Heuristic Search for Target-Value Path Problem

Lukas Kuhn, Tim Schmidt, Bob Price, Johan de Kleer, Rong Zhou and Minh Do

Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
lukas.kuhn@parc.com

Abstract

In this paper, we define a class of combinatorial search problems in which the objective is to find a set of paths in a graph whose elements' value is as close as possible to some *target value*. Unlike the usual shortest path problem, the goal is not necessarily to find paths with minimum length. We show that in most cases it is possible to decompose the problem into components where heuristic search can be used. We demonstrate the benefits of this approach on a synthetic domain and illustrate an instantiation of the approach for a problem in model-based diagnosis.

1. Introduction

In this paper, we define a class of combinatorial search problems in which the objective is to find a set of paths in a graph whose elements' value is as close as possible to some *target value*. Unlike the usual shortest path problem, the goal is not to find a path with minimum length.

These target value problems arise in a variety of domains. In consumer recommendation domains people often have approximate targets [6]. For instance one might want to plan a hike in a park with a duration of about 3 hours and have a system pull together a selection of trails and lookouts that would take close to 3 hours to complete. In diagnosis, we often wish to find plans which succeed with a probability as close to a certain value as possible in order to test a hypothesis. To find such a plan we can define a valuation related to the probability of success and search for a plan whose value matches some target value as close as possible.

Unfortunately, this important class of combinatorial problems cannot be solved straightforwardly with heuristic search methods such as A^* [4]. We show, however, that a principled understanding of the nature of target value search leads to a decomposition of which significant parts can be solved using A^* methods.

In the next section we formally define the target value problem and derive a heuristic search method. The heuristic search is then evaluated on a set of (synthetic) problems from the diagnosis domain designed to explore the performance envelope of this new technique.

2. Target Value Search

2.1. Problem Definition

Let $G = (\mathbb{V}_G, \mathbb{E}_G)$ be a finite, acyclic, directed graph, $\mathbb{V}_G$ be a set of vertices and $\mathbb{E}_G \subseteq \mathbb{V}_G \times \mathbb{V}_G$ be a set of edges with positive edge weights. A sequence $p = \langle s_0, ..., s_n \rangle$ is called a path from s_0 to s_n in G iff $\forall s_i \in p : s_i \in \mathbb{V}_G \wedge \forall i \in [1; n] : (s_{i-1}, s_i) \in \mathbb{E}_G$.

We define the *length* of path $||p||$ as the number of elements in its sequence minus one, that is, in other words, its number of transitions. Let $s, g \in \mathbb{V}_G$ be vertices of G and $\mathbb{P}_{s \to g}$ be the set of all paths from s to g in G. Let *predecessor* $\prec_G \subseteq \mathbb{V}_G \times \mathbb{V}_G$ and *successor* $\succ_G \subseteq \mathbb{V}_G \times \mathbb{V}_G$ be relations on vertices of a graph, such that $a \prec_G b \Leftrightarrow \exists p \in \mathbb{P}_G : p = \langle a, ..., b \rangle$ and $a \succ_G b \Leftrightarrow \exists p \in \mathbb{P}_G : p = \langle b, ..., a \rangle$. We denote their respective complements $\not\prec_G = (\mathbb{V}_G \times \mathbb{V}_G) \setminus \prec_G$ and $\not\succ_G = (\mathbb{V}_G \times \mathbb{V}_G) \setminus \succ_G$. Furthermore, let $v : \mathbb{E}_G \to \mathbb{R}^+$ be a function mapping each edge to a postive real, from here on referred to as the *value function*. In some notational abuse, we define $v(p)$ for some path in G as the sum of the values of edges in p and call $v(p)$ the *value* of a path.

For some value $T \in \mathbb{R}$ we call a path $p \in \mathbb{P}_{s \to g}$ whose value $v(p)$ is closest to T a *target value path* with respect to s,g and T and denote the set of these paths $\mathbb{P}^T_{s \to g}$. Formally,

$$\mathbb{P}^T_{s \to g} = \underset{p \in \mathbb{P}_{s \to g}}{\operatorname{argmin}} |v(p) - T| \tag{1}$$

2.2. Target Value Search Algorithm

In the following, we describe a non-deterministic version of target value search, that returns some target value path $p^T_{s \to g} \in \mathbb{P}^T_{s \to g}$. Extension to a deterministic version is straightforward and will be sketched out where it differs from the following considerations. Formally,

$$p^T_{s \to g} \in \underset{p \in \mathbb{P}_{s \to g}}{\operatorname{argmin}} |v(p) - T| \tag{2}$$

A *target value path* set in regards to s, g and T can be straightforwardly computed in the following way: one can iteratively or recursively construct the $\mathbb{P}_{s\to g}$ set and evaluate (1). Due to the finiteness of G, the set $\mathbb{P}_{s\to g}$ will be finite $\forall s, g \in \mathbb{V}_G$, and each path $p \in \mathbb{P}_{s\to g}$ will be of finite lengh, so the set can always be constructed and thus, (1) can be evaluated.

