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

Artificial Intelligence planning systems consist of a planner and a problem domain. The problem domain is the environment that the planner interacts with 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 heuristic 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 modeled 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 environment that the planner interacts with 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 in order to determine what feasible actions should be used to transform the problem domain into the goal. This is shown in the diagram 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.

\[ \text{Figure 1.1 AI Planning System Architecture} \]

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. For example, a state estimator 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. In this paper it is assumed that although some of the basic ideas in AI planning systems are well understood empirically, they have not been adequately quantified in a mathematical framework. It was shown in [4] that the development of a foundation of basic concepts for AI planning systems from their control theoretic counterparts is fundamental to the formulation of a mathematical theory for modeling, 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 actions to take to achieve a given plan is goal. First 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 modeled by the Extended Input/Output (IO) 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 planed domain independent, a class of admissible and consistent heuristics for the A* algorithm is specified for AI planning problems that can be modeled 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, modeled 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 net" was defined in a somewhat different manner. The definition in Section 2 includes the so called "inhibitor arc", general input and output arcs, and a generalized transition to obtain a Petri net with a relatively high expressive power. The Petri net defined here allows also for the specification of a cost for 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 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 planning 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 generalized and used to prove that for any 5-Graph there exists a non-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 using the A* algorithm in a Petri net framework is introduced. It is proven that the Extended IO Petri net defined in Section 2 generates a certain class of 5-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 techniques involve searching for a value of the heuristic at each step in the heuristic search; consequently, they are computation intensive [14]. The results developed here allow also for the specification of the heuristic function a priori and thus avoid the search for the value of the heuristic at each step. Detailed results on 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 (IO) 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 allows the planner to issue the state of the problem domain via the measurement places. Also, associated with each transition of the Extended IO Petri net is its cost to fire. Generalized IO 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 [13]. 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 IO 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 IO 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 F30602-77-C-0010. The remainder of this work was partially supported by the Jet Propulsion Laboratory, Pasadena California under contract 957856.

**1988 IEEE 350

Let \( \mathbf{R} \) denote the set of reals and \( \mathbb{R}^n \) the strictly positive reals. Let \( \mathbb{N} \) denote the set of natural numbers. A multi-set is a collection of objects of some domain \( X \), but unlike standard definitions of a set, multitudes allow multiple occurrences of elements [15]. Let \( B \) be a multi-set, then \( \#(B) \) represents the number of occurrences of element \( x \) in multi-set \( B \). The set \( N^0 \) is the set of all multiset over a domain \( X \). If \( x_1, x_2, ...; x_n \) and \( y_1, y_2, ...; y_n \) \( (n \geq 1) \) is an \( n \)-tuple then the statement \( x=y \) is true iff \( x_1=y_1, x_2=y_2, ...; x_n=y_n \). Similarly for \( x, y, z \) and \( s, t, u \).

Let \( \emptyset \) denote the null set.

The Extended I/O Petri Net structure is described by \( \mathbf{PN}=\langle \mathbf{P}, \mathbf{T}, \mathbf{X}, \mathbf{U}, \mathbf{Y}, \mathbf{O}, \mathbf{Z} \rangle \) where:

(i) \( \mathbf{P}=\{p_1, p_2, ...; p_n \} \) is a non-empty finite set of \( \mathbb{N}^0 \)-place (state) places represented graphically with circles.

(ii) \( \mathbf{T}=\{t_1, t_2, ...; t_m \} \) is a non-empty finite set of \( \mathbb{N}^0 \)-transitions represented graphically with line segments.

(iii) \( \mathbf{X}=\{x_1, x_2, ...; x_n \} \) is a finite set of \( \mathbb{N}^0 \times \mathbb{N} \) generalized transitions represented graphically with \( \mathbb{N}^0 \times \mathbb{N} \) directed arcs.

(iv) \( \mathbf{U} = \{u_1, u_2, ...; u_r \} \) will also be used to denote any transition, either normal or generalized.

