRETE EVENT SYSTEM THEORETIC FRAMEWORK FOR THE MODELING AND ANALYSIS OF ARTIFICIALY INTELLIGENT PLANNING SYSTEMS

In this Section we will introduce a "logical" DES model that can be used to model and analyze AI planning problems. For simplicity we focus here on goal-directed planning systems where from the initial state it takes only a finite number of steps to reach the goal state. Moreover, as is standard in the AI literature we focus mainly on plan generation and not on execution monitoring and re-planning. Our study here amounts to a formal approach to planning system verification (for an overview of work in formal verification of AI planning systems see [11,1]).

The problem of plan generation is essentially an optimal control problem in that one wants to generate a plan that will transfer the system from the current state to the goal state while using the least amount of resources. Such problems can be solved with dynamic programming but often this approach causes problems with computational complexity. We will show that the plan generation problem is essentially a reachability problem similar to that discussed in Section 3.

4.1 A Problem Domain Model

We consider problem domains that can be accurately modeled with

$$P=(X,Q,\delta,\chi,x_0,X_f) \tag{1}$$

where

(i) X is the possibly infinite set of problem domain states,
(ii) Q is the possibly infinite set of problem domain inputs,
(iii) $\delta:Q\times X\rightarrow X$ is the problem domain state transition function (a partial function),
(iv) $\chi:X\times X\rightarrow \mathbb{R}^+$ is the *event cost function* (a partial function describing the cost of executing a planning step),
(v) x_0 is the initial problem domain state, and
(vi) $X_f \subset X$ is the non-empty finite set of *goal states.*

$\mathbb{R}^+$ denotes the set of positive reals. The model P is limited to representing problem domains that are deterministic in the sense that for a given input there is exactly one possible next state. A state transition can occur in a non-deterministic fashion relative to time so asynchronous problem domains can be modeled. The set

$$E(P)=\{(x,x')\in X\times X:x'=\delta(q,x)\} \tag{2}$$

denotes the (possibly infinite) set of events for the DES P . The event cost function $\chi(x,x')$ is defined for all $(x,x')\in E(P)$; it specifies the "cost" for each event (planning step) to occur and it is required that there exist a $\delta'>0$ such that $\chi(x,x')\geq\delta'$ for all $(x,x')\in E(P)$. Finally, we require that for each $x\in X$, $|\{\delta(q,x):q\in Q\}|$ is finite, i.e., that the graph of P is *locally finite* (hence, P is equivalent to a δ-graph [19,20]).

Let Z be an arbitrary set. Z^* denotes the set of all finite strings over Z including the empty string $\emptyset$. For any $s,t\in Z^*$ such that $s=zz'\cdots z''$ and $t=yy'\cdots y''$, st denotes the concatenation of the strings s and t, and $t\in s$ is used to indicate that t is a substring of s, i.e., $s=zz'\cdots t\cdots z''$.

A string $s\in X^*$ is called a *state trajectory* or *state path* of P if for all successive states $xx'\in s$, $x'=\delta(q,x)$ for some $q\in Q$. Let

$$E_S(P)\subset E(P) \tag{3}$$

denote the set of all events needed to define a particular state path $s\in X^*$ that can be generated by P. For some state path $s=xx'x''x'''\cdots$, $E_S(P)$ is found by simply forming the pairs (x,x'), (x',x''), (x'',x'''),$\cdots$. An *input sequence* (sequence of planning steps) $u\in Q^*$ that produces a state trajectory $s\in X^*$ is constructed by concatenating $q\in Q$ such that $x'=\delta(q,x)$ for all $xx'\in s$. Let $X_Z\subset X$ then

$$\mathcal{X}(P,x,X_Z)\subset X^* \tag{4}$$

denotes the set of all finite state trajectories $s=xx'\cdots x''$ of P beginning with $x\in X$ and ending with $x''\in X_Z$. Then, for instance, $\mathcal{X}(P,x_0,X_f)$ denotes the set of all finite length state trajectories for P that begin with the initial state x_0 and end with a final state $x\in X_f$ and $\mathcal{X}(P,x,X)$ denotes the set of all valid state trajectories for P that begin with $x\in X$. Therefore, $\mathcal{X}(P,x_0,X_f)$ denotes the set of all possible sequences of states that can be traversed by any plan that can be generated that will result in successfully attaining the goal state. A problem domain P is said to be *(x,X_Z)-reachable* if there exists a sequence of inputs $u\in Q^*$ that produces an state trajectory $s\in\mathcal{X}(P,x,X_Z)$. This notion of reachability is closely related to the definition of controllability in Section 3 except that here it is recognized that the more general idea of the ability to reach a *set* of (goal) states is needed (since most often there is more than one goal state).