Note that argmin is a function $2^{\mathbb{S}} \times (\mathbb{S} \to \mathbb{R}) \to 2^{\mathbb{S}}$. Let $\delta : \mathbb{P} \times \mathbb{R} \to \mathbb{R}$ be a function defined as the difference between the value of some path to some value $\delta(p, t) = v(p) - t$. Then the set of *target value paths* can be recursively defined as,

$$\mathbb{P}^T_{s\to g} = \begin{cases} \{\langle g \rangle\} & \text{if} \quad s = g \\ \emptyset & \text{if} \quad g \not\succ_G s \\ & \quad \wedge\, s \neq g \\ s \circ \underset{p' \in \bigcup_{(s,s')\in E_G} \mathbb{P}^{T^{s'}}_{s'\to g}}{\operatorname{argmin}} |\delta(p', T^{s'})| & \text{otherwise} \end{cases} \tag{3}$$

with $T^{s'} = T - v(\langle s, s' \rangle)$ target value minus the value of the path $\langle s, s' \rangle$. The mapping $\circ : \mathbb{V} \times 2^{\mathbb{P}} \to 2^{\mathbb{P}}$ is defined for (vertex, set of paths) pairs, such that there are edges between vertex and the first element of each path in the set defined in G. It maps those pairs to the set comprising of the concatenations of s and each path, resulting in a path in G. Equation 3 shows the recursive definition of $\mathbb{P}^T_{s\to g}$. The set of target value paths, is either the singleton $\{\langle g \rangle\}$ iff $s = g$, the empty set $\emptyset$ iff $g \not\succ_G s$ (this case ensures, that $\mathbb{P}^T_{s\to g}$ is defined for all pairs of vertices $s, g \in \mathbb{V}_G$) or the set of paths resulting from concatenating to s the "best" (with respect to the respective target values $T^{s'}$ of the successors) completions in G. Note that for paths $p = \langle s_0, s_1, ..., s_n \rangle, p' = \langle s_1, ..., s_n \rangle$ and values $t, t' = t - v(\langle s_0, s_1 \rangle)$, $\delta(p, t) = \delta(p', t')$. Hence, in the third case, the optimal completions can always be found in the union of *target value path* sets of the immediate successors s', g and their respective target value $T - v(\langle s, s' \rangle)$. Hence, the definition is tree recursive in the first argument of argmin. It remains to be shown, that this recursion is guaranteed to terminate. As G is a finite DAG, it follows that all sets of paths between two vertices of the graph are finite $\forall s, g \in \mathbb{V}_G, \forall p \in \mathbb{P}_{s\to g} : |\mathbb{P}_{s\to g}| < \infty$ and that all paths in these sets are finite $\forall s, g \in G, \forall p \in \mathbb{P}_{s\to g} : ||p|| < \infty$. Furthermore, if a vertex v' is an immediate successor of some vertex v and v'' is a common successor to both, it follows that longest path form v to v'' is at least one step longer than the longest path from v' to v'' and there is at least one successor to v such that its longest path to g is exactly one step shorter $\prec g \to \forall s' \succ s : \operatorname{argmax}_{p\in\mathbb{P}_{s\to g}} ||p|| = \operatorname{argmax}_{p'\in\mathbb{P}_{s'\to g}} ||p'|| + 1$. Finally, if the longest path in some $\mathbb{P}_{s\to g}$ is 1, it follows that s equals g ($\operatorname{argmax}_{p\in\mathbb{P}_{s\to g}} ||p|| = 1 \to s = g$). From these properties it follows that the recursive depth of the decomposition of $\mathbb{P}^T_{s\to g}$ is bounded by the (finite) length of the longest path in $\mathbb{P}_{s\to g}$ ($max_{p\in\mathbb{P}_{s\to g}} ||p||$) and is guaranteed to terminate. It follows, that $p^T_{s\to g}$ can be calculated straightforwardly through the above decomposition.