(v) \( \mathbf{Y}=\{y_1, y_2, ...; y_n \} \) is a mapping from \( \mathbf{X} \) to \( \mathbb{N}^0 \times \mathbb{N} \) functions (4).

(vi) \( \mathbf{O} = \{o_1, o_2, ...; o_m \} \) is a mapping from \( \mathbf{X} \) to \( \mathbb{N}^0 \times \mathbb{N} \) functions (4).

(vii) \( \mathbf{Z} = \{z_1, z_2, ...; z_r \} \) is a mapping from \( \mathbf{X} \times \mathbb{N} \rightarrow \mathbb{N}^0 \times \mathbb{N} \) functions (4).

The initial time is 0, and each successive natural number of steps into nonnegative integers representing the marking of the place. It

\[ \mathbf{PN}=(\mathbf{P}, \mathbf{T}, \mathbf{X}, \mathbf{U}, \mathbf{Y}, \mathbf{O}, \mathbf{Z}) \]

is finite.

If \( \mathbf{y_1}(\mathbf{p}_i(k))=\#(\mathbf{p}_i \mathbf{X}(\mathbf{t}_j)) \), then the statement \( \mathbf{x}_2 \mathbf{y} \) is true if \( \mathbf{x}_1 \mathbf{y}=\#(\mathbf{p}_i \mathbf{X}(\mathbf{t}_j)) \). Similarly for \( \mathbf{x}, \mathbf{y}, \mathbf{z} \) and \( \mathbf{s}, \mathbf{t}, \mathbf{u} \).

The enable rule, \( \mathbf{E}_k \), is a mapping from \( \mathbf{X} \times \mathbf{T} \rightarrow \mathbb{N}^0 \times \mathbb{N} \) the Petri net enable rule, a mapping from an \( n \)-dimensional column vector of nonnegative integers representing \( \mathbf{x}_k \) and time steps into subsets of transitions that are said to be enabled at \( k \). The notation \( \mathbf{E}_k(\mathbf{x}(k)) \) is used to indicate that \( \mathbf{t}_j \mathbf{T} \mathbf{E}_k \) is enabled at step \( k \).

The enable rule is chosen based on the specific modelling task. The generalized arc defined by \( \mathbf{D} \) is used to specify portions of the enable rule. The enable rule defined by \( \mathbf{D} \) will be denoted \( \mathbf{E}_k \). For instance, a type 1 arc connected to place \( \mathbf{p}_i \) may indicate that for \( \mathbf{t}_j \) to be enabled at step \( k \) it must be the case that \( \mathbf{x}_p(\mathbf{p}_i(k))=\#(\mathbf{p}_i \mathbf{X}(\mathbf{t}_j)) \). New types of generalized input arcs can be invented as required, but all must be represent functions \( \mathbf{E}_k \) of the form \( \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \).

The form for the generalized transition \( \mathbf{t}_j \mathbf{E}_k \) is

\[ \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \]

\[ = (\theta_1(\mathbf{x}(k), \mathbf{t}_j), \theta_2(\mathbf{x}(k), \mathbf{t}_j), \theta_3(\mathbf{x}(k), \mathbf{t}_j)) \]

\[ \text{where} \quad \theta_1(\mathbf{x}(k), \mathbf{t}_j) \quad \theta_2(\mathbf{x}(k), \mathbf{t}_j) \quad \theta_3(\mathbf{x}(k), \mathbf{t}_j) \]

\[ \text{are functions.} \]

The function \( \mathbf{E}_k \) is the enable function for the generalized transition \( \mathbf{t}_j \mathbf{E}_k \) and it is defined by \( \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \rightarrow \mathbb{N}^0 \times \mathbb{N} \). It is also used to specify portions of \( \mathbf{E}_k \).

It uses the value of the state of the Petri net to determine if \( \mathbf{t}_j \mathbf{E}_k \) is enabled at some time. Note that

\[ \text{"1" indicates that } \mathbf{t}_j \mathbf{E}_k \text{ is enabled while "0" indicates that it is not enabled.} \]

A full specification of the enable rule \( \mathbf{E}_k \) involves saying how each transition \( \mathbf{t}_j \mathbf{E}_k \) is enabled.

