K. M. Passino and P. J. Antsaklis, "Planning via Heuristic Search in a Petri Net Framework," Proc. of the 3rd IEEE International Symposium on Intelligent Control, pp. 350-355, Arlington, VA, Aug. 24-26, 1988.

PLANNING VIA HEURISTIC SEARCH IN A PETRI NET FRAMEWORK*

K.M. Passino and P.J. Antsaklis
Department of Electrical and Computer Engineering
University of Notre Dame, Notre Dame IN 46556

Abstract

An Artificial Intelligence planning system's main components consist of a planner and a problem domain. The problem domain is the environment that the planner reasons about and takes actions on. In this paper, a special type of Extended Input/Output Petri net is defined and then used as the problem representation for a wide class of problem domains. A planning strategy is developed using results from the theory of heuristic search. In particular, using the developed Petri net framework and metric spaces, a class of heuristic functions is specified that are both admissible and consistent for the A* algorithm. The planning system architecture is discussed and as an illustration of the results two simple planning problems are modelled and solved.

1.0 Introduction

According to the viewpoint presented in [13], Artificial Intelligence (AI) planning systems consist of a planner and a problem domain, their interconnections, and exogenous inputs. The problem domain outputs are fed back to the planner and the planner outputs are the control inputs to the problem domain. There are disturbance inputs to the problem domain and the exogenous inputs to the planner are the goals. The planning system functional architecture considered here is depicted in Figure 1.1. The *problem domain* is the domain (environment) that the planner reasons about and takes actions on. One develops a model of the real problem domain, called the *problem representation*, to study planning systems. It is the task of the *planner* to examine the problem domain outputs, compare them to the current goal, and determine what control inputs to apply to the problem domain so that the goal is met. AI planners employ intelligent problem solving techniques that are fundamental to many intelligent control systems, for instance, in the intelligent control of robots.

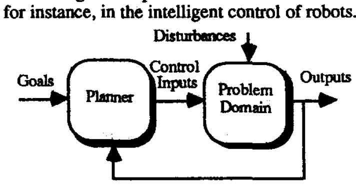


Figure 1.1 AI Planning System Architecture

General information on the theory of AI planning is given in [1-2], and [17]. A very brief overview is given here to establish the terminology. The functional components and details of the planner architecture vary widely depending on the class of problem domains under consideration. The functional components of a typical AI planner are as follows [1]: *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, using the problem representation) 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. More advanced planners may use *execution monitoring* to determine if the currently executing plan is progressing as expected. *Situation assessment* can be used to generate an estimate of the state of the problem domain. The estimate of the state can be used in execution monitoring or to update the state of the model used for projection.

From a system and control theoretic viewpoint there are concepts analogous to controllability, observability, stability, etc., for AI planning systems. There also exists AI planner functional architectures that are analogous to certain control system configurations. A detailed discussion of this analogy (exemplified by the planning system of Figure 1.1) between control systems and planning systems is given in [13]. Planner architectures such as the one in Figure 1.1 are discussed at length, and system and control theoretic concepts are introduced into AI planning theory. In that paper it is argued that although many of the basic issues in AI planning systems are well understood empirically, they have not been adequately quantified in a mathematical framework. It was proposed that the development of a foundation of basic concepts for AI planning systems from their control theoretic counterparts is fundamental to the formation of a mathematical theory for modelling, analysis, and design of AI planning systems for real time environments.

In this paper, a planner which uses projection and plan decisions to determine what plans to execute is considered. The complete plan is generated before it is executed and there is only one goal to be achieved. Also, as in most planning theory research, no disturbances are allowed in the problem domain. The work here is problem representation independent, but a particular representation (the Petri net) is chosen and studied in detail. The results here are domain independent. The results of this paper are now summarized.

The class of problem domains considered here are those which can be modelled by the "Extended Input/Output (I/O) Petri Net" defined in Section 2. Based on this Petri net framework, a planner which uses heuristic search techniques, the A* algorithm, is developed in Section 3. Although for certain special applications the heuristic function is easy to choose so that it is admissible and consistent, in general, practitioners often find it difficult to obtain. To make the planner domain independent, a class of admissible and consistent heuristics for the A* algorithm is specified for AI planning problems that can be modelled by the Petri net. This is done by defining metric spaces associated with the Petri net and using the metric in the A* algorithm. To illustrate the theory, two simple AI planning problems are given in Section 4.0, modelled with Petri nets, metrics are specified, and the A* algorithm is used to generate solutions. Next, relevant research is summarized and compared to the results of this paper.

The Petri net definition here is similar to the definition given in [15] but it also allows for control inputs to the Petri net and outputs. In [7] a "Controlled Petri Net" was defined in a somewhat different manner. The definition in Section 2 includes the so called "inhibitor arc", generalized input and output arcs, and a generalized transition to obtain a Petri net with a relatively high expressive power. The Petri net defined here also allows for the specification of a cost to fire a transition via the specification of the transition cost function. Such costs could, for example, represent a measure of the resources consumed in performing the actions associated with firing a transition.

The results from the theory of heuristic search using the A* algorithm outlined in Section 3.1 mainly come from [5, 6, 3, 16]. Other information can be found in [10, 11, 14]. Under certain conditions the A* algorithm can, from an initial node of a graph, find a least cost path to some goal node. When applied to planning problems the algorithm can be used for projection and the plan decisions discussed above, to determine which plan will achieve some goal with least cost in terms of, for instance, resource consumption. Once the appropriate plan is found A* gives it to the plan executor so that the actions can be taken on the problem domain.

In Section 3 the problem of specifying the heuristic function for a wide class of problem domains is addressed. Results from the theory of metric spaces are outlined and used to prove that for any δ-Graph there exists an admissible and consistent heuristic function for the A* algorithm. It is shown that if a bounded metric is used then there is a whole class of admissible and consistent heuristics. Also, it is shown that if the costs are picked in a certain way then any metric can be used in the heuristic function. Heuristic search via the A* algorithm in a Petri net framework is introduced. It is proven that the Extended I/O Petri net defined in Section 2 generates a certain class of δ-Graphs for which there are known metrics that can be used to specify admissible and consistent heuristic functions.

Work most closely related to ours on how to find admissible and consistent heuristics is called "the generation of heuristics", and "auxiliary models" are often used. These methods are outlined in [14]. They involve searching for a value of the heuristic at each step in the heuristic search; consequently, they are computationally intensive [14]. The results developed here allow for the specification of the heuristic function a priori and thus avoid the search for the value of the heuristic at each step. Details about how the results of this paper extend those originally reported in [12] are provided.

Other relevant research is given in [4]. There the authors use a high level Petri net to represent both the knowledge and inference strategy in expert systems. Some analysis results are obtained. Some planning systems are implemented in the computer programming language named PROLOG. An analysis of concurrency in PROLOG via Petri nets is reported in [9].

2.0 The Extended Input/Output Petri Net for Problem Representation

In this section, the Extended Input/Output (I/O) Petri net model used for the problem domain representation is defined. The definition allows for the control inputs from the planner via a control input arc and allow the planner to sense the state of the problem domain via the measurement places. Also, associated with each transition of the Extended I/O Petri net is its cost to fire. Generalized I/O arcs and transitions are added for modelling flexibility.

A Petri net with a high expressive power (language complexity) has the ability to represent a wide class of dynamic systems whereas one with poor expressive power is limited in the sort of dynamic system that can be represented. For a detailed discussion on this topic see [15]. The net to be defined below contains "inhibitor arcs", hence it has the expressive power of the "Extended Petri Nets" discussed in [15, pp. 189-203]. Peterson provides a proof that "an extended petri net is equivalent to a Turing machine", and hence can model any computable system, i.e., one that can be simulated on a computer. Consequently, the Extended I/O Petri net defined below has a relatively high expressive power. The definition below refines and extends the definition originally given in [12] by adding the generalized I/O arcs and transitions.

* The ideas in the Introduction pertaining to the system and control theoretic perspective on AI planning systems were partially supported by McDonnell Aircraft's Artificial Intelligence Technology Group contract Z71145. The remainder of this work was partially supported by the Jet Propulsion Laboratory, Pasadena California under contract 957856.