Unfortunately, due to the tree recursive nature of this algorithm, the computation can be quite expensive, especially on graphs with high fan-out. For example, the graphs used in the experiments section (roughly 40 nodes, avg. fan-out of 3) result in a recursion tree of about 20000 nodes. However, there is a way to significantly reduce the costs for computing the *target value path* set. To this end, we first have to introduce some concepts and lay out a simple algorithm the *target value search* problem. Assume we are given an extract of some graph $G = (\mathbb{V}_G, \mathbb{E}_G)$ as shown in Figure 1. We define the *successor graph* S^G_v of a vertex $v \in G$ as

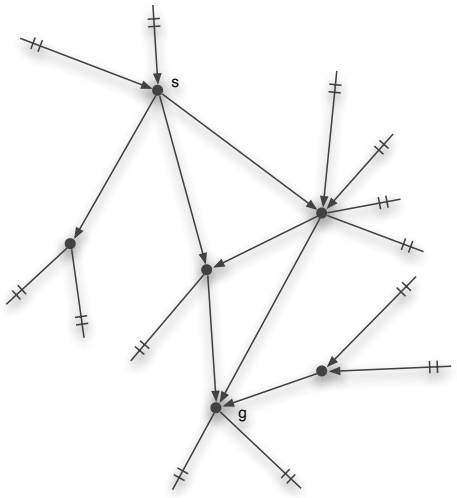

Figure 1: Extract of some graph G

the subgraph comprising of all successors of v in G and all edges between these successors in G. Formally,

$$\begin{aligned} S^G_v &= (\mathbb{V}_{S^G_v}, \mathbb{E}_{S^G_v}) \\ &= (\{v \in \mathbb{V}_G | v \succ_G s\}, \\ &\qquad \{(v_1, v_2) | v_1, v_2 \in \mathbb{V}_{S^G_v} \wedge (v_1, v_2) \in \mathbb{E}_G\}) \end{aligned} \tag{4}$$

Figure 2 shows the successor graph S^G_s of the extract of some graph G.

Analogously, we define the *predecessor graph* P^G_v of a vertex $v \in G$ as the subgraph comprising of all predecessors of v in G and all edges between these predecessors in G.

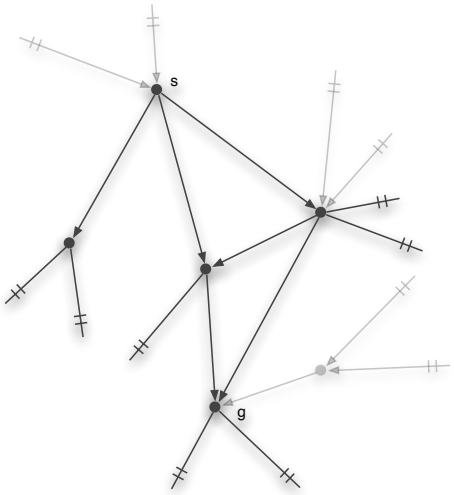

Figure 2: Successor graph of s

Formally,

$$\begin{aligned} P_v^G &= (\mathbb{V}_{P_v^G}, \mathbb{E}_{P_v^G}) \\ &= (\{v \in \mathbb{V}_G | v \prec_G s\}, \\ &\quad \{(v_1, v_2) | v_1, v_2 \in \mathbb{V}_{P_v^G} \wedge (v_1, v_2) \in \mathbb{E}_G\}) \end{aligned} \quad (5)$$

Figure 3 shows the predecessor graph P_g^G of the extract of some graph G.

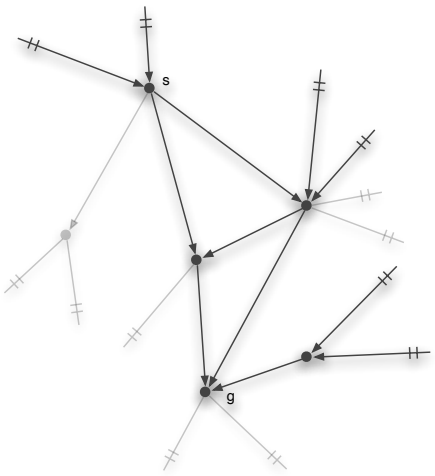

Figure 3: Predecessor graph of g

Finally, we define the *connection graph* $C^G_{v_1 \to v_2}$ of vertices $v_1, v_2 \in G$ as the intersection of the successor graph of v_1 in G and the predecessor graph of v_2 in G. Formally,

$$\begin{aligned} C^G_{v_1 \to v_2} &= (\mathbb{V}_{C^G_{v_1 \to v_2}}, \mathbb{E}_{C^G_{v_1 \to v_2}}) \\ &= (\mathbb{V}_{S^G_{v_1}} \cap \mathbb{V}_{P^G_{v_2}}, \mathbb{E}_{S^G_{v_1}} \cap \mathbb{E}_{P^G_{v_2}}) \end{aligned} \quad (6)$$

The resulting connection graph from vertex s to vertex g is shown in Figure 4.