4.2 An Optimal Plan Generation Problem

Intuitively, the *valid behavior* that the problem domain can exhibit which is modeled by P can be characterized by the set of all its valid state trajectories $\mathcal{X}(P,x,X)$ where $x\in X$, along with its input sequences (it is specified with the graph of P). Let $P=(X,Q,\delta,\chi,x_0,X_f)$ specify the valid behavior of the problem domain and

$$A=(X_a,Q_a,\delta_a,\chi_a,x_{a0},X_{af}) \tag{5}$$

be another model which we think of as specifying the "allowable" behavior for the problem domain P. Allowable problem domain behavior must also be valid problem domain behavior. Formally, we say that the allowable plant behavior described by A *is contained in* P, denoted with A[P], if the following conditions on A are met:

(i) $X_a\subset X$, $Q_a\subset Q$,

(ii) $\delta_a:Q_a\times X_a\rightarrow X_a$, $\delta_a(q,x)=\delta(q,x)$ if $\delta(q,x)\in X_a$
and $\delta_a(q,x)$ is undefined otherwise

(iii) $\chi_a:X_a \times X_a \to \mathbb{R}^+$ is a restriction of $\chi:X \times X \to \mathbb{R}^+$,

(iv) $x_{a0}=x_0$, $X_{af} \subset X_f$.

Also, let $E(A) \subset E(P)$ denote the set of *allowable events* defined as in (2), χ_a is defined for all $(x,x') \in E(A)$, and $E_s(A)$ is defined as in (3) above. In a graph-theoretic sense, A is a *partial subgraph* of P. The model A, specified by the designer, represents the "allowable" problem domain behavior which is contained in the valid problem domain behavior described by the given P. It may be that entering some state, using some input, or going through some sequence of planning steps is undesirable. Such design objectives relating to what is "permissible" or "desirable" plant behavior are captured with A. This formulation is similar to that used for the "supervisor synthesis problems" in [15]. There the authors introduced a minimal acceptable language and legal language and study the synthesis of supervisors so that the resulting language controlled by the supervisor in the plant lies between the acceptable and legal languages. Their minimal acceptable and legal languages specify what we call the allowable behavior A which is contained in P.

To specify optimal plan generation problems let the *performance index* be

$$J:X_a^* \to \mathbb{R}_+ \tag{6}$$

where the cost of a state trajectory s is defined by

$$J(s) = \sum_{(x,x') \in E_s(A)} \chi(x,x') \tag{7}$$

for all $x \in X_a$ and $s \in \mathcal{X}(A,x,X_a)$. By definition, $J(s)=0$ if $s=x$ where $x \in X_a$. Let A describe the allowable behavior for a plant P such that A[P] then we have:

Optimal Plan Generation Problem (OPGP): Assume that A is (x_0,X_{af})-reachable. Find a sequence of inputs $u \in Q_a^*$ that drives the system A along an *optimal state trajectory* s^*, i.e., $s^* \in \mathcal{X}(A,x_0,X_{af})$ such that $J(s^*)=\inf\{J(s):s \in \mathcal{X}(A,x_0,X_{af})\}$.

Since we require that $u \in Q_a^*$ and $s \in X_a^*$, the solutions to the OPGP will achieve not only optimal but also allowable problem domain behavior. There may, in general, be more than one optimal state trajectory, i.e., the solution to the OPGP is not necessarily unique. Here we are concerned with finding only one optimal state trajectory for the OPGP and finding it in a computationally efficient manner. To do this we will invoke the A^* algorithm for the solution to the OPGP. Note that the key problem in using this approach is the specification of the "heuristic function" $\hat{h}(x)$. Here, we use the approach developed in [2-7] for the specification of the heuristic function.