K. M. Passino and P. J. Antsaklis, "Planning via Heuristic Search in a Petri Net Framework," Proc. of the 3rd IEEE International Symposium on Intelligent Control, pp. 350-355, Arlington, VA, Aug. 24-26, 1988.

Let $\mathbf{R}$ denote the set of reals and $\mathbf{R}^+$ the strictly positive reals. Let $\mathbf{N}$ denote the set of nonnegative integers. A *multiset* (bag) is a collection of objects over some domain $\mathbf{X}$, but unlike standard definitions of a set, multisets allow multiple occurrences of elements [15]. Let $\mathbf{B}$ be a multiset, then $\#(x,\mathbf{B})$ represents the number of occurrences of element x in multiset $\mathbf{B}$. The set $\mathbf{X}^\infty$ is the set of all multisets over a domain $\mathbf{X}$. If $x,y\in\mathbf{N}^n$, $x=[x_1,x_2,\ldots,x_n]^t$, and $y=[y_1,y_2,\ldots,y_n]^t$ (t indicates transpose) then the statement $x\geq y$ is true iff $x_i\geq y_i$ $1\leq i\leq n$. Similarly for $>,<$, and $\leq$. Let $\emptyset$ denote the null set.

The Extended I/O Petri Net structure is described by $\mathbf{P_S}=(\mathbf{P},\mathbf{T},\mathbf{T_g},\mathbf{I_D},\mathbf{I_N},\mathbf{O_D},\mathbf{I_{G_k}},\mathbf{O_{G_k}},\mathbf{U},\Gamma,\mathbf{Y},\Psi)$ where:

(i) $\mathbf{P}=\{p_1,p_2,\ldots,p_n\}$ is a non-empty finite set of $n=|\mathbf{P}|$ (*state*) *places* represented graphically with circles (◯).

(ii) $\mathbf{T}=\{t_1,t_2,\ldots,t_m\}$ is a non-empty finite set of $m=|\mathbf{T}|$ *transitions* represented graphically by line segments (|).

(iii) $\mathbf{T_g}=\{t_{g1},t_{g2},\ldots,t_{gm_g}\}$ is a finite set of $m_g=|\mathbf{T_g}|$ *generalized transitions* represented graphically by rectangles (▯). In the development below, when it is clear, t_j will also be used to denote any transition, either normal or generalized. The notation t_{gk} will be used for emphasis.

Note that $\mathbf{P}\cap\mathbf{T}\cap\mathbf{T_g}=\emptyset$ and $\mathbf{P}\cup\mathbf{T}\cup\mathbf{T_g}\neq\emptyset$.

(iv) $\mathbf{I_D}:\mathbf{T}\cup\mathbf{T_g}\to\mathbf{P}^\infty$ is a mapping from transitions or generalized transitions to the set of all multisets over $\mathbf{P}$. It is represented graphically by *directed arcs* (⟶) pointing from each *input place of t_j*, $p_i\in\mathbf{I_D}(t_j)$, to t_j. If for some $p_i\in\mathbf{P}$, $\#(p_i,\mathbf{I_D}(t_j))=k>1$, then the mapping can be represented graphically by a directed arc with multiplicity k ($\xrightarrow{k}$). Note that k is finite. If $t_j\in\mathbf{T_g}$ then for all $p_i\in\mathbf{P}$, $\#(p_i,\mathbf{I_D}(t_j))\leq 1$.

(v) $\mathbf{I_N}:\mathbf{T}\to\mathbf{P}^\infty$ is a mapping from transitions to the set of all multisets over $\mathbf{P}$. It is represented graphically by *not arcs* (inhibitor arcs) (—o) pointing from each input place of t_j, $p_i\in\mathbf{I_N}(t_j)$, to t_j. If for some $p_i\in\mathbf{P}$, $\#(p_i,\mathbf{I_N}(t_j))=k>1$, then the mapping can be represented by a not arc with multiplicity k ($\overset{k}{—}$o). Note that k is finite.

(vi) $\mathbf{O_D}:\mathbf{T}\cup\mathbf{T_g}\to\mathbf{P}^\infty$ is a mapping from transitions or generalized transitions to the set of all multisets over $\mathbf{P}$. It is represented graphically by directed arcs pointing from the transition $t_j\in\mathbf{T}\cup\mathbf{T_g}$ to each *output place of t_j*, $p_i\in\mathbf{O_D}(t_j)$. A directed arc with multiplicity k can be used in a manner similar to (iv), except $k=\#(p_i,\mathbf{O_D}(t_j))$. If $t_j\in\mathbf{T_g}$ then for all $p_i\in\mathbf{P}$, $\#(p_i,\mathbf{O_D}(t_j))=k\leq 1$.

(vii) $\mathbf{I_{G_k}}:\mathbf{T}\to 2^{\mathbf{P}}$ is a mapping from transitions to the set of all subsets of $\mathbf{P}$. It is represented graphically by *generalized directed input arcs of type k* (⟹) pointing from each *input place of t_j*, $p_i\in\mathbf{I_{G_k}}(t_j)$, to t_j.

(viii) $\mathbf{O_{G_k}}:\mathbf{T}\to 2^{\mathbf{P}}$ is a mapping from transitions to the set of all subsets of $\mathbf{P}$. It is represented graphically by *generalized directed output arcs of type k* (similar to (vii)) pointing from the transition $t_j\in\mathbf{T}$ to each *output place of t_j*, $p_i\in\mathbf{O_{G_k}}(t_j)$.

For convenience, a *two-way directed arc* (⟷) is used to indicate a *self loop*, i.e. $p_i\in\mathbf{O_D}(t_j)$ and $p_i\in\mathbf{I_D}(t_j)$. A *directed not arc* (←o) is used to indicate the connections $p_i\in\mathbf{O_D}(t_j)$ and $p_i\in\mathbf{I_N}(t_j)$. Also, every arc has a transition on one end and a place on the other, and no transition or place exists without being connected to an arc.

(ix) $\mathbf{U}=\{u_1,u_2,\ldots,u_{n_c}\}$ is the nonempty finite set of $n_c=|\mathbf{U}|$ *control places* which are represented graphically with circles as in (i).

(x) $\Gamma:\mathbf{T}\cup\mathbf{T_g}\to 2^{\mathbf{U}}$ is a mapping from transitions or generalized transitions to the set of all subsets of $\mathbf{U}$. It is represented graphically by *control arcs* (—▪) pointing from each $u_i\in\Gamma(t_j)$ to t_j. It is required that for all $t_j\in\mathbf{T}\cup\mathbf{T_g}$ there exist $u_i\in\Gamma(t_j)$, i.e., all transitions have control arcs connected to them.

(xi) $\mathbf{Y}=\{y_1,y_2,\ldots,y_{n_y}\}$ is the finite set of $n_y=|\mathbf{Y}|$ *measurement places*, represented graphically with circles as in (i).

(xii) $\Psi:\mathbf{P}\to\mathbf{Y}$ is a mapping from places to measurement places. It is represented graphically with a *connection arc* (▷—◁) from $p_i\in\mathbf{P}$ to $y_j\in\Psi(p_i)$. In this paper we require that for all $p_i\in\mathbf{P}$ there exist a unique $y_j\in\Psi(p_i)$.

A complete description, which also includes the *execution* characteristics of the Extended I/O Petri Net, is given by $\mathbf{P_N}=(\mathbf{P_S},\mathbf{T_s},\mathbf{X_p},\mathbf{X_{po}},\mathbf{U_p},\mathbf{Y_p},\mathbf{E_r},\Phi_r,Z)$ where:

(i) $\mathbf{P_S}$ is described above.

(ii) $\mathbf{T_s}=\mathbf{N}$ is the *time index set*. The initial time is 0, and each successive natural number represents an arbitrary length time step.

(iii) $\mathbf{X_p}:\mathbf{P}\mathrm{x}\mathbf{T_s}\to\mathbf{N}$ is the *marking function*, a mapping from (state) places and time steps into nonnegative integers representing the *marking* of the place. The n-dimensional column vector $x_p(k)=[X_p(p_1,k),X_p(p_2,k),\ldots,X_p(p_n,k)]^t$ is used to denote the *state* of the Petri net. The state is represented graphically by *tokens* (●) that are put inside places (e.g. $X_p(p_i,k)=2$ is represented as P_i ⦿). The complete set of "reachable states" [15] will be denoted by $\mathbf{X_p}$.