As an example, assume that \( \mathbf{t}_j \mathbf{E}_k \) and that there are no generalized arcs. A candidate for the enable rule is given by \( \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \rightarrow \mathbb{N}^0 \times \mathbb{N} \) for all \( \mathbf{t}_j \) and \( \mathbf{p}_i \) if \( \mathbf{p}_i \mathbf{X}(\mathbf{t}_j) \rightarrow \mathbf{p}_i \mathbf{X}(\mathbf{t}_j) \).

A transition \( \mathbf{t}_j \) 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.

\[ \mathbf{t}_j \mathbf{E}_k \]

\[ \text{is the next state function for the generalized transition } \mathbf{t}_j \mathbf{E}_k \text{ and it is defined by } \mathbf{t}_j \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \rightarrow \mathbb{N}^0 \times \mathbb{N} \text{. It only defined if } \mathbf{t}_j \mathbf{E}_k \text{ is enabled at some time.} \]

\[ \text{Function } \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \text{ is given by } \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) = \mathbf{t}_j \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \]

\[ \text{defined by } \mathbf{t}_j \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \rightarrow \mathbb{N}^0 \times \mathbb{N} \text{. It uses values of the state of the Petri net to say how tokens are redistributed if } \mathbf{t}_j \mathbf{E}_k \text{ is fired at some time.} \]

The function \( \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \) is given above in (iv) and (v) the dimension of the portion of the next state that is affected by firing \( \mathbf{t}_j \mathbf{E}_k \) is given by

\[ \mathbf{E}_k(\mathbf{x}(k), \mathbf{t}_j) \]
Let $C = (c_{ij})$ with $c_{ij} = 1$ if $y_{ij}$ represents some computation or action performed. Often, $y_{ij}$ is the transition or generalized transition $t_{ji}$ added and subtracted from $y_{ij}$, i.e., with the connection $(pi, 0, y_{ij})$ for all $i, k \in \mathbb{N}$. Hence, for $y_{ij} = (k_{ij}, c_{ij})$ the firing the next state function $\Phi = \Phi(y_{ij+1})$ is

$$y_{ij+1} = y_{ij} + c_{ij} \text{ or } y_{ij+1} = y_{ij} - c_{ij}$$

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

Therefore, choose $v(x) = f(x) + h(x)$, where $f(x)$ is the heuristic component of the evaluation function, is used to capture information from the problem domain to guide the search. If $f(x)$ satisfies certain properties then the A* algorithm performs well.

If the heuristic function is defined from the $\Phi(x)$, then $\Phi(x)$ is the transition cost function, a mapping from a transition or generalized transition $t_{ji}$ to $\mathbb{R}$ and $\Phi(x)$ is a non-negative real number that represents the cost of firing $t_{ji}$. Since the firing of a transition often represents some computation or action performed, $Z$ is a measure of the cost to process the state $t_{ji}$ into $t_{ji+1}$ by firing $t_{ji}$. The transition cost function is defined iff the transition $t_{ji}$ 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 $\delta$-Graph there exists a 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 MO Petri net defined in Section 2 generates a certain class of $\delta$-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 [16].

Also, the function $v(x), \forall x \in X$, is called a metric on $X$ and the mathematical system consisting of $\delta$ and $X$, denoted $(\delta, X)$, is called a metric space. The elements of $X$ are often called points, and $d(x, y)$ is frequently called the distance from a point $x \in X$ to $y \in X$. Equivalently, $d(x, y)$ is a metric if $d(x, y) \geq 0$ for all $x, y \in X$, and $d(x, y) = 0$ if and only if $x = y$. The metric $d(x, y)$ is said to be monotone if for all $x, y, z \in X$, $d(x, y) \leq d(x, z) + d(z, y)$. The metric $d(x, y)$ is said to be a bounded metric if there exists a number $M > 0$ such that $d(x, y) \leq M$ for all $x, y \in X$. The metric $d(x, y)$ is said to be a metric space if it is an extension of a metric space.

