An A* perspective on deterministic optimization for deformable templates

A. L. Yuille and James M. Coughlan
Smith-Kettlewell Eye Research Institute,
San Francisco, CA 94115, USA.

Abstract

Many vision problems involve the detection of the boundary of an object, like a hand, or the tracking of a one-dimensional structure, such as a road in an aerial photograph. These problems can be formulated in terms of Bayesian probability theory and hence expressed as optimization problems on trees or graphs. These optimization problems are difficult because they involve search through a high dimensional space corresponding to the possible deformations of the object.

In this paper, we use the theory of A* heuristic algorithms [1] to compare three deterministic algorithms – Dijkstra, Dynamic Programming, and Twenty Questions, – which have been applied to these problems. We point out that the first two algorithms can be thought of as special cases of A* with implicit choices of heuristics. We then analyze the twenty questions, or minimum entropy, algorithm which has recently been developed by Geman and Jedynak [2] as a highly effective, and intuitive, tree search algorithm for road tracking. First we discuss its relationship to the focus of attention planning strategy used on causal graphs, or Bayes nets, [3]. Then we prove that, in many cases, twenty questions is equivalent to an algorithm, which we call A+, which is a variant of A*. Thus all three deterministic algorithms are either exact, or approximate, versions of A* with implicit heuristics.

We then discuss choices of heuristics for optimization problems and contrast the relative effectiveness of different heuristics. Finally, we briefly summarize some recent work [4],[5] which show that the Bayesian formulation can lead to a natural choice of heuristics which will be very effective with high probability.

1 Introduction

A promising approach to object detection and recognition assumes that we represent the objects by deformable templates [6, 7, 8, 9, 10, 11, 12, 13]. These deformable templates have been successfully applied in such special purpose domains as, for example, medical images [11], face recognition [6, 13, 9, 12], and galaxy detection [10]. Deformable templates specify the shape and intensity properties of the objects and are defined probabilistically so as to take into account the variability of the shapes and the intensity properties. Objects are typically represented in terms of tree or graph structures with many nodes.

There is a formidable computational problem, however, in using such models for the detection and recognition of complex objects from real images with background clutter. The very flexibility of the models means that we have to do tree/graph search over a large space of possible object configurations in order to determine if the object is present in the image. This is possible if the objects are relatively simple, there is little background clutter, and/or if prior knowledge is available to determine likely configurations of the objects. But it becomes a very serious problem for the important general case where the objects are complex and occur in natural scenes.

Statistical sampling algorithms [7, 10], and in particular the jump diffusion algorithm [8], have been very successful but do not, as yet, work in real time except on a restricted, though important, class of problems. Moreover, the only time convergence bounds currently known for such algorithms are extremely large.

This motivates the need to investigate algorithms which perform fast intelligent search over trees and graphs. Several deterministic algorithms of this type have recently been adapted from the computer science literature. Standard Dynamic Programming [14] has been applied to find the optimal solutions for a limited, though important, class of problems [15], [16],[17],[18], [19]. The time complexity is typically a low order polynomial in the parameters of the problem but this is still too slow for many applications. Dijkstra's algorithm [20] is a related greedy algorithm, which also uses the Dynamic programming principle. It has been successfully adapted [21] to some vision applications.

More recently, Geman and Jedynak have proposed the novel *twenty questions*, or minimal entropy, algorithm for tree search [2]. (Note that Geman and Jedynak refer to their algorithm as “active search” and use “twenty questions” in a more restricted sense. But we feel that twenty questions is more descriptive.) The twenty questions algorithm is intuitively very attractive because it uses information theory [22] to guide the search through the representation tree/graph. More recently, it has been rediscovered by Kontsevich and applied to analyzing psychophysical data [23]. The basic intuition of twenty questions has also been proposed as a general approach to vision in order to explain how the human visual system might be able to solve the enormous computational problems needed to interpret visual images and interpret visual scenes [24].

We examine these algorithms from the perspective of heuristic A* algorithms which have been studied extensively by Pearl [1]. These algorithms include a heuristic function which guides the search through the graph and choice of this function can significantly affect the convergence speed and the optimality. Both Dynamic Programming and Dijkstra are known to be special cases of A* with implicit choice of heuristics. Our analysis of twenty questions shows that it is closely related to a new algorithm, which we call A+, and which can be shown to be a variant of A*. (A+ is formulated almost identically to A* but differs from it by making more use of prior expectations during search).

The main difference between these algorithms, therefore, reduces to their implicit choice of heuristic. We analyze the properties of heuristics which are important for optimization problems involving Bayesian models. We show that these models motivate certain heuristics which, in related work [4],[5], can be proven (mathematically) to be very effective.

In section (2) we set up the framework by describing A* algorithms, the way they can represent problems in terms of trees and graphs, and their applications to deformable templates. In section (3) we describe Dynamic programming and Dijkstra from an A* perspective. In section (4) we introduce the twenty questions algorithm and show its relations to the focus of attention strategy used in causal, or Bayes, nets [3]. In section (5) we demonstrate a close relationship between twenty questions and A*. Finally, in section (6) we argue that better heuristics can be derived by exploiting the Bayesian formulation and briefly summarize our recent proofs of this claim [4],[5].

2 Framework: Deformable Templates and A*

The deformable templates we consider in this paper will only specify intensity properties at the boundaries of objects (i.e. they will ignore all regional intensity information). They will represent the template configuration X by a tree or a graph, see figures (2,3). There will be a probabilistic model $P(I|X)$ for generating an image I given the configuration $X \in \chi$ of the deformable template. This model will specify an intensity distribution at the template boundary and a background distribution elsewhere. There will also be a prior probability $P(X)$ on the set of possible configurations of the template. The goal is to find the most probable configuration of the template. In mathematical terms, we wish to the *a posteriori* estimate X^* such that:

$$X^* = \arg \max_{X \in \chi} P(I|X)P(X). \tag{1}$$

The distributions $P(X)$ and $P(I|X)$ are typically chosen to be Markov. This means that they have local interactions only occurring within finite, and usually small, neighbourhoods. These neighbourhood connections can be mapped directly onto connections in the graph [3]. For example, X might correspond to a configuration $x^1, x^2, \ldots, x^N$, where the x^i's denote points in the image lattice corresponding to positions of the deformable template elements (see figure (3)), and $P(X) = P(x^1) \prod_{i=1}^{N-1} P_i(x^{i+1}|x^i)$. This is a first order Markov chain where the probability of the position of a point depends only on the position of the preceding point, see [18] for an example of a higher order Markov chain. An important special case of distributions are those which are shift-invariant, i.e. so that $P_i(.|.)$ is independent of i. Such distributions are suitable for objects like roads where the statistical properties tend to be similar everywhere on the road and can, for example, be the typical generic smoothness priors used in vision. There are less suitable, however, to model objects like hands where the shape variations along the side of a finger are quite different from those at a fingertip. The imaging model $P(I|X)$ typically assumes that the image measurements are conditionally independent and hence can be expressed in the form $P(I|X) = \prod_{i=1}^{N} P_{on}(I_e(x^i)) \prod_{x \neq x^i, i=1,\ldots,N} P_{off}(I_e(x))$, where $P_{on}(I_e(x))$ is the probability of an image measurement $I_e(x)$ *if x lies on the deformable template* and $P_{off}(I_e(x))$ is the corresponding probability if x does not lie on the object boundary. We might, for example, let $I_e(x)$ be the response of an edge detector at point x and let $P_{on}(.)$ and $P_{off}(.)$ be the probability of these responses on and off the boundary. Such distributions can be learnt from training data [2], [25] and will imply that the object boundaries are likely to be places where the edge responses are strong.