(iv) $\mathbf{X_{po}}$ is a non-empty finite set of *initial conditions* $x_p(0)$ for the state of the Petri Net; $\mathbf{N}^n\supseteq\mathbf{X_{po}}$. For this paper $|\mathbf{X_{po}}|=1$.

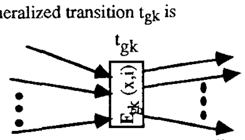
(v) $\mathbf{U_p}:\mathbf{U}\mathrm{x}\mathbf{T_s}\to\mathbf{N}$ is a mapping from control places and time steps into nonnegative integers representing the marking of the control place. The n_c-dimensional column vector $u_p(k)=[U_p(u_1,k),U_p(u_2,k),\ldots,U_p(u_{n_c},k)]^t$ is used to denote the *control input* to the Petri net. It is represented graphically with tokens as in (iii). It is required that $U_p(u_i,k)=0$ or 1 for all $u_i\in\mathbf{U}$ and $k\in\mathbf{T_s}$, and only one element of $u_p(k)$ is equal to 1 for any k.

(vi) $\mathbf{Y_p}:\mathbf{Y}\mathrm{x}\mathbf{T_s}\to\mathbf{N}$ is a mapping from measurement places and time steps to a nonnegative integer representing the marking of the measurement place. Here we let $Y_p(y_j,k)=X_p(p_i,k)$ for all $k\in\mathbf{T_s}$ where $y_j\in\Psi(p_i)$. The $n_y(n)$-dimensional column vector $y_p(k)=[Y_p(y_1,k),Y_p(y_2,k),\ldots,Y_p(y_{n_y},k)]^t$ is used to denote the *output* of the Petri net.

(vii) $\mathbf{E_r}:\mathbf{N}^n\mathrm{x}\mathbf{T_s}\to 2^{\mathbf{T}\cup\mathbf{T_g}}$ is the Petri net *enable rule*, a mapping from an n-dimensional column vector of nonnegative integers representing $x_p(k)$ and time steps into subsets of transitions that are said to be *enabled* at step k. The notation $t_j\in\mathbf{E_r}(k)$ is used to indicate that $t_j\in\mathbf{T}\cup\mathbf{T_g}$ is enabled at step k.

The enable rule is chosen based on the specific modelling task. The generalized arc defined by $\mathbf{I_{G_k}}$ is used to specify certain portions of the enable rule $\mathbf{E_r}$ that will be denoted $\mathbf{E_{rk}}$. For instance, a type 1 arc connected to place p_i may indicate that for t_j, where $p_i\in\mathbf{I_{G_1}}(t_j)$, to be enabled at step k it must be the case that $X_p(p_i,k)=\#(p_i,\mathbf{I_{G_1}}(t_j))$. New types of generalized input arcs can be invented as required, but all must be represent functions $\mathbf{E_{rk}}$ of the form $\mathbf{E_{rk}}:\mathbf{N}^{n1}\mathrm{x}\mathbf{T_s}\to 2^{\mathbf{T}\cup\mathbf{T_g}}$ where $n1\leq n$.

The form for the generalized transition t_{gk} is



The function $\mathbf{E_{gk}}$ is the enable function for the generalized transition $t_{gk}\in\mathbf{T_g}$ and it is defined by $\mathbf{E_{gk}}:\mathbf{N}^{n_{gk}}\mathrm{x}\mathbf{T_s}\to\{0,1\}$. It is also used to specify portions of $\mathbf{E_r}$. It uses the value of the state of the Petri net to determine if t_{gk} is enabled at some time. Note that

$$n_{gk}=\sum_{i=1}^{|\mathbf{P}|}\#(p_i,\mathbf{I_D}(t_{gk}))$$

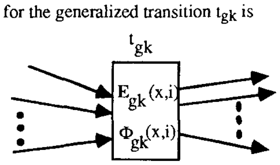
and "1" indicates that t_{gk} is enabled while "0" indicates that it is not enabled. A full specification of the enable rule $\mathbf{E_r}$ involves saying how each transition $t_j\in\mathbf{T}\cup\mathbf{T_g}$ is enabled.

As an example, assume that $\mathbf{T_g}=\emptyset$ and that there are no generalized arcs. A candidate for the enable rule is given by $\mathbf{E_r'}(x_p(k),k)=\{t_j|X_p(p_i,k)\geq\#(p_i,\mathbf{I_D}(t_j))$, for all $p_i\in\mathbf{P}$, and if $p_i\in\mathbf{I_N}(t_j)$, $X_p(p_i,k)<\#(p_i,\mathbf{I_N}(t_j))\}$. A transition t_j can *fire* whenever it is enabled. Tokens are redistributed in the Petri net when a transition fires according to the next state function described below.

(viii) $\Phi_r:\mathbf{T}\cup\mathbf{T_g}\mathrm{x}\mathbf{N}^n\mathrm{x}\mathbf{T_s}\to\mathbf{N}^n$ is the Petri net *next state function (firing rule)*, a mapping from a transition t_j, an n-dimensional column vector of nonnegative integers representing $x_p(k)$, and time steps into an n-dimensional column vector of nonnegative integers representing the next state $x_p(k+1)$. The next state function is defined iff $t_j\in\mathbf{E_r}$.

The firing rule is also chosen according to the specific modelling task at hand. For instance, the generalized output arc defined by $\mathbf{O_{G_k}}$ is used to specify certain portions of the next state function Φ_r which will be denoted Φ_{rk}. As an example, a type 1 arc connected from transition t_j to place p_i may indicate that if t_j is enabled and it fires then no matter how many tokens are in place p_i, 3 tokens should be placed there. New types of generalized output arcs can be invented as required but all must represent functions Φ_{rk} where $\Phi_{rk}:\mathbf{T}\cup\mathbf{T_g}\mathrm{x}\mathbf{N}^{n1}\mathrm{x}\mathbf{T_s}\to\mathbf{N}^{n2}$, where $n1\leq n$ and $n2\leq n$.

The general form for the generalized transition t_{gk} is



The function Φ_{gk} is the next state function for the generalized transition $t_{gk}\in\mathbf{T_g}$ and it is defined by $\Phi_{gk}:\mathbf{N}^{n_{gk}}\mathrm{x}\mathbf{T_s}\to\mathbf{N}^{n_{fk}}$. It is only defined if $t_{gk}\in\mathbf{E_{gk}}$. It uses values of the state of the Petri net to say how tokens are redistributed if t_{gk} is fired at some time. The function $\mathbf{E_{gk}}$ is defined above. The value of n_{gk} is given above in (vii) and n_{fk}, the dimension of the portion of the next state that is affected by firing t_{gk} is given by

$$n_{fk}=\sum_{i=1}^{|\mathbf{P}|}\#(p_i,\mathbf{O_D}(t_{gk})).$$

The full specification of the firing rule Φ_r involves saying how the tokens are redistributed if any transition $t_j\in\mathbf{T}\cup\mathbf{T_g}$ is fired.