5. PLANNING APPLICATIONS

In this Section we apply the approach in Section 4 to two planning applications: (i) an optimal part distribution problem in flexible manufacturing systems, and (ii) simple AI planning problems (games). In each case we specify the model P for the problem, the allowable behavior A, and state the particular OPGP. Then, using the results in [2-7] we specify admissible and monotone heuristic functions so that A^* can find solutions to the OPGPs in a computationally efficient manner. A^* and the generalized Dijkstra's algorithm were implemented to compare the complexity of the two algorithms.

5.1 Optimal Parts Distribution Problem in Flexible Manufacturing Systems

A Flexible Manufacturing System (FMS) that is composed of a set of identical machines connected by a transportation system is described by a directed graph (M,T) where $M=\{1,2, ..., N\}$ represents a set of machines numbered with $i \in M$ and $T \subset M \times M$ is the set of transportation tracks between the machines. We assume that (M,T) is *strongly connected*, i.e., that for any $i \in M$ there exists a path from i to every other $j \in M$. This ensures that no machine is isolated from any other machine in the FMS. Each machine has a queue which holds parts that can be processed by any machine in the system (with proper set-up). Let the number of parts in the queue of machine $i \in M$ be given by $x_i \geq 0$. There is a robotic transporter that travels on the tracks represented by $(i,j) \in T$ and moves parts between the queues of various machines. The robot can transfer parts from any $i \in M$ to any other $j \in M$ on any path between i and j (it is assumed that the robot knows the path to take, if not A^* could be used to find it). The robot can transfer no more than $\beta \in \mathbb{N}-\{0\}$ parts at one time between two machines. It is assumed that the robot knows the initial distribution of parts, the graph (M,T), and we wish to find the sequence of inputs to the robot of the form "move α ($\alpha \leq \beta$) parts from machine i to machine j" that will achieve an even distribution of parts in the FMS. In this way, we ensure that every machine in the FMS is fully utilized. It is assumed that no new parts arrive from outside the FMS and that no parts are processed by the machines while the redistribution takes place. Our example is similar to the "load balancing problem" in Computer Science except that we require that a minimum number of parts be moved to achieve an even distribution. Next, we specify the model P in (1) of this FMS.

Let $X=\mathbb{N}^N$ denote the set of states and $x_k=[x_1\ x_2 \cdots x_N]^t$ and $x_{k+1}=[x'_1\ x'_2 \cdots x'_N]^t$ denote the current and next state respectively. Let $Q=\{u_{ij}^{\alpha}: \alpha \in \mathbb{N}-\{0\}\}$ be the set of inputs where u_{ij}^{α} denotes the command to the robot to move α parts from machine i to machine j. The state transition function is given by $\delta(u_{ij}^{\alpha}, x_k)= [x_1\ x_2 \cdots x_i-\alpha \cdots x_j+\alpha \cdots x_N]^t$, the event cost function by $\chi(x_k, x_{k+1})=\alpha$, and $x_0=[x_{01}\ x_{02} \cdots x_{0N}]^t$. The set X_f characterizes the state (or states) for which we consider the the parts in the FMS to be at an even distribution. Let int(x) denote the integer part of x (e.g. int(3.14)=3) and "mod" denote modulo. Let

$$L=\text{int}\left(\sum_{i=1}^{N} \frac{x_{0i}}{N}\right) \quad \text{and} \quad L_e=\left(\sum_{i=1}^{N} \frac{x_{0i}}{N}\right) \text{mod}\ N.$$

The value of L represents the amount of parts each machine would have if the parts could be evenly distributed and L_e represents the number of extra parts that we seek to, for instance, distribute across the first L_e machines. With this intent we let $\bar{x}=[\bar{x}_1\ \bar{x}_2 \cdots \bar{x}_N]^t$ where $\bar{x}_i=L+1$ for i, $i \leq L_e$ and $\bar{x}_j=L$ for j, $L_e<j\leq N$ (other states where the parts are considered to be evenly distributed can be specified in a similar manner - an example of this is given below). We often let $X_{af}=\{\bar{x}\}$ so that the evaluation function $\hat{f}(s_x)$ for A^* is easy to compute.