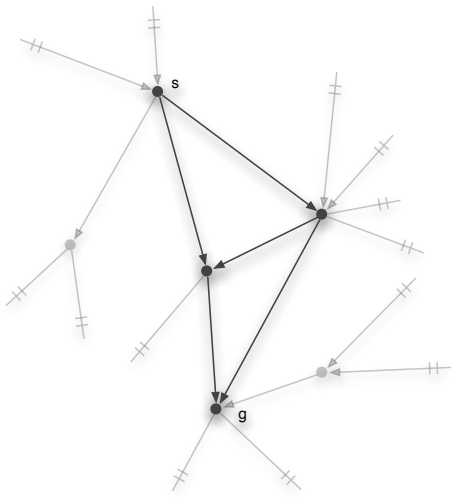

Figure 4: Connection graph $C^G_{s \to g}$ of two vertices s and g

The above, decomposition algorithm can be thought of as expanding the connection graph $C^G_{s \to g}$ into a directed tree, beginning with (s, T) as its root node, iteratively generating for each node (n, t) a child for each successor n' of n in $C^G_{s \to g}$, with the node labeled $(n', t - v((n, n'))$. Due to the structure of $C^G_{s \to g}$ expansion will end with a tree where all leaves are labeled with g and some value t_i, due to g being (by construction) the only node without successors in $C^G_{s \to g}$. These leaf nodes each represent one of the possible paths between s and g with their respective t_i denoting their absolute difference to the target value, or formally the absolute value of the $|\delta|$-function of the path in regards to T. Consequently, the paths from the root node of the expansion tree to the leaves with minimal $|t_i|$ comprise the target value path set $\mathbb{P}^T_{s \to g}$. See Fig 5 for an example. Listing 1 gives the pseudo code for the *tvs* algorithm.

Algorithm 1: tvs(G,s,g,T)

```
begin
    C = conGraph(G,s,g);
    Tree = (s,T);
    expand(Tree.root,C);
    return T.bestSolutions();
end
```

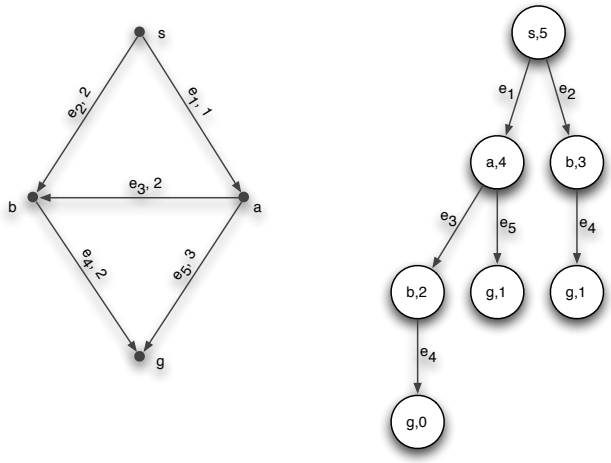

Figure 5: Connection graph with edge weights and its corresponding expansion tree for a target value of 5. The *target value paths* set is $\mathbb{P}^T_{s\to g} = \{\langle s, a, b, g\rangle\}$

Algorithm 2: expand((n,t),C)

```
begin
    foreach n' : C.succOf(n) do
        t' = t-v(n,n');
        (n,t).addChild(n', t');
        expand(n',t');
end
```

2.3. Heuristic Target Value Search Algorithm

Let $l : \mathbb{V}_{C^G_{s\to g}} \to \mathbb{R}^+_0$ and $u : \mathbb{V}_{C^G_{s\to g}} \to \mathbb{R}^+_0$ be functions such that $\forall n \in \mathbb{V}_{C^G_{s\to g}} : l(n) \leq \min_{p\in\mathbb{P}_{n\to g}} v(p) \wedge u(n) \geq \max_{p\in\mathbb{P}_{n\to g}} v(p)$ some lower bound. The nodes' T values during generation of the above tree represent the target values for their respective suffix paths. If for a given node (n, t), $t \leq l(n)$ holds, the best completion for this node is the shortest path in $\mathbb{P}_{n\to g}$, as each other completion will result in a larger offset from the target value. In the analogous case of $t \geq u(n)$ the best completion is the longest path in $\mathbb{P}_{n\to g}$. Thus, if either of these conditions hold for a node, graph expansion for that node can be stopped and its completion can be calculated by a shortest (longest) path search in $L_{n\to g}$. In both cases this can be done efficiently using the A^*-Algorithm with heuristics l and u respectively. Note as $L_{n\to g}$ is a subgraph of G it will also be cycle free, thus longest path search can be done with a modified A^*.