Consider the following example of a next state function. Let $\Phi'_r=[\phi_1,\phi_2,\ldots,\phi_n]^t$. Let $A^-=[a^-_{ij}]$, where $a^-_{ij}=\#(p_i,\mathbf{I_D}(t_j))$ and $A^+=[a^+_{ij}]$, where $a^+_{ij}=\#(p_i,\mathbf{O_D}(t_j))$. Let $A=A^+-A^-$. Let $B=[b_{ij}]$ where $b_{ij}=\#(p_i,\mathbf{I_N}(t_j))$. Let a^-_j, and b_j refer to the jth

K. M. Passino and P. J. Antsaklis, "Planning via Heuristic Search in a Petri Net Framework," Proc. of the 3rd IEEE International Symposium on Intelligent Control, pp. 350-355, Arlington, VA, Aug. 24-26, 1988.

columns of A^- and B respectively. Assume $T_g=\emptyset$ and that there are no generalized arcs. Then the example enable rule E_r above can be restated as $E_r'=\{t_j \mid x_p(k)\geq a^-_j, x_p(k)<b_j\}$. Let $u_p(k)=[0\ 0\ \dots\ 1\ \dots\ 0\ 0]^t$ where the 1 is in the jth position $1\leq j\leq m$. Let $C=[c_{ij}]$ with $c_{ij}=1$ if $y_i\in\Psi(p_j)$ and $c_{ij}=0$ if $y_i\notin\Psi(p_j)$. Using E_r' as the enable rule, the next state function $\phi_i(X_p(p_i,k),t_j,k) = X_p(p_i,k+1)=X_p(p_i,k) - \#(p_i,I_D(t_j)) + \#(p_i,O_D(t_j))$ for all i, $1\leq i\leq n$. Hence, for $t_j\in E_r'(k)$ firing the next state function $\Phi'_r(t_j,x_p(k),k)=x_p(k+1)$ or,

$$x_p(k+1)=x_p(k) + Au_p(k)$$
$$y_p(k)=Cx_p(k)$$

which are similar to the *state equations* described in [15]. For this next state function tokens are not removed from any control input place if t_j fires, but they must be present to fire t_j. Likewise, tokens are not removed from places connected to transitions via a not arc. The connection arcs between any place p_i and a measurement place y_j indicate that any tokens added or subtracted from p_i are also added and subtracted from y_j, i.e. with the connection arc they are essentially duplicate places.

(ix) $Z:T\cup T_g x N^n x N^n \rightarrow R^+\cup\{0\}$ is the *transition cost function*, a mapping from a transition or generalized transistion $t_j\in E_r$, $x_p(k)$, and $x_p(k+1)$ into a nonnegative real number that represents the *cost* of firing t_j. Since the firing of a transition often represents some computation or action performed, Z is a measure of the cost to process the state $x_p(k)$ into $x_p(k+1)$ by firing t_j. The transition cost function is defined iff the transition t_j is enabled at step k.

3.0 Metric Spaces and Admissible and Consistent Heuristic Functions in a Petri Net Framework

In this section some of the main results of the theory of heuristic search involving the A^* algorithm are briefly outlined. Results from the theory of metric spaces are outlined and used to prove that for any δ-Graph there exists an admissible and consistent heuristic function for the A^* algorithm. It is shown that if a bounded metric is used then there is a whole class of admissible and consistent heuristics. Also, it is shown that if the costs are picked in a certain way then any metric can be used in the heuristic function. Heuristic search via the A^* algorithm in a Petri net framework is introduced. It is proven that the Extended I/O Petri net defined in Section 2 generates a certain class of δ-Graphs for which there are known metrics that can be used to specify an admissible and consistent heuristic function.

3.1 Heuristic Search: The A^* Algorithm

Some results from the theory of a heuristic search involving the A^* algorithm are outlined below. The main results of the A^* algorithm studied here were obtained in [5, 6, 3]. Probably the most complete reference on heuristic search is [15].

The problem space is represented explicitly by a <u>δ-Graph</u> G=(X,E,C) where:

(i) $X=\{x_1,x_2,x_3, \dots\}$ is the non-empty possibly infinite set of nodes.

(ii) $E=\{e_{ij}\}=\{(x_i,x_j) \mid x_i,x_j\in X\}$ is the non-empty possibly infinite set of directed arcs pointing from x_i to x_j.

(iii) $C=\{c_{ij}\}=\{c_{ij} \mid e_{ij}\in E\}$ is the non-empty possibly infinite set of costs associated with each arc. Also, for all $c_{ij}\in C$, $c_{ij}\geq\delta>0$.

An *implicit* representation of the δ-Graph G is given by a set of source nodes and a *successor operator* $\Gamma:X\rightarrow 2^{XxC}$. When Γ is applied to a node x it is *expanded*. It is required that for all x_i, $|\Gamma(x_i)|$ is finite. The explicit δ-Graph is generated by repeated application of the successor operator Γ to nodes that are generated in expanding nodes.

The *subgraph* G_x from any $x\in X$ is the graph defined implicitly by a single source node x and some Γ defined on X. Each node in G_x is *accessible* from x. A *path* from x_1 to x_k is an ordered set of nodes $<x_1,x_2, \dots ,x_k>$ such that $x_{i+1}\in\Gamma(x_i)$ for all $1\leq i\leq k-1$. There exists a path from x_i to x_j iff x_j is accessible from x_i. Every path has a cost which is obtained by adding the costs of each arc $c_{i,i+1}\in C$. An *optimal path* from x_i to x_j is a path having the smallest cost over the set of all paths from x_i to x_j, call this cost $h(x_i,x_j)$. Denote an estimate of this cost by $\hat{h}(x_i,x_j)$. The concern is with the subgraph G_{x_0} from a single specified *start node* $x_0\in X$. Define the non-empty set X_g, $X\supseteq X_g$ of nodes in G_{x_0} as *goal nodes*. For any node x in G_{x_0} an element $x_g\in X_g$ is a *preferred* goal node of x iff the cost of the optimal path from x to x_g does not exceed the cost of any other path from x to any other member $x'\in X_g$. The symbols x'_g and x''_g will be used to denote preferred goal nodes below.

The objective is to find the optimal path from the start node to a preferred goal node. To help guide the search an *evaluation function* $v:X\rightarrow R^+\cup\{0\}$ is used to rank how promising it is that a node is on an optimal path. The evaluation function is defined so that the node with the smallest value of v(x) is chosen for expansion. One algorithm that performs heuristic search is the A^* algorithm[5]. The details of the algorithm are given there. The evaluation function v(x) must be chosen. Let f(x) be the actual cost of an optimal path constrained to go through x from the start node x_0 to a preferred goal node $x_g\in X_g$. Let f(x)=g(x)+h(x) where g(x) is the actual cost of an optimal path from x_0 to x and h(x) is the cost of an optimal path from x to a preferred goal node of x called x''_g, i.e., $h(x,x''_g)=\min_{x_g\in X_g}(h(x,x_g))$. Since f(x), g(x), and h(x) are not known a priori, the estimates $\hat{f}(x)$, $\hat{g}(x)$, and $\hat{h}(x)$ are used. Therefore, choose $v(x)=\hat{f}(x)=\hat{g}(x)+\hat{h}(x)$. The function $\hat{h}(x)$, called the *heuristic* component of the evaluation function, is used to capture information from the problem domain to guide the search. If $\hat{h}(x)$ satisfies certain properties then the A^* algorithm performs well.

If some goal node is accessible from the start node and $0\leq\hat{h}(x,x'_g)\leq h(x,x''_g)$ for all $x\in X$, then A^* is *admissible* i.e., it is guaranteed to find an optimal path from the start node to a preferred goal node for any δ-Graph [5]. The heuristic $\hat{h}(x)$ is said to be *consistent* if $h(x_i,x_j)+\hat{h}(x_j,x''_g)\geq\hat{h}(x_i,x'_g)$ for all $x_i,x_j\in X$. The heuristic $\hat{h}(x)$ is said to satisfy a *monotone restriction* (or is said to be monotone) if for all $x_j\in\Gamma(x_i)$, $x_i\in X$, $h(x_i,x_j) + \hat{h}(x_j,x''_g)\geq\hat{h}(x_i,x'_g)$. If $\hat{h}(x)$ is consistent then it automatically satisfies the monotone restriction. If $\hat{h}(x)$ satisfies the monotone restriction then if A^* selects x for expansion $\hat{g}(x)=g(x)$, and the value of $\hat{f}(x')$ for x' on the path from x_0 to x is nondecreasing; consequently step (iii) in the A^* algorithm is vacuous and can be removed [5, 11]. Suppose that there are two versions of the A^* algorithm called A_1 and A_2 which use evaluation functions $\hat{f}_i(x)=\hat{g}_i(x)+\hat{h}_i(x)$ where $0\leq\hat{h}_i(x)\leq h(x)$ for all $x\in X$, i=1,2. The algorithm A_2 is said to be *more informed* than A_1 if for all nongoal nodes x, $\hat{h}_2(x)>\hat{h}_1(x)$. The following *optimality* result was obtained. If $\hat{h}_2(x)>\hat{h}_1(x)$ then at the termination of their A^* searches every node expanded by A_2 was also expanded by A_1. It follows that A_1 expands at least as many nodes as does A_2. See [14] for a more complete discussion.