3.2 Metric Spaces and Admissible and Consistent Heuristics

In the literature on heuristic search the heuristic function $h(x) \geq 0$, $\forall x \in X$, is used to estimate the distance from the current node $x \in X$ to a preferred goal node $g$.

The theory developed to date makes the notion of distance intuitively clear. In this section the notion of distance is quantified in a formal mathematical framework and then finding heuristic functions for $X \subseteq \mathbb{R}^n$ is explored. In what follows, some of the results from the theory of metric spaces taken from [8] are outlined.

Let $X$ be an arbitrary non-empty set and let $p: X \times X \rightarrow \mathbb{R}$ where $p$ has the following properties:

(i) $p(x, y) \geq 0$ for all $x, y \in X$ and $p(x, y) = 0$ if $x = y$,

(ii) $p(x, y) = p(y, x)$ for all $x, y \in X$,

(iii) $p(x, y) + p(y, z) \geq p(x, z)$ for all $x, y, z \in X$ (Triangle inequality).

The function $p$ is called a metric on $X$ and the mathematical system consisting of $p$ and $X$, denoted $(p, X)$, is called a metric space. The elements of $X$ are often called points, and $p(x, y)$ is frequently called the distance from a point $x \in X$ to $y \in X$. Equivalently, $p(x, y) \geq 0$ for all $x, y \in X$, and $p(x, y) = 0$ if and only if $x = y$. The metric $p(x, y)$ is said to be a bounded metric if there exists a number $M > 0$ such that $p(x, y) \leq M$ for all $x, y \in X$. The metric $p(x, y)$ is said to be a metric space if it is an extension of a metric space.

Let $X$ be a metric space, and let $Y$ and $Z$ be two non-empty subsets of $X$. The distance between sets $Y$ and $Z$ is defined to be $d(Y, Z) = \inf\{p(x, y) : x \in Y, y \in Z\}$.

Let $\delta$ and $X$ define $d(p, Z) = \inf\{p(x, y) : x \in X, y \in Z\}$.

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

(i) $r \geq s$ for all $s \in S$

(ii) if $r \geq s$ for all $s \in S$, then $r = s$.

If $|S| \leq \infty$ then $\inf(S)$ is equivalent to inf(S), hence if $S$ is finite inf is equivalent to min.

Theorem 3: Given any $\delta$-Graph $G(X, E)$ there exists a heuristic function $h(x) \geq 0$, $\forall x \in X$, that can be constructed from functions in the class $\delta(X)$ that is both admissible and consistent.

The proof proceeds by constructing a particular heuristic function that is both admissible and consistent for any &Graph. Let &theta; be an arbitrary metric from &Delta;(X). Let

\[ p_\theta(x,y) = \inf \{ p(x,z) + &theta;z(y) : z \in X \} \]

so that \( X_1(\theta) \) is a bounded metric space. Recall that for all \( c \in C \), it is given that \( c_U \leq \theta \). It follows that \( X_1(\theta) \) is a metric space where \( p_\theta(x,y) = \delta(x,y) \) for all \( x,y \in X \). Define functions \( \eta^i_r: R^+ \cup \{0\} \rightarrow R^+ \) and \( \phi^i_r: R \rightarrow R \) for all \( r \in R' \cup \{0\} \) associated with each edge \( c \in E \). The functions \( \eta \) are chosen so that \( c_l \cdot \eta \) is a metric over \( E \). For instance, a simple choice for \( \eta \) is \( \eta(c_l) = \delta(x,y) \cdot \gamma^l \), where \( \gamma^l \leq 0 \). The \( \eta \) and \( \phi \) functions are given, hence the values of the \( \theta \) can be found.