The possible configurations in χ can be represented as trees or graphs. In figure (2) we show three types of representation. In figure (2A) we represent all positions in the spatial lattice, in the vertical axis, and the horizontal axis represents the stages of the template. Paths in the graph from left to right correspond to spatial trajectories. An advantage of this representation [18] is that it does not assume known starting or ending positions and so many trajectories can be represented in parallel. In figure (2B) an alternative representation, used in [2], assumes an initial starting segment and represents trajectories by a ternary tree. The representation in figure (2C) is based directly on the image lattice and is suitable for graph search algorithms such as Dijkstra, see for example [21], where the starting and ending position are known.

Observe that representation (A) can be restricted if the starting and ending position are known. If we also restrict the fan out factor to nearest neighbour on the image lattice then representation (A) can be re-expressed in form (C) (providing the prior is stationary, as is the case throughout most of this paper). Moreover, we see that the twenty questions' representation, (B), ignores the fact that paths can cross over each other. However, representations (A) and (B) are more general than (C) and, in particular, are more suited for priors which, like most deformable templates, are non-stationary.

In all three cases χ is typically very large and it is impractical to evaluate all its possibilities. Instead it is desirable to search the tree/graph in an efficient manner.

The A* graph search algorithm is an important search strategy developed in the artificial intelligence literature (e.g. see [26],[1],[27]). This algorithm is used to find the path of maximum cost between a start node A and a goal node B in a graph. The cost of a particular path is the sum of the costs of each edge traversed. The A* procedure maintains a tree of partial paths already explored, and computes a measure f of the "promise" of each partial path (i.e. leaf in the search tree). When computing these partial paths it makes use of the dynamic programming principle [14], illustrated in the

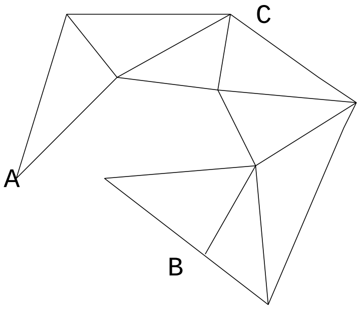

Figure 1: The A* algorithm tries to find the most expensive path from A to B. For a partial path AC the algorithm stores $g(C)$, the best cost to go from A to C, and an overestimate $h(C)$ of the cost to go from C to B.

next paragraph, to reduce the computation. The measure f for any node C is defined as $f(C) = g(C) + h(C)$, where $g(C)$ is the best cumulative cost found so far from A to C and $h(C)$ is an overestimate of the remaining cost from C to B. The closer this overestimate is to the true cost then the faster the algorithm will run. (A* can also be expressed in terms of cost minimization in which case the overestimate must become an underestimate. For example, the arcs might represent lengths of roads connecting various "cities" (nodes), and $h(C)$ could be estimated as the straight-line distance from city C to B.)

New paths are considered by extending the most promising node one step. The Dynamic Programming principle says that if at any time two or more paths reach a common node, then all the paths but the best (in terms of f) are deleted. (This provision is unnecessary in searching a tree, in which case there is only one path to each leaf.)

It is straightforward to prove that A* is guaranteed to converge to the correct result provided the heuristic $h(.)$ is an upper bound for the true cost from all nodes C to the goal node B. A heuristic satisfying these conditions is called *admissible*. Conversely, a heuristic which does not satisfy them is called *inadmissible*. For certain problems it can be shown that admissible A* algorithms have worst case time bounds which are exponential in the size of the goal path. Alternatively, for some of these problems it can be shown that there are inadmissible algorithms which converge with high probability close to the true solution in polynomial expected time [1]. Inadmissible heuristics may therefore lead to faster convergence in many practical applications.

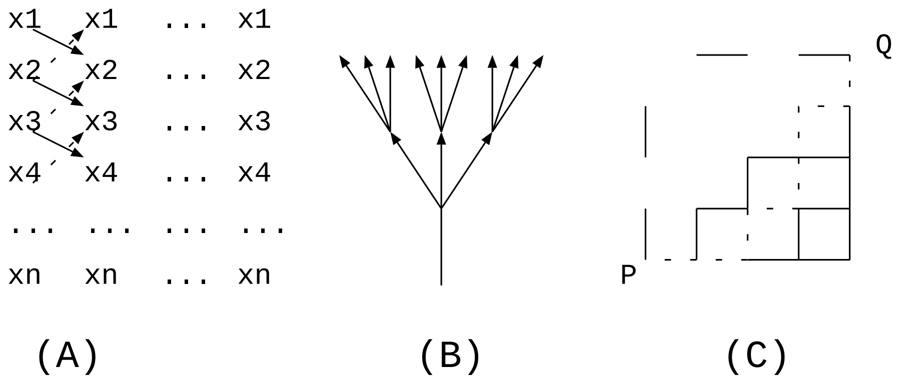

Figure 2: Examples of representation trees or graphs. On the left, fig (A), each columns represents the n lattice points $x1, ..., xn$ in the image and a trajectory from left to right through the columns is a possible configuration of the deformable template. The arrows show the fan out from each lattice point and put restrictions on the set of paths the algorithm considers. Fig (B) shows the ternary representation tree used in Geman and Jedynak. The arcs represent segments in the image some of which correspond to the road being tracked. The start and end point of each arc will be points in the image lattice. The root arc, at bottom, is specified by the user. To track the road the algorithm must search through subsequent arcs. In fig (C) we represent the lattice points in the image directly. A boundary segment of a person's silhouette can be represented by points on a lattice linking salient points P and Q (see Geiger and Tyng 1996). To determine the boundary involves searching for the best path through the graph.

3 Dijkstra and Dynamic Programming

Dijkstra and Dynamic Programming are two algorithms which have been successfully applied to detecting deformable templates in computer vision. See [21] for an example of Dijkstra. Dynamic Programming has been used by many people, see for example [18],[19]. We now discuss them from an A* perspective.

Dijkstra's algorithm [20] is a general purpose graph search algorithm which searches for the *minimum* cost path. It requires that the cost of going from node to node is always positive or zero. It can be converted into a maximization problem by changing the signs of the costs. If so, it becomes exactly like A* with heuristic $h(.) = 0$, see [1]. Because the node to node costs in the Dijkstra formulation are constrained to be positive this means that $h(.) = 0$ is an admissible heuristic (recall we change the signs of the costs when we convert to A*). Therefore we can consider Dijkstra to be an admissible version of A*[1]

Moreover, it should be recalled that admissible A* algorithms, and hence Dijkstra, can be slow. Moreover, see section (6) there is no guarantee that Dijkstra has chosen the best *admissible* heuristic. Dijkstra can be shown to have polynomial complexity in terms of the graph size which can be slow on a large graph (note that there is no contradiction with the fact that admissible A* can be exponential in the *solution size*). In fact, experimentally Dijkstra can slow down and require large memory resources when the target path is hard to find, due to noise, (Geiger – personal communication) and in such a case an inadmissible heuristic, or even an alternative admissible heuristic, may be preferable, see section (6).