3.2 Metric Spaces and Admissible and Consistent Heuristics

In the literature on heuristic search the heuristic function $\hat{h}(x,x'_g)$ is used to estimate the *distance* from the current node x to a preferred goal node x'_g. The theory developed to date makes the notion of this distance intuitively clear. In this section the notion of distance is quantified in a formal mathematical framework then used in finding heuristic functions. First, some of the results from the theory of metric spaces taken from [8] are outlined.

Let $\bar{X}$ be an arbitrary non-empty set and let $\rho:\bar{X}x\bar{X}\rightarrow R$ where ρ has the following properties:

(i) $\rho(x,y)\geq 0$ for all $x,y\in\bar{X}$ and $\rho(x,y)=0$ iff x=y,

(ii) $\rho(x,y)=\rho(y,x)$ for all $x,y\in\bar{X}$,

(iii) $\rho(x,y)\leq\rho(x,z)+\rho(z,y)$ for all $x,y,z\in\bar{X}$ (Triangle inequality).

The function ρ is called a *metric* on $\bar{X}$ and the mathematical system consisting of ρ and $\bar{X}$, denoted $\{\bar{X};\rho\}$, is called a *metric space*. The elements of $\bar{X}$ are often called *points*, and ρ(x,y) is frequently called the *distance* from a point $x\in\bar{X}$, to a point $y\in\bar{X}$. Equivalently, ρ is a metric iff, (i) ρ(x,y)=0 iff x=y, and (ii) $\rho(y,z)\leq\rho(x,y)+\rho(x,z)$ for all $x,y,z\in\bar{X}$. Let $\Delta_\rho(\bar{X})$ denote the class of functions that are metrics over the set $\bar{X}$.

If ρ is a metric over the set $\bar{X}$ then so is αρ where α>0. There is a distinction between bounded and unbounded metric spaces. Let $\{\bar{X};\rho\}$ be a metric space. If there exists a positive number γ where $\rho(x,y)\leq\gamma$ for all $x,y\in\bar{X}$, $\{\bar{X};\rho\}$ is said to be a *bounded metric space*. If $\{\bar{X};\rho\}$ is not bounded, $\{\bar{X};\rho\}$ is said to be an *unbounded metric space*. The class of all bounded metrics over $\bar{X}$ will be denoted $\Delta_b(\bar{X})$. Define the function $\rho_1:\bar{X}x\bar{X}\rightarrow R$ by

$$\rho_1(x,y)=\frac{\rho(x,y)}{1+\rho(x,y)}.$$

The metric space $\{\bar{X};\rho_1\}$ is a bounded metric space even though $\{\bar{X};\rho\}$ may not be bounded. In fact, $0\leq\rho_1(x,y)<1$ for all $x,y\in\bar{X}$. The notion of distance between sets is also defined. Let $\{\bar{X};\rho\}$ be a metric space, and let Y and Z be two non-empty subsets of $\bar{X}$. The *distance between sets Y and Z* is defined to be

$$d(Y,Z)=\inf\{\rho(y,z) \mid y\in Y, z\in Z\}.$$

Let $p\in\bar{X}$ and define

$$d(p,Z)=\inf\{\rho(p,z) \mid z\in Z\}.$$

The value of d(p,Z) is called the *distance between point p and set* Z. If |Z| is finite then d(p,Z)=0 implies that $p\in Z$. The standard notation "inf" is used to denote the infimum. A number $u\in R$ is an *infimum* of some set S, $R\supseteq S$ (denoted u=inf{S}) if it satisfies the two conditions:

(i) $u\leq s$ for all $s\in S$

(ii) if $v\in R$ and $v\leq s$ for all $s\in S$, then $v\leq u$.

If $\inf\{S\}\in S$ then min{S} is equivalent to inf{S}, hence if |S| is finite inf is equivalent to min.

<u>Theorem 1</u>: Given <u>any</u> δ-Graph G=(X,E,C) there exists a heuristic function $\hat{h}(x_i,x'_g)$ that can be constructed from functions in the class $\Delta_\rho(X)$ that is both admissible and consistent.

K. M. Passino and P. J. Antsaklis, "Planning via Heuristic Search in a Petri Net Framework," Proc. of the 3rd IEEE International Symposium on Intelligent Control, pp. 350-355, Arlington, VA, Aug. 24-26, 1988.

Proof: The proof proceeds by constructing a particular heuristic function that is both admissible and consistent for any δ-Graph. Let ρ be an arbitrary metric from $\Delta_\rho(\mathbf{X})$. Let

$$\rho_1(x,y)=\frac{\rho(x,y)}{1+\rho(x,y)}$$

so that $\{\mathbf{X};\rho_1\}$ is a bounded metric space. Recall that for all $c_{ij}\in \mathbf{C}$, it is given that $c_{ij}\geq\delta>0$. It follows that $\{\mathbf{X};\rho_a\}$ is a metric space where $\rho_a(x,y)=\delta\rho_1(x,y)$ for all $x,y\in \mathbf{X}$. Define functions $\eta_{ij}:\mathbf{R}^+\cup\{0\}\rightarrow\mathbf{R}^+$ and $\eta_{ij}(r)\geq r$ for all $r\in \mathbf{R}^+\cup\{0\}$ associated with each edge $e_{ij}\in \mathbf{E}$. The functions η_{ij} are chosen so that $c_{ij}=\eta_{ij}(\rho_a(x_i,x_j))$ with $x_j\in\Gamma(x_i)$. For instance, a simple choice for η_{ij} is $\eta_{ij}(\rho_a(x_i,x_j))=c_{ij}=\rho_a(x_i,x_j)+b'_{ij}$, where $b'_{ij}\geq 0$. The c_{ij} and $\rho_a(x_i,x_j)$ are given, hence the values of the b'_{ij} can be found.

Let $<x_0,x_1, \ldots ,x_k>$ be any path generated by A^*. From the triangle inequality, $\rho_a(x_i,x_k)\leq\rho_a(x_i,x_{i+1})+\rho_a(x_{i+1},x_k)$ for all i, $0\leq i\leq k-1$. Therefore,

$$\rho_a(x_i,x_k)\leq\sum_{i=0}^{k-1}\rho_a(x_i,x_{i+1}).$$

Selecting η_{ij} as stated above, it follows that

$$\rho_a(x_i,x_k)\leq\sum_{i=0}^{k-1}\eta_{i,i+1}(\rho_a(x_i,x_{i+1}))=\sum_{i=0}^{k-1}c_{i,i+1}=h'(x_i,x_k).$$

Where $h'(x_i,x_k)$ is the actual cost of some path between x_i and x_k, but not necessarily the optimal one. The A^* algorithm is known to be *complete*, i.e., it will terminate with a solution if one exists [5, 14]. By assumption, the node x_g is accessible, therefore for some path generated by A^*, $x_k=x_g$, hence $\rho_a(x_i,x_g)\leq h'(x_i,x_g)$. Since this is true for any path it will be true for the optimal path, hence for all $x_i\in \mathbf{X}$ and $x_g\in \mathbf{X}_g$, $\rho_a(x_i,x_g)\leq h(x_i,x_g)$. If there were just one goal node x_g the proof would be finished by choosing $\hat{h}(x_i,x_g)=\rho_a(x_i,x_g)$. In general, $|\mathbf{X}_g|\geq 1$ so it must be shown that $\hat{h}(x_i,x'_g)\leq h(x_i,x''_g)$ where x'_g is the preferred goal node for $\hat{h}$ from node x_i and x''_g is the preferred goal node for h from x_i. It is not necessarily the case that $x'_g=x''_g$. The preferred goal node changes depending on what the current node x_i is.

Choose

$$\hat{h}(x_i,x'_g)=d(x_i,\mathbf{X}_g)=\inf\{\rho_a(x_i,x_g)\mid x_g\in \mathbf{X}_g\}$$

as the heuristic function. The value at which the inf is achieved is called x'_g and is the preferred goal node at x_i. For this heuristic function it is assumed that for any set S $\inf\{S\}\in S$ hence if $x_i\in \mathbf{X}_g$ then $d(x_i,\mathbf{X}_g)=0$. For all $x_i\in \mathbf{X}$ and $x_g\in \mathbf{X}_g$, $\hat{h}(x_i,x'_g)\leq\rho_a(x_i,x_g)$, and in particular $\hat{h}(x_i,x'_g)\leq\rho_a(x_i,x''_g)$. From above, for all $x_i\in \mathbf{X}$, $\rho_a(x_i,x''_g)\leq h(x_i,x''_g)$. This gives $\hat{h}(x_i,x'_g)\leq h(x_i,x''_g)$ for all $x_i\in \mathbf{X}$. For admissibility it must be shown that $0\leq\hat{h}(x_i,x'_g)\leq h(x_i,x''_g)$ for all $x_i\in \mathbf{X}$. Since $d(x_i,\mathbf{X}_g)\geq 0$ the proof for admissibility is done.