We let A=P, i.e., all valid DES behavior is allowable, but note that our solution will work for any allowable behavior A so long as A is (x_0,X_{af})-reachable. This is very important since it shows that if the robot is informed that some machine or transportation track is out of order, the robot can still evenly distribute the load for the remaining machines that are still appropriately connected to the FMS. It is in this sense that our solution is "fault tolerant". The OPGP for this optimal parts distribution problem involves finding a sequence of inputs u_{ij}^{α} to the robot which will result in it moving the least number of parts to achieve an even distribution, i.e., $x_k \in X_{af}$.

Let Z be an arbitrary non-empty set and let $\rho: Z \times Z \rightarrow \mathbb{R}$ where ρ has the following properties: (i) $\rho(x,y) \geq 0$ for all $x,y \in Z$ and $\rho(x,y)=0$ iff $x=y$, (ii) $\rho(x,y)=\rho(y,x)$ for all $x,y \in Z$, and (iii) $\rho(x,y) \leq \rho(x,z)+\rho(z,y)$ for all $x,y,z \in Z$ (triangle inequality). The function ρ is called a *metric* on Z and $\{Z;\rho\}$ is a *metric space.* Let $z \in Z$ and define $d(z,Z)=\inf\{\rho(z,z'): z' \in Z\}$. The value of d(z,Z) is called the *distance between point z and set Z.* Recall that if $x,y \in \mathbb{R}^n$, $x=[x_1\ x_2 \cdots x_n]^t$, $y=[y_1\ y_2 \cdots y_n]^t$, and $1 \leq p < \infty$, then $\rho_p(x,y)=[\Sigma_{i=1}^{n}|x_i-y_i|^p]^{1/p}$, $\rho_\infty(x,y)=\max\{|x_1-y_1|,|x_2-y_2|, \ldots ,|x_n-y_n|\}$, and ρ_d (*discrete metric*) where $\rho_d(x,y)=0$ if $x=y$ and $\rho_d(x,y)=1$ if $x \neq y$, are all valid metrics on $\mathbb{R}^n$.

First, consider using the metric ρ_p with p=1. Notice that $\rho_1(x_k,x_{k+1})=2\alpha$ for all $(x_k,x_{k+1}) \in E(A)$. Hence, from [2-7] $\hat{h}_1(x_k)=(1/2)\rho_1(x_k,\bar{x})$ ($\bar{x}$ defined above) is admissible and monotone so we get an efficient solution to the OPGP. The results in [2-7] offer another possibility. Consider the metric ρ_∞. Notice that $\rho_\infty(x_k,x_{k+1})=\alpha$ for all $(x_k,x_{k+1}) \in E(A)$, all $x_k \in X$ are isolated points, and hence $\hat{h}_\infty(x_k)=\rho_\infty(x_k,\bar{x})$ ($\bar{x}$ defined above) is admissible and monotone.

Consider the FMS with 3, 4, and 6 machines and track topologies shown in Figure 5.1. For the 3-machine FMS in Figure 5.1 let $\beta=1$ and $x_0=[10\ 0\ 4]^t$; then L=4 and $L_e=2$ and we choose $X_{af}=\{[5\ 5\ 4]^t\}$. $A^*(\hat{h}_1(x_k))$ and $A^*(\hat{h}_\infty(x_k))$ both expand 5 states and result in a optimal state trajectory of cost 5 (i.e., 5 parts is the minimum number of parts that must be moved to achieve a even distribution). The generalized Dijkstra's algorithm expands 36 states to find a solution. If we let $x_0=[11\ 3\ 2]^t$ then L=5 and $L_e=1$. If we choose $X_{af}=\{[6\ 5\ 5]^t\}$, $A^*(\hat{h}_1(x_k))$ and $A^*(\hat{h}_\infty(x_k))$ both expand 11 states and result in a optimal state trajectory of cost 5. The generalized Dijkstra's algorithm expands 51 states to find a solution; hence we see that for the 3-machine FMS A^* using the heuristic functions specified via the results of in [2-7] far outperforms the generalized Dijkstra's algorithm.