We now turn to Dynamic Programming (DP). This algorithm acts on a representation of type given in figure (2A). The algorithm starts by finding the smallest cost path from any point in the first column to each point in the second column. At each point in the second column we store the cheapest cost (for the minimization formulation of the problem) to get there. DP then finds the smallest cost path from the first column to

[1] Observe that any admissible A* algorithm can be converted into Dijkstra by simply absorbing the heuristic into the cost of getting to the node (and changing the signs). I.e. by setting $\bar{g}(.) = g(.) + h(.)$. In this sense admissible A* and Dijkstra are equivalent. But this equivalence is misleading. By making the heuristic *explicit* A* can determine design principles for choosing it and allows analysis of the the effectiveness of different choices [1]. We will return to this issue in section (6).

each point in the third column. The dynamic programming principle means that these costs can be computed from the stored cheapest costs in the second column plus the costs of getting from the second column to the third. This procedure iterates as we go from column to column. It means that the complexity of DP, with this representation, will be of order NQM where N is the number of lattice points in the relative region of the image, Q is the fan out factor, and M is the (quantized) length of the deformable template.

Using DP in this fashion corresponds to a special case of A* acting on the graph made up of all the columns, see [26]. A heuristic is picked so that it is the same for all points on the column and is so large that A* always finds the best cost to all elements of a column before preceding to investigate the next column. This requires that the heuristic cost must be greater than any possible future rewards and hence this is an admissible A* algorithm.

We can contrast DP with a more general A* algorithm which, by using alternative heuristics, does not search in this militarist column by column strategy. A* will have to keep a record of all nodes which have been explored and maintain an ordered search tree of promising nodes to explore. By contrast, DP only needs to store the state of the last column and has no need to store a search list because the form of the heuristic implies that the nodes to be explored next are precisely those in the next column. This advantage is reduced, however, because DP is conservative and will continue to explore all elements of the columns even though many may have very low costs. This can be summarized by saying that DP does a breadth-first search while more general A* can perform a mixed depth-first and breadth-first strategy (depending on the heuristic and on the data). Another significant difference between the two algorithms is that A* requires sorting to determine which node to expand next and the cost of sorting can sometimes be prohibitive (by constrast DP needs no sorting).

One interesting aspect of Dynamic Programming is that for these type of problems its complexity may not increase significantly even if the start and finish points of the deformable template are unknown, see figure (3) from [18]. If the branching factor Q is large then the advantage of knowing the initial and final points rapidly becomes lost and the complexity is still $O(NQM)$.

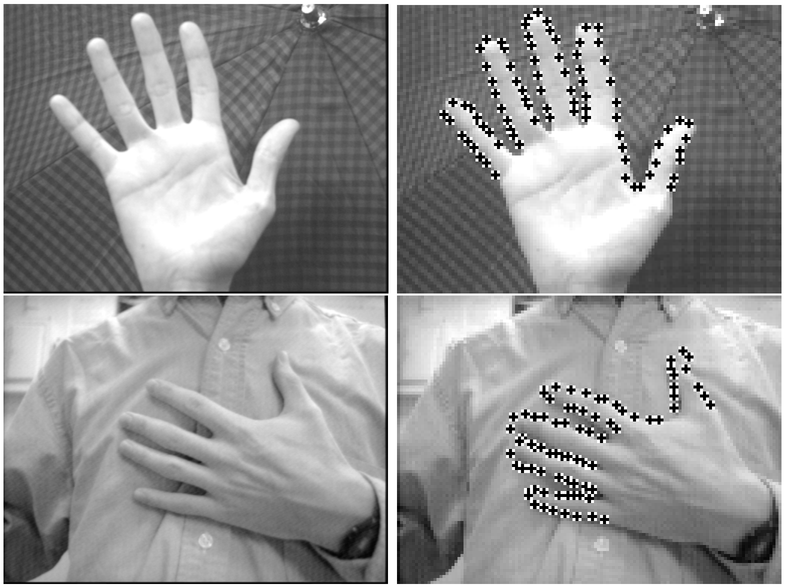

Figure 3: Examples of a deformable template using Dynamic Programming, from Coughlan (1995). The input images are shown on the left and the outputs on the right. The deformable templates, marked by the positions of the crosses, find the true maximum a posteriori estimate, hence locating the correct positions of the hands, without knowing the positions of the start and finish points.

4 Twenty Questions and focus of attention

Geman and Jedynak [2] design a deformable template model for describing roads in satellite images and a novel algorithm *twenty questions* for tracking them given an initial position and starting direction. The twenty question algorithm, using the representation (B), has an empirical complexity of order L [2].

They represent roads in terms of a tree of straight-line segments, called 'arcs', each approximately twelve pixels long with the root arc being specified by the user. Each arc has three ‘children’ which can be in the same direction as its ‘parent’ or may turn slightly to the left or the right by a small, fixed angle. In this way the arcs form a discretization of a smooth, planar curve with bounded curvature, see figure (2). A road is defined to be a sequence of L arcs (not counting the root arc). The set χ of all possible roads has 3^L members. The set A of all arcs has $(1/2)(3^{L+1} - 1)$ members. Many of the following results will depend only on the representation being a tree (i.e. not including closed loops) and so other branching patterns are possible, see figure (4).

Geman and Jedynak assume a priori that all roads are equally likely and hence $P(X) = 3^{-L}$ for each X. This is equivalent to assuming that at each branch the road has an equal probability of going straight, left or right. Much of the analysis below is independent of the specific prior probabilities $P(X)$, see figure (4).

They assume that there are a set of test observations which can be made on the image using non-linear filters. For each arc $a \in A$, a test Y_a consists of applying a filter whose response $y_a \in \{1, ..., 10\}$ is large when the image near arc a is road-like. (An arc a is considered road-like if the intensity variation of pixels along the arc is smaller than the intensity differences between pixels perpendicular to the arc.) The distribution of the tests $\{Y_a\}$ (regarded as random variables) depends only on whether or not the arc a lies on the road candidate X and the tests are assumed to be conditionally independent given X. Thus the probabilities can be specified by

$$
\begin{aligned}
P(Y_a|X) = p_1(Y_a) \; &if \; a \; lies \; on \; X, \\
P(Y_a|X) = p_0(Y_a) \; &otherwise. \qquad (2)
\end{aligned}
$$

The probability distributions $p_1(.)$ and $p_0(.)$ are determined by experiment (i.e. by running the tests on and off the road to gain statistics). These distributions overlap, otherwise the tests would give unambiguous results (i.e. "road" or "not-road") and the road could be found directly. The theoretical results we obtain are independent of the precise nature of the tests and indeed the algorithm can be generalized to consider a larger class of tests, but this will not be done in this paper.

The true road may be determined by finding the MAP estimate of $P(X| \; all \; tests)$. However, there is an important practical difficulty in finding the MAP: the number of possible candidates to search over is 3^L, an enormous number, and the number of possible tests is even larger (of course, these numbers ignore the fact that some of the paths will extend outside the domain of the image and hence can be ignored. But, even so, the number of possible paths is exorbitant). To circumvent this problem, Geman and Jedynak propose the *twenty questions* algorithm that uses an intelligent testing rule to select the

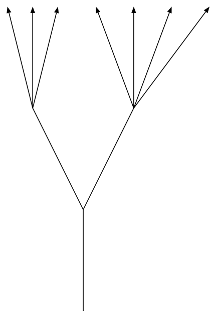

Figure 4: A variation of Geman and Jedynak's tree structure with a different branching pattern. The prior probabilities may express a preference for certain paths, such as those which are straight.

most informative test at each iteration.