Consequently, the *htvs* algorithm operates in two phases. The first phase is graph expansion, where successors for nodes (n, t) are generated unless either the leaf level is reached (i.e. $n = g$), or t falls out of the $[l(n), u(n)]$ interval. These nodes are stored in a candidate list for later processing. At the end of this phase, the algorithm determines the leaf node with the minimum t value. The algorithm should terminate as soon as no better path can be found, so in case some generated leaf node is optimal (i.e. its t value is zero), the algorithm terminates and returns the corresponding path. Otherwise the algorithm advances to phase two, where it marks the best solution found in phase 1 (or ∞, if the leaf level was not reached) and searches the candidate list for pairs $(n, t_1), (n, t_2)$, such that both t_1 and t_2 are lower (resp. greater) than $l(n)$ (resp. $u(n)$). In such cases the node with the larger (resp. lower) t is pruned, as the corresponding path will always be worse. Finally it initializes, for each node (n, t) in the retained set, a shortest (respectively longest) path A^* search from n to g in their particular graph $C^G_{n\to g}$, with initial cost set to the corresponding t_i. Note, that these searches are done in the much smaller graph space, as opposed to the tree expansion in prefix space. Once again the $C^G_{n\to g}$ DAGs are cycle free, so A^* longest path searches are straightforward. Now, in each step the algorithm first checks the termination condition: is the current best solution's t value lower or equal then the intermediate t values of all the first elements in the open queues? The intermediate t values is the f function value of the A^* search plus the t value of the original candidate node. If this condition is satisfied, there can be no better path and the algorithm returns the current best solution. Otherwise the search with the lowest intermediate t value is advanced (one node expanded). Should the selected A^* terminate, its solution is compared to the current best solution, which it eventually displaces. If all A^* searches terminate, the current best solution is returned. Note that *(*htvs) is nondeterministic, i.e. it only calculates a single representative of $\mathbb{P}^T_{n\to g}$. However, extension to a deterministic version is straightforward. Listing 3 gives the pseudo code for this *heuristic target search* algorithm.

Algorithm 3: htvs(G,s,g,T,l,u)

```
begin
                                        /* Phase I */
    C = conGraph(G,s,g);
    Tree = (s, T);
                                         /* Main I */
    expandH(Tree.root,C, Candidates);
    if Tree.hasSolution() then
        Best = Tree.bestSolutions().some();
    else
        Best = (<>,infinity);
                                       /* Phase II */
    Candidates.pruneDuplicates();
    foreach (n,t) : Candidates do
        if t <= l(n) then
            A*[].add(new Amin*(n,g), new Offset(t));
        else
            A*[].add(new Amax*(n,g), new Offset(t));
                                         /* Main II */
    while A*[].hasCandidate() do
        if Best.t <= A*[].getBestIntermediateT() then
            return Best;
        A*[].advanceBestIntermediateT();
        if A*[].searchFinished() &&
        A*[].lastFinishedSearch().t < Best.t then
            Best = A*[].lastFinishedSearch();
    return Best;
end
```

Algorithm 4: expandH((n,t),Candidates,C,l,u)

```
begin
    if t<=l(n) || t>=u(n) && (C.hasSucc(n)) then
        Candidates.add((n,t));
    else
        foreach n' := C.succOf(n) do
            t' = t-v(n,n');
            (n,t).addChild(n', t');
            expandH(n',t');
end
```

2.4. Heuristic for the Target Value Search

In A^* search, heuristics are generally created by hand using deep insight into the domain. There are a number of methods for generating heuristics automatically. One method relies on using dynamic programming to compute actual path completion costs [1]. Complete dynamic programming would solve the problem. A novel sparse form of dynamic programming, however provides accurate lower and upper bounds for each vertex.

The graph structure and edge values can be used to compute a search heuristic. The idea is to calculate value bounds for each vertex n of the graph for the path leading to the goal vertex g that goes through n. We construct the heuristic by exploring the graph similar to breadth-first search (BFS). Intuitively, we start from the goal vertex g and examine all the immediate predecessor vertices by computing the bound interval for their path value to g. Then for each of those immediate predecessor vertices, we examine their immediate predecessor vertices recursively, and so on, until we have examined the entire graph.

The data structures used to compute these bounds are a queue of vertices and a map $map(n) = (l(n), u(n))$, where the keys are vertices and the values are intervals. For each vertex n the associated interval $(l(n), u(n))$ represents the best value bounds for paths from n to g with a lower bound $l(n)$ and an upper bound $u(n)$.