For the 4-machine FMS in Figure 5.1 let $\beta=1$ and $x_0=[0\ 5\ 2\ 6]^t$ so that L=3 and $L_e=1$. Choose $X_{af}=\{[4\ 3\ 3\ 3]^t, [3\ 3\ 3\ 4]^t\}$. $A^*(\hat{h}_1(x_k))$ and $A^*(\hat{h}_\infty(x_k))$ expand 38 and 53 states respectively and result in a optimal state trajectory of cost 6 that ends in $[3\ 3\ 3\ 4]^t$. The generalized Dijkstra's algorithm expands 141 states to find a solution to the OPGP for the 4-machine FMS. For the 6-machine FMS in Figure 5.1 let $\beta=1$ and $x_0=[4\ 0\ 1\ 2\ 0\ 5]^t$ so that L=2 and $L_e=0$. Let $X_{af}=\{[2\ 2\ 2\ 2\ 2\ 2]^t\}$. $A^*(\hat{h}_1(x_k))$ expands 82 and results in a optimal state trajectory of cost 6. The generalized Dijkstra's algorithm expanded 798 states to produce the same solution.

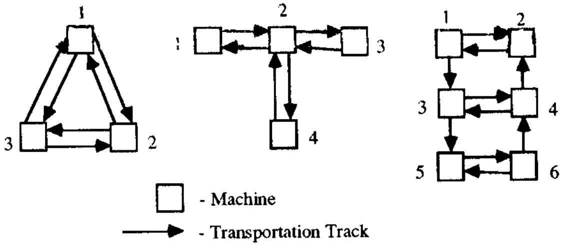

Figure 5.1. Example Flexible Manufacturing System Topologies

Note that if we had allowed $\beta>1$ for the above examples then the computational savings obtained by using A^* over the generalized Dijkstra's algorithm would even be more pronounced. This is the case since A^* would exploit the fact that the robot could move multiple parts so that an even distribution could be achieved quicker. For the generalized Dijkstra's algorithm large β will drastically increase the number of states it visits in finding an optimal state trajectory. Also note that for large N and total number of parts initially in the FMS, for many FMS track topologies the OPGP can easily become too difficult to solve via any method due to combinatorial explosion. We have, however, illustrated that for typical FMS systems the A^* algorithm, with the appropriate heuristic function, can solve the optimal parts distribution problem efficiently - and with significantly fewer computations than conventional techniques.

5.2 Simple Artificial Intelligence Planning Problems: Games

The A^* algorithm has already been used for the solution to many AI planning problems such as tic-tac-toe, the 8 and 15 puzzle, etc. [20]. The extensions to the theory of heuristic search in [2-7] allow for a wider variety of such problems to be studied. For instance, in [4] the authors showed that the metric space approach could be used to specify the standard heuristic functions for the 8-puzzle and discovered several new heuristics for this problem that also work for the more general N-puzzle. In [3] heuristic functions were specified for a "triangle and peg" problem and a simple robotics problem ("blocks world"). Here we study the Missionaries and Cannibals Problem as in [2,3], an AI planning problem for which there currently exist no admissible and monotone heuristic functions. In this way we illustrate that the results in [2-7] facilitate the *discovery* of new heuristics.

The problem statement is as follows: Three missionaries and three cannibals are trying to cross a north-south river by crossing from east to west. As their only means of navigation, they have a small boat, which can hold one or two people. If the cannibals outnumber the missionaries on either side of the river, the missionaries will be eaten; this is to be avoided. Find a way to get them all across the river which minimizes the number of boat trips taken.