To prove consistency, which is equivalent to monotonicity, it must be shown that $\hat{h}(x_i,x'_g)\leq h(x_i,x_j)+\hat{h}(x_j,x''_g)$ for all $x_i,x_j\in \mathbf{X}$ such that $x_j\in\Gamma(x_i)$ where x'_g and x''_g are the (not necessarily equal) preferred goal nodes for $\hat{h}$ from x_i and x_j respectively. By the triangle inequality $\rho_a(x_i,x''_g)\leq\rho_a(x_i,x_j)+\rho_a(x_j,x''_g)$. Also due to the way that preferred goal nodes are chosen $\rho_a(x_i,x'_g)\leq\rho_a(x_i,x''_g)$. It follows that $\rho_a(x_i,x'_g)\leq\rho_a(x_i,x_j)+\rho_a(x_j,x''_g)$ for all $x_j\in \mathbf{X}$, so it is the case that $d(x_i,\mathbf{X}_g)\leq d(x_i,x_j)+d(x_j,\mathbf{X}_g)$. Given the choice for $\hat{h}(x_i,x'_g)$ above the proof for consistency is done. Q.E.D.

Theorem 1 does not give the actual heuristic function to be used in A^*. It only says that for any problem space that can be represented by a δ-Graph one can always construct a heuristic function that is both admissible and consistent no matter how difficult it may seem. The often rather difficult job of finding the heuristic function remains. Also the theorem does not say how good the heuristic is. Theorem 1 quantifies in a mathematical framework the statements in the theory of heuristic search about "distance". It also makes it clear how the A^* algorithm operates in evaluating distance to preferred goal nodes and how it switches preferred goal nodes.

Often it is the case that a bounded metric can be found for the underlying set **X**. In this case the following corollary is useful.

Corollary 1: For any δ-Graph $\mathbf{G}=(\mathbf{X},\mathbf{E},\mathbf{C})$, $\hat{h}(x_i,x'_g)=d(x_i,\mathbf{X}_g)=\inf\{\rho(x_i,x_g)\mid x_g\in \mathbf{X}_g\}$, where $\rho\in\Delta_b(\mathbf{X})$, is an admissible and consistent heuristic function.

Proof: Suppose that ρ is bounded by $\gamma>0$. Replace ρ_a above with $\rho_a(x_i,x_j) = (\delta/\gamma)\rho(x_i,x_j)$. The heuristic function $\hat{h}(x_i,x'_g)$ is the one constructed in the proof of Theorem 1, hence it is admissible and consistent. Q.E.D.

Sometimes it is the case that the costs are not specified for a problem domain, and the objective is to just find a solution, i.e., any path from the start node to any goal node. Also, sometimes it is possible to load heuristic information from the problem domain into the costs. For instance, if a heuristic says "if you are at some set of nodes it is in general better to have some other set of nodes as successors". The costs associated with the arcs that connect these sets of nodes can be made small, hence A^* may be more likely to make the proper expansion. In either case the following corollary to Theorem 1 is useful.

Define functions $\eta'_{ij}:\mathbf{R}^+\cup\{0\}\rightarrow\mathbf{R}^+$ and $\eta'_{ij}(r)>r$ for all $r\in \mathbf{R}^+\cup\{0\}$ associated with each edge $e_{ij}\in \mathbf{E}$.

Corollary 2: Assume that the costs $c_{ij}\in \mathbf{C}$ associated with the edges **E** that connect nodes from **X** are not specified. If the costs are obtained by choosing η'_{ij} so that $c_{ij}=\eta'_{ij}(\rho(x_i,x_j))$ with $x_j\in\Gamma(x_i)$, where $\rho\in\Delta_\rho(\mathbf{X})$, then $d(x_i,\mathbf{X}_g)=\inf\{\rho(x_i,x_g)\mid x_g\in \mathbf{X}_g\}$ is an admissible and consistent heuristic.

Proof: From the definition of η'_{ij} it is clear that $\mathbf{G}=(\mathbf{X},\mathbf{E},\mathbf{C})$ is indeed a δ-Graph. Let $\rho_a(x_i,x_j)=\rho(x_i,x_j)$. Using the proof of Theorem 1 and replacing η_{ij} with η'_{ij} the corollary is proven. Q.E.D.

Hence if the costs for the δ-Graph can be chosen, and they are chosen in a certain manner described above, then the metric used in the heuristic function can be chosen arbitrarily from $\Delta_\rho(\mathbf{X})$. This corollary was originally proven in [12] for the case of $|\mathbf{X}_g|=1$. Note that if there are no self loops on the δ-Graph, i.e., it is not the case that $x\in\Gamma(x)$ for some $x\in \mathbf{X}$, then the functions η'_{ij} can be chosen as $\eta'_{ij}(r)\geq r$ and Corollary 2 will still be valid. This is important since $\eta_{ij}(r)=r$ is simpler to implement. The problem of specifying the exact value of the metric for the heuristic function remains and will now be addressed.

3.3 Admissible and Consistent Heuristics in a Petri Net Framework

First, a certain class of δ-Graphs is defined. Let Ω_θ denote the class of δ-Graphs that have nodes that are θ-dimensional vectors of real numbers, i.e., $\Omega_\theta=\{(\mathbf{X},\mathbf{E},\mathbf{C})\mid \text{for all } x_i\in \mathbf{X},\ x_i\in \mathbf{R}^\theta,\ \theta\geq 1\}$.

Theorem 2: Suppose that an Extended I/O Petri net $\mathbf{P_N}$ is used for a problem representation and that its initial state $\mathbf{X}_{po}$ is specified. Then the Petri net generates a δ-Graph of class Ω_θ. Also, if the metric for the heuristic function (defined in the proof of Theorem 1) is of class $\Delta_b(\mathbf{R}^\theta)$ then there are known metrics that can be used to form an admissible and consistent heuristic function.

Proof: Let $\mathbf{X}=\mathbf{N}^n$, the state space of the Petri net $\mathbf{P_N}$. The start node $x_0=x_p(0)\in \mathbf{X}_{po}$ and the edges and costs are generated by

$$\Gamma(x_p(k))=\{(x_p(k+1),c)\mid x_p(k+1)=\Phi(t_j,x_p(k),k) \text{ and } c=Z(t_j,x_p(k),x_p(k+1)) \text{ for all } t_j\in \mathbf{E}_r\}$$

The function Z is chosen according to η_{ij}. Note that $|\Gamma(x_p(k))|$ is finite for all $x_p(k)$ and the assigned cost $c\geq\delta>0$. Let $\theta=n$ and the Petri net generates a δ-Graph of class Ω_θ. Using Corollary 1, the proof is complete if there are known metrics that can be used in the heuristic function $d(x_i,\mathbf{X}_g)$. Some of these are given below.

First note that for $\mathbf{R}^n\supseteq\bar{\mathbf{X}}$ if $\{\mathbf{R}^n;\rho\}$ is a metric space then so is $\{\bar{\mathbf{X}};\rho\}$ [8]. Hence any metric over $\mathbf{R}^n$ is also a metric over $\mathbf{N}^n$, the nodes of the δ-Graph, i.e., the state space of the Petri net. Denote elements $x,y\in \mathbf{R}^n$ by $x=[\xi_1,\xi_2, \ldots ,\xi_n]^t$ and $y=[\lambda_1,\lambda_2, \ldots ,\lambda_n]^t$ where $\xi_i,\lambda_i\in \mathbf{R}$ for i=1,2, ... ,n. A few candidate metrics are listed below:

(i) $\rho_d(x,y)=0$ if $x=y$ and 1 if $x\neq y$ is a metric on $\bar{\mathbf{X}}$ (an arbitrary non-empty set) called the *discrete metric*. If it is used in the heuristic function breadth first search is obtained.