They introduce the concept of partial paths and show that it is only necessary to calculate the probabilities of these partial paths rather than those of all possible road hypotheses. They define the set C_a to consist of all paths which pass through arc a. Observe, see figure (8), that this condition specifies a unique path from the root arc to a. Thus $\{X \in C_a\}$ can be thought of as the set of all possible extensions of this partial path. Their algorithm only needs to store the probabilities of certain partial paths, $z_a = P(X \in C_a| \text{ } \textit{test results})$, rather than the probabilities of all the 3^L possible road paths. Geman and Jedynak describe rules for updating these probabilities z_a but, in fact, the relevant probabilities can be calculated directly (see next section). It should be emphasized that calculating these probabilities would be significantly more difficult for general graph structures where the presence of closed loops introduces difficulties which require algorithms like dynamic programming to overcome [28].

The testing rule is the following: after having performed tests Y_{π_1} through Y_{π_k}, choose the next test $Y_{\pi_{k+1}} = Y_c$ so as to minimize the conditional entropy $H(X|b_k, Y_c)$ given by:

$$H(X|b_k, Y_c) = -\sum_{y_c} P(Y_c = y_c|b_k)\{\sum_{X} P(X|b_k, Y_c = y_c) \log P(X|b_k, Y_c = y_c)\}, \quad (3)$$

where $b_k = \{y_{\pi_1}, ..., y_{\pi_k}\}$ is the set of test results from steps 1 through k (we use capitals to denote random variables and lower case to denote numbers such as the results of tests). The conditional entropy criterion causes tests to be chosen which will be expected to maximally decrease the uncertainty of the distribution $P(X|b_{k+1})$.

We also point out that their strategy for choosing tests has already been used in Bayes Nets [3]. Geman and Jedynak state that there is a relationship to Bayes Nets [2] but they do not make it explicit. This relationship can be seen from the following theorem.

Theorem 1. *The test which minimizes the conditional entropy is the same test that maximizes the mutual information between the test and the road conditioned on the results of the proceeding tests. More precisely,* $\arg\min_c H(X|b_k, Y_c) = \arg\max_c I(Y_c; X|b_k)$.

Proof. *This result follows directly from standard identities in information theory [22]:*

$$I(Y_a; X|b_k) = H(X|b_k) - H(X|b_k, Y_a) = H(Y_a|b_k) - H(Y_a|X, b_k), \tag{4}$$

This maximizing mutual information approach is precisely the *focus of attention* strategy used in Bayes Nets [3], see figure (5). It has proven an effective strategy in medical probabilistic expert systems, for example, where it can be used to determine which diagnostic test a doctor should perform in order to gain most information about a possible disease [28]. Therefore the twenty questions algorithm can be considered as a special case of this strategy. Focus of attention, however, is typically applied to problems involving graphs with closed loops and hence it is difficult to update probabilities after a question has been asked (a test has been performed). Moreover, on graphs it is both difficult to evaluate the mutual information and to determine which, of many possible, tests will maximize the mutual information with the desired hypothesis state X.

By contrast, Geman and Jedynak are able to specify simple rules for deciding which tests to perform. This is because: (i) their tests, equation (2), are simpler than those typically used in Bayes Nets and (ii) their tree structure (i.e. no closed loops) makes it easy to perform certain computations.

The following theorem, which is stated and proven in their paper, simplifies the problem of selecting which test to perform. As we will show later, this result is also important for showing the relationship of twenty questions to A+. The theorem is valid for any graph (even with closed loops) and for arbitrary prior probabilities. It relies only on the form of the tests specified in equation (2). The key point is the assumption that roads either contain the arc which is being tested or they do not.

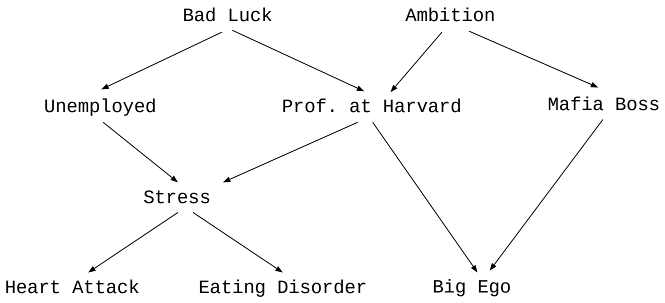

Figure 5: A Bayes Net is a directed graph with probabilities. This can be illustrated by a game show where the goal is discover the job of a participant. In this case the jobs are "unemployed", "Harvard professor" and "Mafia Boss". The players are not allowed direct questions but they can ask about causal factors – e.g. "bad luck" or "ambition" – or about symptoms – "heart attack", "eating disorder", "big ego". The focus of attention strategy is to ask the questions that convey the most information. Determining such questions is straightforward in principle, if the structure of the graph and all the probabilities are known, but may require exorbitant computational time if the network is large.

Theorem 2. *The test Y_c which minimizes the conditional entropy is the test which minimizes a convex function $\phi(z_c)$ where $\phi(z) = H(p_1)z + H(p_0)(1-z) - H(zp_1 + (1-z)p_0)$.*

Proof. *From the information theory identities given in equation (4) it follows that minimizing $H(X|b_k, Y_c)$ with respect to a is equivalent to minimizing $H(Y_c|X, b_k) - H(Y_c|b_k)$. Using the facts that $P(Y_c|X, b_k) = P(Y_c|X)$, $z_c = P(X \in C_c|b_k)$, $P(Y_c|b_k) = \sum_X P(Y_c|X)P(X|b_k) = p_1(Y_c)z_c + p_0(Y_c)(1-z_c)$, where $P(Y_c|X) = p_1(Y_c)$ if arc c lies on X and $P(Y_c|X) = p_0(Y_c)$ otherwise, we find that:*

$$H(Y_c|X, b_k) = \sum_X P(X|b_k)\{-\sum_{Y_c} P(Y_c|X)\log P(Y_c|X)\} = z_c H(p_1) + (1-z_c)H(p_0),$$
$$H(Y_c|b_k) = H(z_c p_1 + (1-z_c)p_0). \quad (5)$$

The main result follows directly. The convexity can be verified directly by showing that the second order derivative is positive.

For the tests chosen by Geman and Jedynak it can be determined that $\phi(z)$ has a unique minimum at $\overline{z} \approx 0.51$. For the game of twenty questions, where the tests give unambiguous results, it can be shown that the minimum occurs at $\overline{z} = 0.5$. (In this case the tests will obey $p_1(Y_c = y_c)p_0(Y_c = y_c) = 0$, $\forall y_c$ and this enforces that $H(z_c p_1 + (1 - z_c)p_0) = z_c H(p_1) + (1 - z_c)H(p_0) - z_c \log z_c - (1 - z_c)\log(1 - z_c)$ and so $\phi(z) = z \log z + (1 - z)\log(1 - z)$ which is convex with minimum at $z = 0.5$).