We set the bounds of the interval associated with g to zero because the value of a zero length path is zero. All other bounds are initialized to positive infinity for the lower bound and zero for the upper bound. We start the construction of the heuristic by putting the goal vertex g in the queue. Then we continue to pop and examine the first vertex from the queue until the queue is empty. The examination of a vertex n includes two operations: (1) add all the immediate predecessors of vertex n to the end of the queue, and (2) update their bounds. The bounds update for a predecessor vertex m is defined as follows:

$$
\begin{aligned}
l(m) = \quad & \min(v(m,n) + l(n), l(m)) \\
u(m) = \quad & \begin{cases} \max(v(m,n) + u(n), u(m)) & \text{if } u(m) < T + l(m) \\ \infty & \text{otherwise} \end{cases}
\end{aligned}
\tag{7}
$$

If the queue is empty, the bounds for every vertex of the graph have reached a fixed point. The computed map has the lower and upper bounds for the path value. We can use this map as l and u functions to guide the search.

3 Motivating Application: Diagnosis

As part of our group work on self-aware, planner-driven systems [3] we have designed and built the modular redundant printing engines illustrated in Figure 6. The system is controlled by a model-based planner [2]. The model of the systems describes all the components in the system, the connections between the components and all the actions a component can take. The task of the planner is to find the sequence of actions, called plan, which will move sheets through the system to generate the requested output.

Expanding this work on model-based controlled system we introduced a framework which integrates planning and

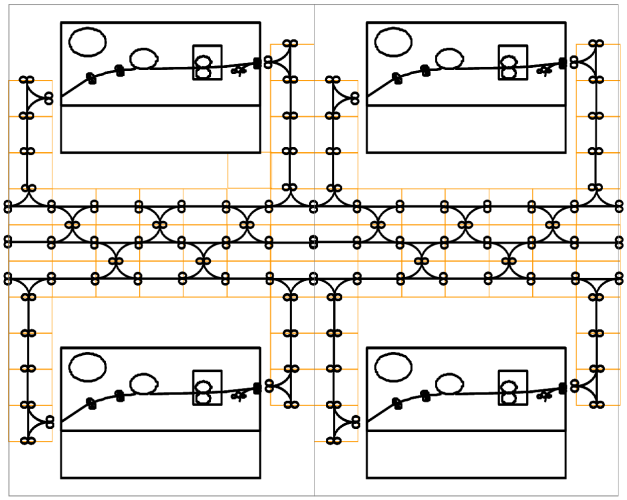

Figure 6: Model of PARC's prototype highly redundant printer. It consists of two towers each containing 2 printers (large rectangles). Sheets enters on the left and exit on the right. Dark black edges with small rollers represents a possible paper path. There are three main paper (horizontal) highways within the fixture. The fixture incorporates 2 types of media handling modules represented by small lighter edge rectangles. The motivation for this design is to continue printing even if some of the print engines fail or some of the paper handling modules fail or jam.

diagnosis to optimize production for long-run productivity called 'Pervasive Diagnosis' [5].

4 Pervasive Diagnosis

Pervasive diagnosis is a new paradigm in which production is actively manipulated to maximize diagnostic information. Active diagnosis and production can therefore occur *simultaneously* leading to higher long run productivity than passive diagnosis or alternating active diagnosis with production.

The integration of diagnostic goals in the production strategy results in *informative production*. The primary objective in informative production is to continue production. Under the assumption that there are various ways to achieve the production goals, informative production simultaneously maximizes diagnostic information. The literature describes different types of production such as simple production, time efficient production, cost efficient production and, robust production. All of those share the primary objective of achieving production but differ in the way they approach the goal. In simple production any strategy that achieves the production goal qualifies. In all other approaches the set of production strategies, those which achieve the production goals, are ranked by a secondary objective function and the best production strategy dominates. For example in time efficient production, strategies are ranked by cost and the most cost efficient production strategy dominates. Similar to other production strategies informative production ranks the set of plans that achieve production goals by their potential information gain and selects the most promising strategy.

5 System Architecture

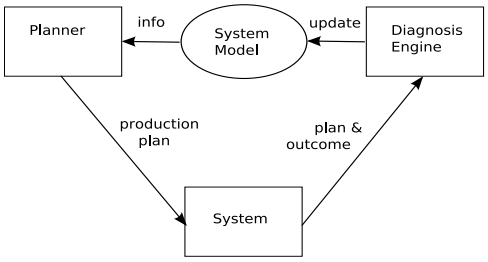

Figure 7: Overall System Architecture

Figure 7 illustrates the basic architecture of the system. The basic task of the planner is to schedule sheets through the reprographic machine. The extended task of the diagnosis engine is to estimate action failure probabilities and provide diagnostic guidance to the planner. Both the planner and diagnosis engine operate with a common model of the machine state.