(ii) Let $\bar{\mathbf{X}}=\mathbf{R}^n$, $p\in \mathbf{R}$, $1\leq p\leq\infty$, and $\alpha_i>0$, then $\{\mathbf{R}^n;\rho_p\}$ is a metric space where

$$\rho_p(x,y)=\left[\sum_{i=1}^{n}\alpha_i|\xi_i-\lambda_i|^p\right]^{1/p}$$

(iii) In particular, let W be a positive definite matrix. Then if

$$\rho_2(x,y)=\left[(x-y)^tW(x-y)\right]^{1/2}$$

$\{\mathbf{R}^n;\rho_2\}$ is a metric space.

(iv) Let $\bar{\mathbf{X}}=\mathbf{R}^n$ and $x,y\in \mathbf{R}^n$, $\alpha_i>0$, and

$$\rho_\infty(x,y)=\max\{\alpha_1|\xi_1-\lambda_1|,\ \alpha_2|\xi_2-\lambda_2|,\ \ldots ,\alpha_n|\xi_n-\lambda_n|\}$$

then $\{\mathbf{R}^n;\rho_\infty\}$ is a metric space.

The metrics listed are several of the more common metrics used. Any metric over $\mathbf{R}^n$ will satisfy Theorem 2. Q.E.D.

The proof for Theorem 2 originally appeared in [12] for the case of $|\mathbf{X}_g|=1$. If the costs c_{ij} for the δ-Graph are not specified then Corollary 2 applies here also. Another question left to be answered is how good the above metrics are. This is a standard problem in the theory of the A^* algorithm. For instance, if the chosen metric is such that it gives an extremely conservative estimate of the cost from all nodes x to the preferred goal node, then A^* may expand too many nodes in finding a solution. The computational complexity involved in computing the metric itself must also be considered when studying the computational demands of a particular A^* algorithm.

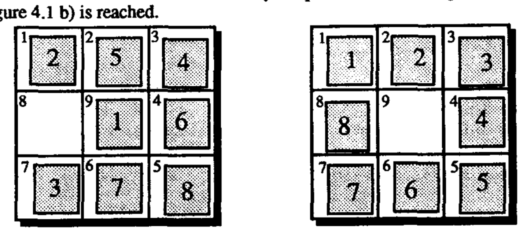
Theorem 2 allows the planning system designer to transfer the work of choosing the heuristic function $\hat{h}(x_i,x'_g)$ to forming the Petri net model of the problem domain under consideration. Since a problem domain representation of some sort must be used, it is useful to use the Extended I/O Petri net model since it leads to the specification of admissible and consistent heuristic functions via the results of this section. This can be valuable if it is not clear how to pick the heuristic function for a particular problem domain. However, if the problem domain cannot be modelled via the Petri net defined the result cannot be utilized. Also, practically speaking, the Petri net model may be too complex to be utilized in the implementation of the A^* algorithm.

4.0 Examples

This section contains two simple examples that illustrate some of the results in Section 3. These include the the 8-Puzzle and a "Think and Jump" game.

8-Puzzle: The first example is called the 8-Puzzle and is a classic example used in the literature on heuristic search [10, 11, 14]. This example will be used to illustrate that the heuristic functions chosen via the results in Section 3 include those which have been previously developed in the literature. This shows that for this example "good" heuristic functions can be developed based on the use of metric spaces and Petri nets in this paper.

The 8-Puzzle has a board with nine cells, eight tiles that lie in the cells, and one blank cell. The game is shown in Figure 4.1. The tiles are shaded, labeled with numbers 1-8, and lie in the cells that are labelled with numbers 1-9. A tile can be moved from one cell to another if any adjacent cell has no tile in it. For instance, from the tile configuration in Figure 4.1 a) tile 1 can be placed in cell 8 leaving cell 9 empty. The game begins with a arbitrary initial state and the proper sequence of tile moves must be chosen by the planner so that the goal state shown in Figure 4.1 b) is reached.



a) Arbitrary Initial State   b) Goal State

Figure 4.1 The 8-Puzzle

First the Petri net model of the 8-Puzzle is developed. To help visualize the Petri net model developed below it is convenient to associate the cells of the 8-Puzzle with points in the 2-dimensional space of real numbers. Use (i,j) to denote a point in this space. For instance, cell 1 is associated with the point (0,2), cell 2 with (1,2), cell 9 with (1,1) and so on. The association is depicted in Figure 4.2.

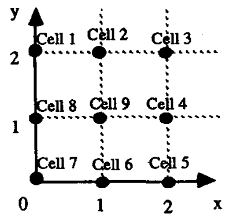


Figure 4.2

We think of the game being played "on" this coordinate system, i.e. if tile 5 is in cell 4 it is "at" point (2,1). Essentially, there is a copy of the coordinate system for each tile. The marking of the Petri net will reflect the position of every tile in this coordinate system. To do this, two places $p_i \in P$ are associated with each tile and the blank cell (which will be considered to be "tile 0"), one with the x-coordinate and one with the y-coordinate. Generalized transitions are used to indicate the action of moving each tile to the blank cell if it is adjacent to the tile. The Petri net is given in Figure 4.3.

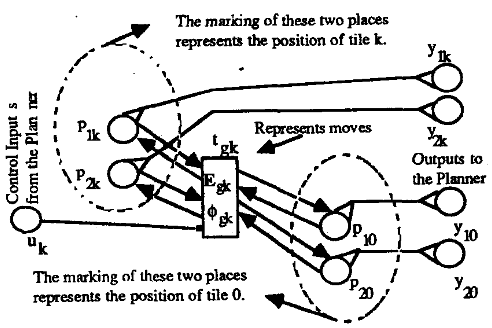


Figure 4.3 Extended I/O Petri Net of the 8-Puzzle

The markings of p_{1k} and p_{2k} represent the x and y coordinates of tile k, and hence its cell position. There is a a copy of the portions of the net indexed with k for each k, $1 \leq k \leq 8$. The 8 control inputs from the planner are connected to each of the 8 generalized transitions. If $U_p(u_i,k)=1$ for some step k then transition t_{gk} is to fire if it is enabled. There are 18 output places to allow the planner to measure the state of the system. Let $x_{pi}(k)=[X_p(p_{1i},k)\ X_p(p_{2i},k)]^t$ be the 2-dimensional column vector representing the marking of places p_{1i} and p_{2i} at step k, i.e., it represents the cell that tile i is in. The enable rule for the 8-Puzzle is given by $E^8_{gi}(x_p(k),k)=\{t_{gi} \,|\, |X_p(p_{1i},k)-X_p(p_{10},k)| + |X_p(p_{2i},k)-X_p(p_{20},k)|=1\}$. The complete enable rule, called E_r^8, is composed of each of the eight generalized transition enable rules E^8_{gi}. It indicates that transition t_{gi} is enabled if the blank cell is a distance of 1 away from tile i. The firing rules Φ_{gi} for each of the eight generalized transtions are: if $t_{gi} \in E^8$, and t_{gi} is chosen to be fired at step k, then the values of $x_{pi}(k)$ and $x_{p0}(k)$ are switched for step k+1. The complete firing rule is called Φ_r^8. The transition cost function is defined by assigning a cost of "1" for firing each generalized transition.

There have been several admissible and consistent heuristics specified for this example. Among these the number of tiles that are not in their appropriate goal cells, and (ii) the so called "Manhattan Distance", i.e., the sum of the number of moves that it would take to move the tiles that are not in their goal state into their goal state assuming that there are no other tiles in the way [14]. Case (i) will be referred to as $\hat{h}_1$, and case (ii) as $\hat{h}_2$. It is now shown that both of these heuristics can be specified via the results of Section 3.