Thus the minimal entropy criterion says that we should test the next untested arc which minimizes $\phi(z_c)$. By the nature of the tree structure and the prior there can be very few (and typically no) untested arcs with $z_c > \overline{z}$ and most untested arcs will satisfy $z_c \leq \overline{z}$. Restricting ourselves to this subset, we see that the convexity of $\phi(.)$, see figure (6), means that we need only find an arc c for which z_c is as close to $\overline{z}$ as possible. It is straightforward to show that most untested arcs, particularly distant descendants of the tested arcs, will have probabilities far less than $\overline{z}$ and so do not even need to be tested (each three way split in the tree will introduce a prior factor 1/3 which multiplies the probabilities of the descendant arcs, so the probabilities of descendants will decay exponentially with the distance from a tested arc). It is therefore simple to minimize $\phi(z_c)$ for all arcs such that $z_c \leq \overline{z}$ and then we need simply compare this minimum to the values for the few, if any, special arcs for which $z_c > \overline{z}$. This, see [2], allows one to quickly to determine the best test to perform. Observe, that because the prior is uniform there may often be three or two arcs which have the same probability. To see this, consider deciding which arc to test when starting from the root node – all three arcs will be equally likely. It is not stated in [2] what their algorithm does in this case but we assume, in the event of a tie, that the algorithm picks one winner at random.

5 Twenty questions, A+ and A*.

In this section we define an algorithm, which we call $A+$, which simply consists of testing the most probable untested arc. We show that this is usually equivalent to twenty questions. Then we show that $A+$ can be re-expressed as a variant of A*.

The only difference between A* and A+ is that A+ (and twenty questions) makes use of prior expectations in an attempt to speed up the search. (Both A+ and twenty questions are formulated with prior probabilities which can be used to make these predictions). The difference in search strategies can be thought of, metaphorically, as the distinction between

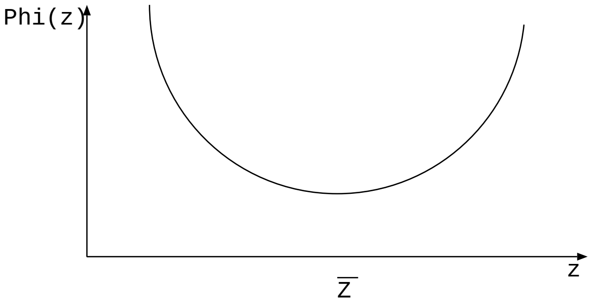

Figure 6: Test selection for twenty questions is determined by the $\phi(z)$ function. This is convex with at minimum at $\overline{z}$. Most untested arcs a with have probabilities z_a less than $\overline{z}$ and twenty questions will prefer to explore the most probable of these paths. It is conceivable that a few untested arcs have probability greater than $\overline{z}$. In this case they may or may not be tested. The exact form of the $\phi(.)$ function depends on specific details of the problem.

eugenics and breeding like rabbits. A* proceeds by selecting the graph node which has greatest total cost (cumulative and heuristic) and then expands *all the children of this node*. This is the rabbit strategy. By contrast, A+ selects the best graph node and then expands only *the best predicted child node*. This is reminiscent of eugenics. The twenty questions algorithm occasionally goes one stage further and expands a grandchild of the best node (i.e. completely skipping the child nodes). In general, if prior probabilities for the problem are known to be highly non-uniform, then the eugenic strategy will on average be more efficient than the rabbit strategy.

The algorithm $A+$ is based on the same model and the same array of tests used in Geman and Jedynak's work. What is different is the rule for selecting the most promising arc c on which to perform the next test Y_c. The arc c that is chosen is the arc with the highest probability z_c that satisfies two requirements: Test Y_c must not have been performed previously and c must be the child of a previously tested arc. For twenty questions the best test will typically be the child of a tested arc though occasionally, as we will describe later, it might be a grandchild or some other descendant.

Theorem 3. *A+ and Twenty questions will test the same arc provided $z_c \leq \overline{z}$ for all untested arcs c. Moreover, the only cases when the algorithms will differ is when A+*

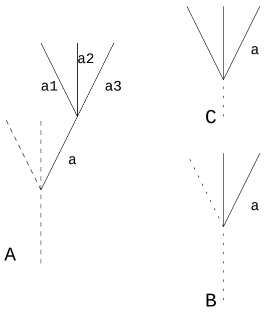

Figure 7: The three possible possibilities for A+'s preferred arc a where dashed lines represent tested arcs. In A, both a's siblings have been tested. In this case the twenty question algorithm might prefer testing one of a's three children or some other arc elsewhere on the tree. In cases B and C, at most one of a's siblings have been tested and so both twenty questions and A+ agree.

chooses to test an arc both siblings of which have already been tested.

Proof. *The first part of this result follows directly from Theorem 2: $\phi(z)$ is convex with minimum at $\overline{z}$ so, provided $z_c \leq \overline{z}$ for all untested c, the most probable untested arc is the one that minimizes the conditional entropy, see figure (6). The second part is illustrated by figure (7). Let c be the arc that A+ prefers to test. Since A+ only considers an arc c that is the child of previously tested arcs, there are only three cases to consider: when none, one, or two of c's siblings have been previously tested. In the first two cases, when none or one of c's siblings has been tested, the probability z_c is bounded: by $z_c < 1/3 < \overline{z}$ or by $z_c < 1/2 < \overline{z}$, respectively. Clearly, since c is the arc with the maximum probability, no other arc can have a probability closer to $\overline{z}$; thus arc c minimizes $\phi(z_c)$ and both algorithms are consistent. In the third case, however, when both of c's siblings have been tested, it is possible for z_c to be larger than $\overline{z}$. In this case it is possible that other arcs with smaller probabilities would lower ϕ more than $\phi(z_c)$. For example, if $\phi(z_c/3) < \phi(z_c)$, then the twenty questions algorithm would prefer any of c's (untested) children, having probability $z_c/3$, to c itself. But conceivably there may be another untested arc elsewhere with probability higher than $z_c/3$, and lower than $\overline{z}$, which twenty questions might prefer.*

Thus the only difference between the algorithms may occur when the previous tests will have established c's membership on the road with such high certainty that the conditional entropy principle considers it unnecessary to test c itself. In this case twenty questions may perform a "leap of faith" and test c's children or it may test another arc elsewhere. If twenty questions chooses to test c's children then this would make it potentially more efficient than A+ which would waste one test by testing c. But from the backtracking histogram in [2] it seems that testing children in this way never happened in their experiments. There may, however, have been cases when untested arcs are more probable than $\overline{z}$ and the twenty questions algorithm tested other unrelated arcs. If this did indeed happen, and the structure of the problem might make this impossible, then it is seems that twenty questions might be performing an irrelevant test. We expect therefore that A+ and twenty questions will usually pick the same test and so should have almost identical performance on the road tracking problem.

This analysis can be generalized to alternative branching structures and prior probabilities. For example, for a binary tree we would expect that the twenty questions algorithm might often make leaps of faith and test grandchildren. Conversely, the larger the branching factor then the more similar A+ and twenty questions will become. In addition, a non-uniform prior might also make it advisable to test other descendants. Of course we can generalize A+ to allow it to skip children too if the children have high probability of being on the path. But we will not do this here because, as we will see, such a generalization will reduce the similarity of A+ with A*.

Our next theorem shows that we can give an analytic expression for the probabilities of the partial paths. Recall that these are the probabilities z_a that the road goes through a particular tested arc a, see figure (8). (Geman and Jedynak give an iterative algorithm for calculating these probabilities). This leads to a formulation of the A+ algorithm which makes it easy to relate to A*. The result holds for arbitrary branching and priors.