The reprographic machines receive a continuous stream of print jobs. Each print job consists of a sequence of sheets of paper. The planner is to translates job requests into optimal (optimal to a given objective function) production plans. These plans p consisting of a sequence of actions $a_1, a_2, \ldots, a_n$ drawn from A_{sys}. We denote the set of unique actions in a plan $A_p = \bigcup_i \{a_i\}$. Executing an action potentially changes the system state. We assume *catastrophic* failures such that a plan fails if one or more of it's action fail. Due to physical or economical constraints executing a plan p results in a single observable, the plan outcome or observation O. In keeping with the diagnosis literature, we define two outcomes: the abnormal outcome, denoted $ab(p)$, in which the plan fails to achieve its production goal and the *not* abnormal outcome, denoted $\neg ab(p)$, in which the plan does achieve the production goal.

Based on the continuous stream of plan and observation pairs the diagnosis engine estimates the action failure probabilities which we denote $\Pr(ab(a))$. We apply Bayes' rule to estimate the hypothesis probabilities:

$$Pr(ab(a)|o) = \alpha Pr(o|ab(a))Pr(ab(a))$$

In a forthcoming paper, we introduce a compact representation and an update algorithm for a more general hypothesis space. For this paper we assume that the diagnosis engine estimates the action failure probabilities $Pr(ab(a)|o)$ for each action a.

6. Search for Highly Diagnostic Plans

The objective of pervasive diagnosis is to use the diagnosis engine's beliefs to influence production plans to gain additional information about the condition of the system. A plan is said to be informative, if it contributes information to the diagnosis engine's beliefs. We can measure this formally as the mutual information between the system beliefs $\Pr(ab(a))$ and the plan outcome conditioned on the plan executed, $I(ab(a); O|P = p)$. The mutual information is defined in terms of entropy or uncertainty implied by a probability distribution. A uniform distribution has high uncertainty and a deterministic one low uncertainty. An informative plan reduces the uncertainty of the system's beliefs. Intuitively, plans with outcomes that are hard to predict are the more informative. If we know a plan will succeed with certainty, we learn nothing by executing it. In a forthcoming paper, we explain how to calculate the optimal amount of uncertainty T a diagnosis engine should seek in a diagnosis plan in order to maximize information. In the case of persistent faults, the optimal uncertainty in outcome is $T = 0.5$. In the intermittent case, $0.36 \leq T \leq 0.5$. In this paper, we focus on the persistent fault case $T = 0.5$.

In order to apply target-value path search to pervasive diagnosis, we need a precise mapping of pervasive diagnosis onto the target-value problem. The planning objective in pervasive diagnosis is to find a plan that achieves the goal and is informative:

$$p^{opt} = \text{argmin}_{\text{achievesGoal}(p)\in\mathcal{P}}|Pr(ab(p)) - T|. \quad (8)$$

We formulate the problem as a target value path problem. Let the action set A_{sys} be the edge set E, let the failure probability of an action be the value of an action and let the optimal uncertainty about the outcome be the target value. In the case of persistent faults $T = 0.5$. Note that we don't want to count a failure probability twice because of the persistent fault assumption. In contrast to the costs in the target-value problem the failure probability of a plan doesn't increase in the persistent fault case if the same action appears multiple times in the plan. Therefore the problem becomes non-decomposable since we can't calculate the failure probability of the prefix plan and the suffix plan independently.

In the following sections of this paper, we discuss the single fault/single appearance scenario where there is no dependencies between $p_{I\to s}$ and $p_{s\to G}$ and $f(s)$ is naturally decomposable into $g(s)$ and $h(s)$.

6.1. Single Fault Diagnosis

In the simplest case we make two assumptions: (1) there is at most one fault in the system at any given time; the single fault assumption. (2) an action appears at most once in a single plan; the single action appearance assumption.

Because of the single fault and the single action appearance assumption a plan $p_{I\to G}$ decomposes into two partial plans $p_{I\to s}$ and $p_{s\to G}$ with two non-overlapping action sets $A_{p_{I\to s}}$ and $A_{p_{s\to G}}$.

Therefore the plan fault probability for a complete plan $p_{I\to G}$ can be decomposed into the plan fault probability of its decomposed partial plans $p_{I\to s}$ and $p_{s\to G}$ with s being any node visited by the plan $p_{I\to G}$:

$$Pr(ab(p_{I\to G})) = Pr(ab(p_{I\to s})) + Pr(ab(p_{s\to G})) \quad (9)$$

We can now apply the target-value path problem. First we construct the heuristic by recursively construct the heuristic starting from the goal state G. For each action $a_{s_m\to s_n}$ leading from state s_m to s_n we update the heuristic as shown in Equation 7 where vertices v corresponds to states s and the cost of an arc $c(v_m, v_n)$ corresponds to $Pr(ab(a_{s_m\to s_n}))$.