Let $x_p(k)=[x_{p0}(k)^t\ x_{p1}(k)^t \ldots x_{p8}(k)^t]^t$ and let x_{pgi} denote the goal state for tile i. For instance, the goal state for tile 5 is given by $x_{pg5}=[2\ 0]^t$. Let the entire goal state be $x_g=[x_{pg0}\ x_{pg1} \ldots x_{pg8}]^t$. The discrete metric is ρ_d. The vector $x_c(k)=[\rho_d(x_{p0}(k),x_{pg0})\ \rho_d(x_{p1}(k),x_{pg1}) \ldots \rho_d(x_{p8}(k),x_{pg8})]^t$ indicates with a 1 at its ith element that tile i is not in its goal state and a 0 if it is. Define x_{co} to be equal to a column vector of zeros the same size as $x_c(k)$. The heuristic function $\hat{h}_1(x_p(k),x_g)=\rho_p(x_c(k),x_{co})$ with p=1 is a sum of the elements of the $x_c(k)$ vector, i.e., the number of tiles that are not in their proper cells. The heuristic function $\hat{h}_2(x_p(k),x_g)=\rho_p(x_p(k),x_g)$ with p=1. This is the case because the ρ_p metric with p=1 measures distance the same way that we count the number of moves it takes to move a tile to its goal position if no other tiles are in the way. Note that both metrics used are bounded because the state space of the Petri net is finite and the $\eta_{ij}(r)=r$. Using Corollary 1 both heuristics are admissible and consistent. The metrics $\rho_2(x_p(k),x_g)$ and $\rho_\infty(x_p(k),x_g)$ are also candidate metrics with special interpretations. Hence, for this example the metric space formulation facilitates the discovery of new heuristics. Notice that for the general N-Puzzle the results indicate that the chosen metrics are still admissible and consistent.

It may be interesting to note that the results of this paper allow for the construction of known heuristics but the results are especially useful if a new problem domain appears and it is not clear how to pick a heuristic function. This is illustrated next.

Think and Jump Game: The second example is a "Think-and-Jump" game involving a triangular board with ten holes in it, and 9 pegs which fit into the holes. The 9 pegs are put in the holes. Pegs are removed if they are "jumped" by other pegs. A peg can jump another peg only if there is an empty hole directly on the other side of the peg. See Figure 4.4.

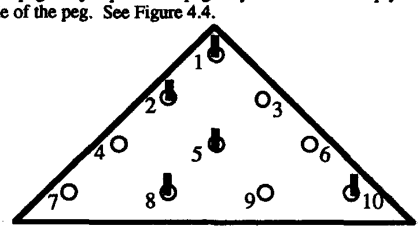


Figure 4.4 Think and Jump Game

For the initial configuration of pegs on the board shown above the peg in hole 2 can jump the peg in hole 5 which leaves no peg in holes 2 and 5 and one peg in hole 9. From this configuration the peg in hole 9 can be used to jump the peg in hole 8 to leave a peg in hole 7, and none in holes 8 and 9. From the initial configuration, the peg in hole 10 cannot be used to jump the peg in hole 5, and the peg in hole 1 cannot be used to jump the peg in hole 5. The object of the game is to remove as many pegs from the board as possible. The best you can do is to end up with only one peg.

First the Petri net model is constructed. Let the places $P=\{p_i\}$, i=1,2, ... ,10, correspond to the holes in the board and let the tokens correspond to the pegs. It is relatively easy to see how the Petri net depicted in Figure 4.5 was developed.

K. M. Passino and P. J. Antsaklis, "Planning via Heuristic Search in a Petri Net Framework," P roc. o f t he
3 rd I EEE I nternational S ymposium o n I ntelligent C ontrol , pp. 350-355, Arlington, VA, Aug. 24-26, 1988.

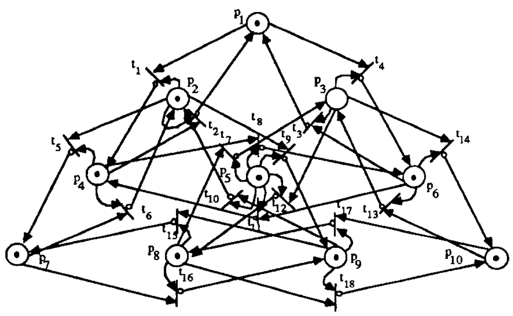


Figure 4.5 Petri Net Model of the Think and Jump Game

The transitions $\mathbf{T}=\{t_j\}$, j=1,2, ... ,18 represent the eighteen possible moves from various peg configurations. The inputs and outputs are not depicted on the graph, but follow directly from the definition of the Petri net in Section 2. The enable rule is given by $\mathbf{E}'_r$, the next state function by Φ'_r, and the cost function Z assigns a cost of 1 to every transition. The state equations were used to represent the game. The initial state is $x_p(0)=[1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^t$ ($\mathbf{X}_{po}=\{x_p(0)\}$), and the goal states are $\mathbf{X}_g=\{[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^t, [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^t, [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^t\}$. The set of goals describes three ways to "win" the game. These are to end up with: (i) one peg left in position 2, (i) pegs in positions 9 and 1, and (ii) pegs in positions 9 and 2. Since one peg is removed from the board for every move, to reach goal (i) it will take 8 steps, whereas it will only take 7 steps to reach goal (ii) or (iii). The planner is to find the solution path to the goal state that takes the least number of steps, hence it should seek goals (ii) or (iii).

The heuristic function is chosen to be $\hat{h}(x_i,x'_g)=\inf\{(1/\gamma)\rho_2(x_i,x_g) \mid x_g\in \mathbf{X}_g\}$ where W is a diagonal matrix with diagonal elements $w_{ii}>0$. Choose W to be an 10x10 identity matrix then the heuristic function is in the form of a Hamming distance. The metric ρ_2 for this example is bounded since **X** is finite. The bound γ is given by

$$\gamma=\left[\sum_{i=1}^{10} w_{ii}\right]^{1/2}$$

Using Theorem 2 and Corollary 1 the chosen metric is both admissible and consistent. The A^* algorithm was implemented and used to find a solution to the Think and Jump problem using the chosen metric. It expanded 58 nodes. The solution generated was for the planner to fire transitions: $t_2, t_{16}, t_{17}, t_{14}, t_7, t_4, t_{13}$, the optimal length solution. If only goal (i) is placed in the goal set then the solution length, found in [12], is 8.

5.0 References

[1] Charniak E., McDermott D.,Introduction to Artificial Intelligence, Addison Wesley, Reading Mass, 1985.
[2] Cohen P.R., Feigenbaum E.A.,The Handbook of Artificial Intelligence, Volume 3, Kaufmann Pub., California, 1982.
[3] Gelperin D., "On the Optimality of A*", Artificial Intelligence, Vol. 8, pp. 69-76, 1977.
[4] Giordana A., Saitta L., "Modelling Production Rules by Means of Predicate Transition Networks", Inf.Sciences, Vol. 35, No. 1, pp. 1-41, 1985.
[5] 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.
[6] Hart P.E., Nilsson N.J., Raphael B.,"Correction to: A Formal Basis for the Heuristic Determination of Minimum Cost Paths", SIGART Newsletter Vol. 37, pp. 28-29, 1972.
[7] Krogh B. "Controlled Petri Nets and Maximally Permissive Feedback Logic", Proc. Allerton Conf., pp. 317-326, Urbana, IL, Sept. 1987.
[8] Michel A.N., Herget C.J., Mathematical Foundations in Engineering and Science: Algebra and Analysis, Prentice Hall, NJ, 1981.
[9] Murata T., Zhang D., "A High-Level Petri Net Model for Parallel Interpretation of Logic Programs", IEEE Conf. on Computer Languages, pp. 123-132, Florida, Oct. 1986.
[10] Nilsson N.J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, NY, 1971.
[11] Nilsson N.J., Principles of Artificial Intelligence, Tioga, NY, 1980.
[12] 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, June 1988.
[13] Passino K.M., Antsaklis P.J., "A System and Control Theoretic Perspective on Artificial Intelligence Planning Systems", Control Systems Technical Report #63, Dept. of Electrical and Computer Engineering, University of Notre Dame, July 1988.
[14] Pearl J., Heuristics, Addison-Wesley, Reading, Mass., 1984.
[15] Peterson J., Petri Net Theory and the Modeling of Systems, Prentice Hall, NJ, 1981.
[16] Vanderbrug G.J., "Problem Representations and Formal Properties of Heuristic Search", Information Sci., Vol. 11, pp. 279-307, 1976.
[17] Wilensky R., Planning and Understanding, Addison Wesley. Reading, Mass., 1983.