First we model this problem with the problem domain model P. Let $X=\mathbb{N}^6$ and $x_k=[x_1\ x_2 \cdots x_6]^t$ and $x_{k+1}=[x'_1\ x'_2 \cdots x'_6]^t$ denote the current and next state respectively. Let x_1 (x_4) and x_3 (x_6) denote the number of cannibals and missionaries on the east (west) side of the river respectively. Let "E" and "W" denote the east and west side of the river respectively. Let "C" and "M" denote cannibals and missionaries. Let $Q=\{q_i:i=1,2,\dots,10\}$ where q_1=2 C W→E (move 2 cannibals from the west side of the river to the east side of the river); q_2=2 C E→W; q_3=1 C W→E; q_4=1 C E→W; q_5=1 C 1M W→E (move 1 cannibal and 1 missionary from the west side of the river to the east side of the river); q_6=1 C 1M E→W; q_7=1M E→W; q_8=1 M E→W; q_9=2 M E→W; q_{10}=2 M W→E. Of course the boat moves in the indicated direction also. For the state transition function we have $\delta(q_2,[3\ 1\ 3\ 0\ 0\ 0]^t) = [1\ 0\ 3\ 2\ 1\ 0]^t$; the other cases are defined similarly. Let $\chi(x_k,x_{k+1})=1$ for all $(x_k,x_{k+1})\in E(P)$, $x_0=[3\ 1\ 3\ 0\ 0\ 0]^t$, and $X_f=\{[0\ 0\ 0\ 3\ 1\ 3]^t\}$.

Notice that we have not represented the part of the problem which states that "the cannibals cannot outnumber the missionaries". We will consider this to be included in the design objectives using the allowable behavior A such that A[P]. Let $X_b=\{x_k\in X: x_1>x_3 \text{ or } x_4>x_6\}$ and $X_a=X-X_b$, $Q_a=Q$, and the definition of A follows immediately. The OPGP for the missionaries and cannibals problem is to (i) find the minimum length sequence of inputs (loads of passengers) that will result in all persons on the west side of the river and (ii) satisfy the constraint that the cannibals cannot outnumber the missionaries which is represented with A.

Currently, there does not exist any monotone $\hat{h}(x_k)$ for this problem. We now show that the results of [2-7] allow for the specification of several such $\hat{h}(x)$. First consider ρ_p where p=2 and notice that $\rho_2(x_k,x_{k+1})\leq\sqrt{10}$ and $\chi(x_k,x_{k+1})=1$ for all $(x_k,x_{k+1})\in E(A)$ so from [2-7] $\hat{h}(x_k)=(1/\sqrt{10})\rho_2(x_k,\bar{x})$ where $\bar{x}=[0\ 0\ 0\ 3\ 1\ 3]^t$ is an admissible and monotone heuristic function. Also notice that $\rho_\infty(x_k,x_{k+1})\leq 2$ so from [2-7] $\hat{h}(x_k)=(1/2)\rho_\infty(x_k,\bar{x})$ where $\bar{x}=[0\ 0\ 0\ 3\ 1\ 3]^t$ is an admissible and monotone heuristic function. When these heuristic functions are used with A^* to find the solution to the OPGP, the minimum length sequence of inputs found was: $q_6, q_8, q_2, q_3, q_9, q_5, q_9, q_3, q_2, q_8, q_6$. The solution involves 11 boat trips, the minimum number of trips needed to solve the problem.

6. CONCLUDING REMARKS

Although a foundation of fundamental concepts has been formed for AI planning systems by drawing an extensive analogy with control theoretic ideas, much work needs to be done in order to mathematically formalize the full range of concepts and planning paradigms presented here. At best, the results of this paper raise many important questions, and clarify some of the issues that may be important in quantitative studies of AI planning systems. Extensive research must be done on developing particular methods to model, analyze, and design AI planning systems. Since the results in this paper are both domain and problem representation independent they are applicable no matter what modeling and analysis methodology is chosen provided that the methodology provides for the study of the fundamental concepts developed here.

As it is quite fundamental to the quantitative study of AI planning systems, the modeling issue must be studied much more extensively. Various questions must be answered: (i) What mathematical formalism to use for the problem representation? (logical or "timed" DES models? hybrid system models?), (ii) What is the expressive power of this formalism? That is, what class of problem domains can be modeled?, (iii) Does the formalism lend itself to analysis, design, and implementation?

Secondly, systematic analysis methods must be developed so that planning system behavior can be studied quantitatively within the developed modeling framework. Before this is done, however, it will be important to determine what is important to analyze. Are there properties other than the ones developed here that need to be analyzed? It is also expected that planning methodologies which lend themselves to analysis will have to be developed. The question of what constitutes good planning system behavior must be answered.

Finally, planning system design must be addressed. It is hoped that a systematic procedure for design is obtained, one that is similar, in character, to the control system design process. Moreover, as indicated in the introduction there is the need to more carefully study implementation issues before we can realistically expect AI planning systems to be used in more challenging real-world applications.