Let \( x_0, x_1, \ldots, x_n > \) be any path generated by \( A^* \). From the triangle inequality, \( p_\theta(x_i,x_{i+1}) = \inf \{ p(x_i,z) + \theta^i_r(z,y) : z \in X \} \) for all \( i \leq k-1 \). Therefore,

\[ \sum_{i=0}^{n} \phi(i) = \sum_{i=0}^{n} \phi(i) \]

Selecting \( \phi \) as above, it follows that

\[ \sum_{i=0}^{n} \phi(i) = \sum_{i=0}^{n} \phi \]

Where \( h(x,y) \) is the actual cost between two points, \( x \) and \( y \), 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 \( x \) is accessible, therefore for some path generated by \( A^* \), \( x_k = x \). Hence, we have this residual for the open path, hence for all \( x \) and \( y \), \( d(x,y) \) is bounded, where \( b(x,y) \) is the preferred goal node for \( h \) from node \( x \), and \( h'(x,y) \) is the preferred goal node for \( h \) from \( x \). It is not necessarily the case that \( x_g = x \). The preferred goal changes depending on what the current node \( x \) is.

Choose

\[ \hat{h}(x,y) = \inf \{ p(x,z) + \theta(z,y) : z \in X \} \]

as the heuristic function. The value at which the inf is achieved is called \( x_g \) and is the preferred goal node at \( x \). For this heuristic function it is assumed that for any \( x \) and \( y \), \( d(x,y) \) is finite for all \( x \) and \( y \). Hence any metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

The function \( d(x,y) \) is chosen according to \( v^i_r \). Note that if \( d(x,y) \) is finite for all \( x \) and \( y \), then the function \( d(x,y) \) is also finite. For this function it is assumed that \( d(x,y) \) is finite for all \( x \) and \( y \). Hence any metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

Therefore, the metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

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 &Graph one can always construct a heuristic function that is both admissible and consistent no matter how difficult it may seem. The theorem does not say how good the heuristic function remains. Also the theorem does not say how good the heuristic function remains. Also the theorem does not say how good the heuristic function remains. Also the theorem does not say how good the heuristic function remains.

3.3 Admissible and Consistent Heuristics in a Petri Net Framework

First, a certain class of &Graphs is defined. Let \( \Omega \) denote the class of &Graphs that have nodes that are \( n \)-dimensional vectors of real numbers, i.e.,

\[ \Omega = \{ (x,y) : x \in \mathbb{R} \} \]

Theorem 2: Suppose that an Extended I/O Petri net \( \mathcal{P} \) is used for a problem representation and that its initial state \( X_0 \) is specified. Then the Petri net generates a &Graph of class \( \Omega \). Also, if the metric for the heuristic function (defined in the proof of Theorem 1) is of class \( \Omega \), then there are known metrics that can be used to form an admissible and consistent heuristic function.

Proof: Let \( X \in \mathbb{R}^n \), the state space of the Petri net \( \mathcal{P} \). The start node \( x_0 \) is \( X_0 \), and the edges and costs are generated by

\[ (x_0, y_0) \rightarrow (x_1, y_1) \]

for some predicate \( \theta(x_0, y_0) \). The state \( x_0 \) and \( y_0 \) are the preferred goal nodes at \( x \). For this heuristic function it is assumed that for any \( x \) and \( y \), \( d(x,y) \) is finite for all \( x \) and \( y \). Hence any metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

The function \( d(x,y) \) is chosen according to \( v^i_r \). Note that if \( d(x,y) \) is finite for all \( x \) and \( y \), then the function \( d(x,y) \) is also finite.

First note that for \( R^n \), \( \delta(x,y) \) is a metric space then so is \( X \) [8]. Hence any metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

Therefore, the metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

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 &Graph one can always construct a heuristic function that is both admissible and consistent no matter how difficult it may seem. The theorem does not say how good the heuristic function remains. Also the theorem does not say how good the heuristic function remains. Also the theorem does not say how good the heuristic function remains. Also the theorem does not say how good the heuristic function remains.

3.3 Admissible and Consistent Heuristics in a Petri Net Framework

First, a certain class of &Graphs is defined. Let \( \Omega \) denote the class of &Graphs that have nodes that are \( n \)-dimensional vectors of real numbers, i.e.,

\[ \Omega = \{ (x,y) : x \in \mathbb{R} \} \]

Theorem 2: Suppose that an Extended I/O Petri net \( \mathcal{P} \) is used for a problem representation and that its initial state \( X_0 \) is specified. Then the Petri net generates a &Graph of class \( \Omega \). Also, if the metric for the heuristic function (defined in the proof of Theorem 1) is of class \( \Omega \), then there are known metrics that can be used to form an admissible and consistent heuristic function.