Theorem 4. *The probabilities $z_a = P(X \in C_a | y_1, ..., y_M)$ of partial paths to an untested arc a, whose parent arc has been tested, can be expressed as:*

$$P(X \in C_a | y_1, ..., y_M) = \frac{1}{Z_M} \prod_{j=1}^{M_a} \frac{p_1(y_{a_j})}{p_0(y_{a_j})} \psi(a_j, a_{j-1}), \tag{6}$$

where $A_a = \{a_j : j = 1...M_a\}$ is the set of (tested) arcs lying on the path to a, see figure (8), and $\psi(a_i, a_{i-1})$ is the prior probability of arc a_i following arc a_{i-1} (a_0 is the

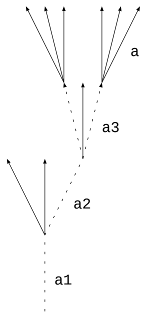

Figure 8: For any untested arc a, there is a unique path $a_1, a_2, ...$ linking it to the root arc. As before, dashed lines indicate arcs that have been tested.

initialization arc).

Proof. *Suppose a is an arc which has not yet been tested but which is a child of one that has. Assume we have test results $(y_1, ..., y_M)$, then there must be a unique subset $A_a = \{a_1, ..., a_{M_a}\}$ of tests which explore all the arcs from the starting point to arc a, see figure (8).*

The probability that the path goes through arc a is given by:

$$P(X \in C_a | y_1, ..., y_M) = \sum_{X \in C_a} P(X | y_1, ..., y_M) = \sum_{X \in C_a} \frac{P(y_1, ..., y_M | X) P(X)}{P(y_1, ..., y_M)}. \qquad (7)$$

The factor $P(y_1, ..., y_M)$ is independent of a and so we can remove it (we will only be concerned with the relative values of different probabilities and not their absolute values). Recall that the tests are independent and if arc i lies on, or off, the road then a test result y_i is produced with probability $p_1(y_i)$ or $p_0(y_i)$ respectively. We obtain:

$$P(X \in C_a | y_1, ..., y_M) \propto \sum_{X \in C_a} P(X) \{ \prod_{i=1,...,M : X \in C_i \bigcap C_a} p_1(y_i) \} \{ \prod_{i=1,...,M : X \notin C_i \bigcap C_a} p_0(y_i) \}$$

$$= \sum_{X \in C_a} P(X) \{ \prod_{i=1,...,M : X \in C_i \bigcap C_a} \frac{p_1(y_i)}{p_0(y_i)} \} \{ \prod_{i=1,...,M} p_0(y_i) \} \quad (8)$$

where the notation $X \in C_i \bigcap C_a$ means the set of all roads which contain the (tested) arc i and arc a. The final factor $\prod_i p_0(y_i)$ can be ignored since it is also independent of a.

Now suppose none of arc a's children have been tested. Then since the sum in equation (8) is over all paths which go through arc a this means that set of arcs $i : X \in C_i$ on the road X for which tests are performed must be precisely those in the unique subset A_a

going from the starting point to arc a. More precisely, $\{i = 1, ..., M : X \in C_i \bigcap C_a\} = A_a$. *Therefore:*

$$\prod_{i=1,...,M:x\in C_i \bigcap C_a} \frac{p_1(y_i)}{p_0(y_i)} = \prod_{i \in A_a} \frac{p_1(y_i)}{p_0(y_i)} = \prod_{j=1}^{M_a} \frac{p_1(y_{a_j})}{p_0(y_{a_j})}. \tag{9}$$

Now $\sum_{X \in C_a} P(X)$ *is simply the prior probability that the path goes through arc a. We can denote it by* P_a. *Because of the tree structure, it can be written as* $P_a = \prod_{1=1}^{M_a} \psi(a_i, a_{i-1})$, *where* $\psi(a_i, a_{i-1})$ *is the prior probability that the road takes the child arc* a_i *given that it has reached its parent* a_{i-1}. *If all paths passing through a are equally likely (using Geman and Jedynak's prior on the ternary graph) then* $\psi(a_i, a_{i-1}) = 1/3$ *for all a and we have:*

$$P_a \equiv \sum_{X \in C_a} P(X) = \frac{3^{L-|A_a|-1}}{3^L}, \tag{10}$$

where L is the total length of the road and $|A_a|$ *is the length of the partial path.*

Therefore in the general case:

$$P(X \in C_a | y_1, ..., y_M) = \frac{1}{Z_M} \prod_{j=1}^{M_a} \frac{p_1(y_{a_j})}{p_0(y_{a_j})} \psi(a_j, a_{j-1}), \tag{11}$$

where Z_M *is a normalization factor.*

We now show that A+ is just a simple variant of A*. We first design an admissible A* algorithm using the smallest heuristic which guarantees admissibility. Then we will show that A+ is a variant of A* with a smaller heuristic and hence is inadmissible.

To adapt A* to apply to the road tracking problem we must convert the ternary road representation tree into a graph by introducing a terminal node to which all the leaves of the tree are connected. We set the cost of getting to this terminal node from any of the leaves of the tree to be constant. Then deciding to go from one node to an adjacent node, and evaluating the cost, is equivalent to deciding to test the arc between these nodes and evaluating the test result.

It follows directly from Theorem 4, or see [2], that the best road is the one which maximizes the log of equation (11):

$$E(X) = \sum_i \{log \frac{p_1(y_{a_i})}{p_0(y_{ai})} + \log \psi(a_i, a_{i-1})\} \tag{12}$$

where $X = \{a_1, ..., a_L\}$ and $\psi(a_i, a_{i-1})$ is the prior probability that arc a_i follows a_{i-1}. Observe that the Z_m factor from equation (11) is the same for all paths so we have dropped it from the right hand side of equation (12).

By Theorem 4 again, the cost for a partial path of length M_a which terminates at arc a is given by:

$$g(a) = \sum_{j=1}^{M_a} \{\log \frac{p_1(y_{a_j})}{p_0(y_{aj})} + \log \psi(a_j, a_{j-1})\}. \tag{13}$$

To determine the smallest admissible heuristic for A*, we observe that the cost to the end of the road has a least upper bound of:

$$h(a) = (L - M_a)\{\Lambda_0 + \Lambda_p\}, \tag{14}$$

where $\lambda_0 = \max_y \log\{p_1(y)/p_0(y)\}$ and $\lambda_p = \max \log \psi(,.,)$ over all possible prior branching factors in the tree.

It is clearly possible to have paths which obtain this upper bound though, of course, they are highly unlikely because they require all the future path segments to have the maximal possible log likelihood ratio, we will return to this issue in section (6). The heuristic given by equation (14) is therefore the smallest possible admissible heuristic.