Acknowledgements: This work was partially supported by McDonnell Aircraft Company's Artificial Intelligence Technology Group (reported in Oct. 1987) and by the Jet Propulsion Laboratory. In addition, the first author was also partially supported by an Engineering Foundation Research Initiation Grant.

REFERENCES

[1] Passino K.M., Antsaklis P.J., "A System and Control Theoretic Perspective on Artificial Intelligence Planning Systems", *Int. Journal Applied Artificial Intelligence*, Vol. 3, No. 1, pp. 1-32, 1989.

[2] Passino K.M., *Analysis and Synthesis of Discrete Event Regulator Systems*, Ph.D. Dissertation, Dept. of Electrical and Computer Eng., Univ. of Notre Dame, April 1989.

[3] Passino K.M., Antsaklis P.J., "Artificial Intelligence Planning Problems in a Petri Net Framework", *Proc. of the American Control Conf.*, pp. 626-631, Atlanta GA, June 1988.

[4] Passino K.M., Antsaklis P.J., "Planning Via Heuristic Search in a Petri Net Framework", *Proc. of the Third IEEE Int. Symp. on Intelligent Control*, pp. 350-355, Arlington VA, August 1988.

[5] Passino K.M., Antsaklis P.J., "On the Optimal Control of Discrete Event Systems", *Proc. of the Conference on Decision and Control*, Tampa, Florida, pp. 2713-2718, Dec. 1989.

[6] Passino K.M., Antsaklis P.J., "Near-Optimal Control of Discrete Event Systems", *Proc. of the Allerton Conf. on Communication, Control, and Computing*, pp. 915-924, Univ. of Illinois, Sept. 1989.

[7] Passino K.M., Antsaklis P.J., "Optimal Stabilization of Discrete Event Systems", *Proc. of the IEEE Conf. on Dec. and Control*, Hawaii, pp. 670--671, Dec. 1990.

[8] Antsaklis P.J., Passino K.M., Wang S.J., "Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues", *Journal of Intelligent and Robotic Systems*, Vol. 1, No. 4, pp. 315-342, 1989.

[9] Antsaklis P.J., Passino K.M., Wang S.J., "An Introduction to Autonomous Control Systems", *IEEE Control Systems Magazine, Special Issue on Intelligent Control*, Vol. 11, No. 4, pp. 5-13, June 1991.

[10] Lunardhi A.D., Passino K.M., "Verification of Dynamic Properties of Rule-Based Expert Systems", *Proc. of the IEEE Conf. on Decision and Control*, pp. 1561-1566, Brighton, UK, Dec. 1991.

[11] Tate A., "A Review of Knowledge-Based Planning Techniques", in Merry M., ed., *Expert Systems 85*, Cambridge Univ. Press, Cambridge 1985.

[12] Kalman R., Falb P., Arbib M., *Topics in Mathematical System Theory*, McGraw Hill, NY, 1969.

[13] Hopcroft J., Ullman J., *Introduction to Automata Theory, Languages and Computation*, Addison Wesley. Reading, Mass., 1979.

[14] Special Issue on Dynamics of Discrete Event Systems, *Proceedings of the IEEE*, Vol. 77, No. 1, Jan. 1989.

[15] Ramadge P.J., Wonham W.M., "Supervisory Control of a Class of Discrete Event Processes", *SIAM J. Control and Optimization*, Vol. 25, No. 1, pp. 206-230, Jan. 1987.

[16] Peterson J.L., *Petri Net Theory and the Modeling of Systems*, Prentice Hall, NJ, 1981.

[17] Passino K.M., Michel A.N., Antsaklis P.J., "Lyapunov Stability of a Class of Discrete Event Systems", *Proc. of the American Control Conference*, pp. 2911-2916, Boston, MA, June 1991.

[18] Charniak E., McDermott D., *Introduction to Artificial Intelligence*, Addison Wesley, Reading Mass, 1985.

[19] Hart P.E., Nilsson N.J., Raphael B.,"A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Trans. on Systems Science and Cybernetics*, Vol. SSC-4, No. 2, pp. 100-107, July 1968.

[20] Pearl J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, Mass., 1984.