Proof: Let \( X \in \mathbb{R}^n \), the state space of the Petri net \( \mathcal{P} \). The start node \( x_0 \) is \( X_0 \), and the edges and costs are generated by

\[ (x_0, y_0) \rightarrow (x_1, y_1) \]

for some predicate \( \theta(x_0, y_0) \). The state \( x_0 \) and \( y_0 \) are the preferred goal nodes at \( x \). For this heuristic function it is assumed that for any \( x \) and \( y \), \( d(x,y) \) is finite for all \( x \) and \( y \). Hence any metric over \( R^n \) is also a metric over \( N^n \), the nodes of the &Graph, i.e.,

\[ d(x,y) \leq \delta(x,y) \]

The function \( d(x,y) \) is chosen according to \( v^i_r \). Note that if \( d(x,y) \) is finite for all \( x \) and \( y \), then the function \( d(x,y) \) is also finite.
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 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 an 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.

![Figure 4.1 The 8-Puzzle](image)

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 p1, p2 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 4.2.

![Figure 4.2](image)

The markings of p1 and p2 represent the x and y coordinates of tile k, and hence its cell position. There is a copy of the portions of the net indexed with k for each k, i.e., cell 8. The control inputs from the planner are connected to each of the generalized transitions. If the tile is chosen for some step k then transition t(k) is fired if it is enabled. There are 18 output places to allow the planner to measure the state of the system. Let s(k) = \{(x1(k), y1(k)), ..., (x8(k), y8(k)), 0\} be the 2-dimensional column vector representing the marking of places p1 and p2 at step k, i.e., it represents the cell that tile i is in. The enable rule for the 8-Puzzle is given by:

\[ E_k(s) = \begin{cases} 1 & \text{if } s(8) = 0 \text{ and } s(0) = 0 \\ 0 & \text{otherwise} \end{cases} \]

The values of s(k) are switched for step k+1. The complete firing rule is called \( f_k \). 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 are (i) 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 \( h_1 \), and case (ii) as \( h_2 \). It is now shown that both of these heuristics can be specified via the results of Section 3.

For the initial configuration of Figure 4.1(a) \( p_1 \) and \( p_2 \) are associated with the tiles in their goal state and \( p_3 \) is associated with the blank cell. There are no tiles in goals state. The heuristic function \( h_1(s) \) is defined to be the sum of the elements of the \( s \) vector, i.e., the number of tiles that are not in their proper cells. The heuristic function \( h_2(s) = \sum_{k=1}^{8} |s(k)| \) is defined to be the sum of the absolute values of the elements of the \( s \) vector. This is the same as \( h_1(s) \) but with the metric changed to Manhattan Distance. The transition cost function is defined by assigning a cost of "1" for firing each generalized transition.

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. 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](image)

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 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.

The Petri net model is constructed. Let the places \( p_1 = [p_1, 1, 2, ... 10] \) 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.
The transitions $T=\{t_i\}, i=1,2, ... 18$ 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 $E_T$, the next state function by $\Phi_T$, 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 $x_P(0)=[1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0]$, and the goal states are $x_G=[0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1], [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0], [0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0]$. The set of goals describes three ways to "win" the game. These are to end up with: (i) one peg left in position 2, (ii) pegs in positions 9 and 1, and (iii) pegs in positions 9 and 2. 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

$$h(x,G) = \inf_{(\gamma,p_2)\in \gamma(p_2)} \sum_{i=1}^{10} w_{ij},$$

where $W$ is a diagonal matrix with diagonal elements $w_{ij}=0$. Choose $W$ to be an identity matrix then the heuristic function is in the form of a Hamming distance. The metric $p_2$ for this example is bounded since $X$ is finite. The bound $\gamma$ is given by

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