We can therefore define the admissible A* algorithm with smallest heuristic to have a cost f given by:

$$f(a) = g(a) + h(a) = \sum_{j=1}^{M_a} \{\log \frac{p_1(y_{a_j})}{p_0(y_{aj})} + \log \psi(a_j, a_{j-1})\} + (L - M_a)\{\Lambda_0 + \Lambda_p\}. \tag{15}$$

Observe that we can obtain Dijkstra by rewriting equation (15) as

$$f = \sum_{j=1}^{M_a} (\log \frac{p_1(y_{a_j})}{p_0(y_{a_j})} + \log \psi(a_j, a_{j-1}) - \Lambda_0 - \Lambda_p) + L\{\Lambda_0 + \Lambda_p\}, \tag{16}$$

and, because the length of all roads are assumed to be the same, the final term $L\{\Lambda_0 + \Lambda_p\}$ is a constant and can be ignored. This can be directly reformulated as Dijkstra with $g = \sum_{j=1}^{M_a} (\log \frac{p_1(y_{a_j})}{p_0(y_{a_j})} + \log \psi(a_j, a_{j-1}) - \Lambda_0 - \Lambda_p)$ and $h = 0$. (Though, strictly speaking, the term Dijkstra only applies if the terms in the sum are all guaranteed to be non-negative. Only with this additional condition is Dijkstra guaranteed to converge.)

The size of $\Lambda_0 + \Lambda_p$ has a big influence in determining the order in which paths get searched by the admissible A* algorithm. The bigger $\Lambda_0 + \Lambda_p$ then the bigger the overestimate cost, see equation (14). The larger the overestimate then the more the admissible A* will prefer to explore paths with a small number of tested arcs (because these paths will have overgenerous estimates of their future costs). This induces a *breadth first* search strategy [26] and may slow down the search.

We now compare A+ to A*. The result is summarized in Theorem 5.

Theorem 5. *A+ is an inadmissible variant of A*.*

Proof. *From Theorem 4 we see that A+ picks the path which minimizes equation (13). In other words, it minimizes the* $g(a)$ *part of A* but has no overestimate term. In other words, it sets* $h(a) = 0$ *by default. There is no reason to believe that this is an overestimate for the remaining part of the path. Of course, if* $\Lambda_0 + \Lambda_p \leq 0$ *then* $h(a) = 0$ *would be an acceptable overestimate. For the special case considered by Geman and Jedynak this would require that* $\max_y \log\{p_1(y)/p_0(y)\} - \log 3 \leq 0$. *For their probability distributions it seems that this condition is not satisfied, see figure 5 in [2].*

This means that A+, and hence twenty questions, are typically suboptimal and are not guaranteed to converge to the optimal solution. On the other hand, admissible A* uses upper bounds means that it prefers paths with few arcs to those with many, so it may waste time by exploring in breadth. We will return to these issues in the next section.

6 Heuristics

So far we have shown that Dynamic Programming, Dijkstra and Twenty Questions are either exact, or approximate, versions of A*. The only difference lies in their choice of heuristic and their breeding strategies (rabbits or eugenics). Both DP and Dijkstra choose admissible heuristics and are therefore guaranteed to find the optimal solution. The downside from this is that they are rather conservative and hence may be slower than necessary. By contrast, Twenty Questions is closely related to A+ which uses an inadmissible heuristic. It is not necessarily guaranteed to converge but empirically is very fast and finds the optimal solution in linear time (in terms of the solution length).

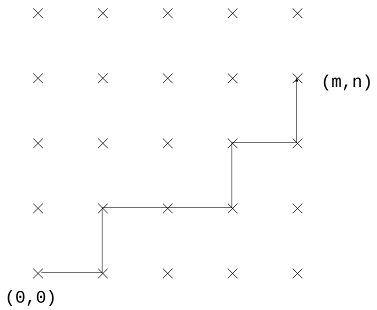

Figure 9: In the example from Pearl, the goal is to find the shortest path in the lattice from $(0, 0)$ to (m, n). The choice of heuristic will greatly influence the search strategy.

The choice of heuristics is clearly very important. What principles can be used to determine good heuristics for a specific problem domain?

An example from Pearl [1] illustrates how different admissible heuristics can affect the speed of search. Pearl's example is formulated in terms of finding the shortest path in a two-dimensional lattice from position $(0, 0)$ to the point (m, n), see figure (9).

The first algorithm Pearl considers is Dijkstra, so it has heuristic $h(.,.) = 0$ which is admissible since the cost of all path segments is positive. It is straightforward to show that this requires us to expand $Z_a(m, n) = 2(m + n)^2$ nodes before we reach the target (by the nature of A* we must expand all nodes whose cost is less than, or equal to, the cost of the goal node).

The second algorithm uses a shortest distance heuristic $h(x, y) = \sqrt{(x - m)^2 + (y - n)^2}$ which is also admissible. In this case the number of nodes expanded, $Z_b(m, n)$, can be calculated. The expression is complex so we do not write it down.

The bottom line, however, is that the ratio of $Z_b(m, n)/Z_a(m, n)$ is always less than 1 and so the shortest distance heuristic is preferable to Dijkstra for this problem. The maximum value of the ratio, approximately 0.18, is obtained when $n = m$. Its minimum value occurs when $n = 0$ (or equivalently when $m = 0$) and is given by $1/2m$ which tends to zero for large m. Thus choosing one admissible heuristic in preference to another can yield significant speed-ups without sacrificing optimality.

It has long been known [1] that inadmissible algorithms, which use probabilistic knowledge of the domain, will converge in linear expected time to close to the optimal solution even when admissible algorithms will take provably exponential time. The problems we are interested in solving are already formulated as Bayesian estimation problems which means that probabilistic knowledge of the domain is already available. How can it be exploited?

Consider, for example, the admissible A* algorithm which we defined for the road tracking problem, see section (5). Let us consider the special case when the branching factor is always three and all paths are equally likely. Then we demonstrated that the smallest admissible heuristic is $h(a) = (L - M_a) \max_y \log\{p_1(y)/p_0(y)\}$. This heuristic, however, is really a worst case bound because the chances of getting such a result if we measure the response on the true path are very small. In fact, if we assume that the image measurements are independent (as our Bayesian model does, see section (2)), then the law of large numbers says the total response of all test results along the true path should be close to $h_p(a) = (L - M_a \sum_y p_1(y) \log\{p_1(y)/p_0(y)\}$. This *average bound* will typically be considerably smaller than the *worst case* bound used to determine $h(a)$ above. We would therefore expect that, in general, the average case heuristic $h_p(a)$ will be far quicker than the worst case heuristic $h(a)$ and should usually lead to equally good results. (Of course, the average case heuristic will become poor as we approach the end point, where $L - M_a$ is small, and so we will have to replace it by the worst case heuristic in such situations.)

Our recent work [4] [5] has made these intuitions precise by exploiting results from the theory of types, see [22], which quantify how rapidly the law of large numbers starts being effective. For example, Sanov's theorem can be used to determine the probability that the average cost for a set of n samples from the true road differs from the expected average cost $\sum_y p_1(y) \log\{p_1(y)/p_0(y)\}$. The theorem shows that the probability of any difference will decrease *exponentially* with the number of samples n. Conversely, we can ask with what probability will we get an average cost close to $\sum_y p_1(y) \log\{p_1(y)/p_0(y)\}$ from a set of n samples *not from the true path* (i.e. the probability that the algorithm will be fooled into following a false path). Again, it follows from Sanov's theorem that the probability of this happening decreases exponentially with n where the coefficient in the exponent is the Kullback-Leibler distance between $p_1(y)$ and $p_0(y)$.