After the construction of the heuristic each state s is associated to an interval representing the lower and upper bound for the failure probability of plans from state s to the goal state G. The heuristic can then be used in the target-value search shown in Equation 3 where the cost for a path $c(p_{s\to v_n})$ corresponds to $Pr(ab(p_{I\to s}))$.

The search algorithm uses an admissible heuristic to guide the search and is therefore guarantee to return an optimal solution.

7 Experiments

In this section we present some results to illustrate how the algorithm performs on synthetic data and on a real-world diagnostic task. We implemented a graph generator constructing directed weighted graphs. A finite set of k costs is chosen for the edges in the graph. The actual cost of any edge will be randomly selected from this finite list of costs. The graphs consist of a start node s, a grid of interior nodes and a goal node g. The grid is parameterized by width W and length L. The start vertex s connects to every initial vertex of the grid and the goal vertex g is connected to every terminal vertex in the grid. The horizontal connections are present for every interior node. The diagonal connections between vertices in the grid are made with probability 'p'. An example of a generated grid appears in Figure 8. This graph is motivated by an application to modeling manufacturing processes.

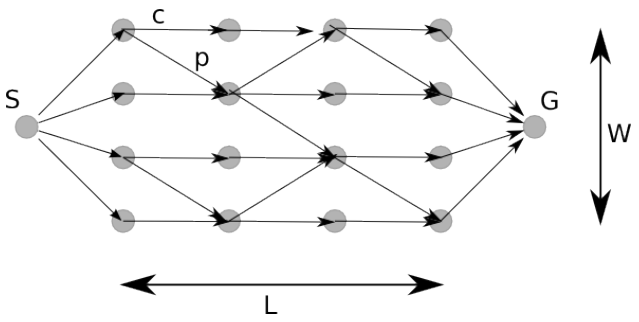

Figure 8: Schematic of graph parameterization

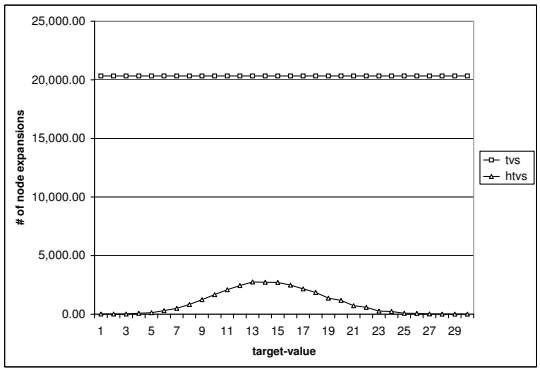

Figure 9: Number of node expansions for different target values.

For the experiments presented below, we set the width to 5 nodes, the length to 5 nodes and the interior node diagonals probability to 0.5. We plot the results for 30 different target values, each averaged over 15 runs. We run two different algorithms on this problem: the brute force *tvs* algorithm and the heuristic *htvs* algorithm.

In Figure 9 we can see that the heuristic algorithm is several orders of magnitude faster than a brute force search. As far as we know, this is the first heuristic search algorithm for this problem. We also note that the problem difficulty clearly varies with target. When the target value is close to the minimum cost (1.28) or maximum length plans (5.8), the problem is considerably easier. When the target value lies in between, the problem is more difficult. This makes sense, as the problem basically becomes an ordinary A^* search or inverse A^* search for the former cases.

8 Conclusions

We have presented a novel and efficient algorithm to find paths closest to a target value. We are unaware of any other efficient algorithms for this problem. This technology is key to enabling heuristic search for problems with target values such as recommendations to users and finding informative diagnosis plans goal-driven systems.

References

[1] J. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[2] Minh Do, Wheeler Ruml, and Rong Zhou. On-line planning and scheduling: An application to controlling modular printers. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, 2008, to appear.

[3] Markus P.J. Fromherz, Daniel .G. Bobrow, and Johan de Kleer. Model-based computing for design and control of reconfigurable systems. *The AI Magazine*, 24(4):120–130, 2003.

[4] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics (SSC)*, 4(2):100–107, 1968.

[5] Lukas Kuhn, Bob Price, Johan de Kleer, Minh Do, and Rong Zhou. Pervasive diagnosis: the integration of active diagnosis into production plans. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, 2008, to appear.

[6] Stephan Winter. Route specifications with a linaer dual graph. In *Symposium on Geospatial Theory*, 2002.