Our papers [4] [5] prove expected convergence time bounds for optimization problems of the type we have analyzed in this paper. For example, in [5] we prove that inadmissible heuristic algorithms can be expected to solve these optimization problems (with a quantified expected error) while examining a number of nodes which varies linearly with the size of the problem. (The expected sorting time per node is shown to be constant). Moreover, the difficulty of the problem is determined by an *order parameter* which is specified in terms of the characteristics of the domain (the distributions $P_{on}, P_{off}, P(X)$ and the branching factor of the search tree). At critical values of the order parameter the problem undergoes a phase transition and becomes impossible to solve *by any algorithm.*

7 Summary

In summary, we argue that A*, and heuristic algorithms [1], give a good framework to compare and evaluate different optimization algorithms for deformable templates. We describe how both Dijkstra and Dynamic Programming can be expressed in terms of A* [1], [26]. We then prove a close relationship between the twenty questions algorithm [2] and a novel algorithm which we call A+. In turn, we prove that A+ is an inadmissible variant of A*. We note that both A+ and twenty questions, unlike A* and Dijkstra, maintain explicit probabilities of partial solutions which allows them to keep track of how well the algorithm is doing and to warn of faulty convergence. In addition, their explicit use of prior knowledge allows them to improve their search strategy (in general) by making predictions. However, both A* and Dijkstra are designed to work on graphs, which include closed loops, and it may be difficult to extend twenty questions and A+ to such representations.

From the A* perspective, the role of heuristics is very important. Most algorithms, implicitly or explicitly, make use of heuristics the choice of which can have a big effect on the speed and effectiveness of the algorithm. It is therefore important to specify them explicitly and analyze their effectiveness. For example, it appears that probabilistic knowledge of the problem domain can lead to heuristics, adapted to the domain, which are provably very effective [1],[4],[5]. By contrast, algorithms such as Dijkstra, which have no explicit heuristics, have no mechanisms for adapting the algorithm to a specific domain.

For example, Dijkstra's algorithm applied to detecting visual shapes is very effective at low noise levels but can break down (in terms of memory and time) at high noise levels (Geiger – private communication). Characterizing the noise probabilistically and using this to guide a choice of heuristic can lead to a more efficient algorithms, see [25].

Acknowledgements

This report was inspired by an interesting talk by Donald Geman and by David Mumford wondering what relation there was between the twenty question algorithm and A*. One of us (ALY) is grateful to Davi Geiger for discussions of the Dijkstra algorithm and to M.I. Miller and N. Khaneja for useful discussions about the results in this paper and their work on Dynamic Programming. This work was partially supported by NSF Grant IRI 92-23676 and the Center for Imaging Science funded by ARO DAAH049510494. ALY was employed by Harvard while some of this work was being done and would like to thank Harvard University for many character building experiences. He would also like to thank Prof.'s Irwin King and Lei Xu's hospitality at the Engineering Department of the Chinese University of Hong Kong and, in particular, to Lei Xu for mentioning Pearl's book on heuristics and for the use of his copy.

References

[1] J. Pearl. **Heuristics**. Addison-Wesley. 1984.

[2] D. Geman. and B. Jedynak. "An active testing model for tracking roads in satellite images". *IEEE Trans. Patt. Anal. and Machine Intel.* Vol. 18. No. 1, pp 1-14. January. 1996.

[3] J. Pearl. **Probabilistic Reasoning in Intelligent Systems**. San Mateo, CA:Morgan Kauffman. 1988.

[4] A.L. Yuille and J.M. Coughlan. "Convergence Rates of Algorithms for Visual Search: Detecting Visual Contours". In *Proceedings NIPS'98*. 1998.'

[5] A.L. Yuille and J.M. Coughlan. “Visual Search: Fundamental Bounds, Order Parameters, Phase Transitions, and Convergence Rates”. Submitted to *Transactions on Pattern Analysis and Machine Intelligence.* 1998.

[6] M.A. Fischler and R.A. Erschlager. “The Representation and Matching of Pictorial Structures”. *IEEE. Trans. Computers.* C-22. 1973.

[7] U. Grenander, Y. Chow and D. M. Keenan, *Hands: a Pattern Theoretic Study of Biological Shapes,* Springer-Verlag, 1991.

[8] U. Grenander and M.I. Miller. “Representation of Knowledge in Complex Systems”. *Journal of the Royal Statistical Society*, Vol. 56, No. 4, pp 569-603. 1994.

[9] T. F. Cootes and C. J. Taylor, “Active Shape Models - 'Smart Snakes',” *British Machine Vision Conference,* pp. 266-275, Leeds, UK, September 1992.

[10] B. D. Ripley. “Classification and Clustering in Spatial and Image Data”. In **Analyzing and Modeling Data and Knowledge**. Eds. M. Schader. Springer-Verlag. Berlin. 1992.

[11] L.H. Straib and J.S. Duncan. “Parametrically deformable contour models”. *Proceedings of Computer Vision and Pattern Recognition*, pp 98-103. San Diego, CA. 1989.

[12] L. Wiscott and C. von der Marlsburg. “A Neural System for the Recognition of Partially Occluded Objects in Cluttered Scenes”. Neural Computation. 7(4):935-948. 1993.

[13] A.L. Yuille “Deformable Templates for Face Recognition”. *Journal of Cognitive Neuroscience.* Vol 3, Number 1. 1991.

[14] R. E. Bellman, *Applied Dynamic Programming,* Princeton University Press, 1962.

[15] U. Montanari. “On the optimal detection of curves in noisy pictures.” *Communications of the ACM*, pages 335–345, 1971.

[16] M. Barzohar and D. B. Cooper, "Automatic Finding of Main Roads in Aerial Images by Using Geometric-Stochastic Models and Estimation," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 459-464, 1993.

[17] D. Geiger, A. Gupta, L.A. Costa, and J. Vlontzos. "Dynamic programming for detecting, tracking and matching elastic contours." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-17, March 1995.

[18] J. Coughlan. "Global Optimization of a Deformable Hand Template Using Dynamic Programming." Harvard Robotics Laboratory. Technical report. 95-1. 1995.

[19] N. Khaneja, M.I. Miller, and U. Grenander. "Dynamic Programming Generation of Geodesics and Sulci on Brain Surfaces". Submitted to *PAMI.* 1997.

[20] D. Bertsekas. *Dynamic Programming and Optimal Control.* Vol. 1. (2nd Ed.) Athena Scientific Press. 1995.

[21] D. Geiger and T-L Liu. "Top-Down Recognition and Bottom-Up Integration for Recognizing Articulated Objects". Preprint. Courant Institute. New York University. 1996.

[22] T.M. Cover and J.A. Thomas. **Elements of Information Theory**. Wiley Interscience Press. New York. 1991.

[23] L. Kontsevich. Private Communication. 1996.

[24] W. Richards, and A. Bobick. "Playing twenty questions with nature." In: **Computational Processes in Human Vision: An Inter- disciplinary Perspective.** Z. Pylyshyn, Ed; Ablex, Norwood, NJ. 1988.

[25] J.M. Coughlan, A.L. Yuille, C. English, and D. Snow. "Efficient Deformable Template Detection and Localization without User Initialization". Submitted to CVIU. 1998.

[26] P.H. Winston. **Artificial Intelligence**. Addison-Wesley Publishing Company. Reading, Massachusetts. 1984.

[27] S. Russell and P. Norvig. “Artificial Intelligence: A Modern Approach. Prentice-Hall. 1995.

[28] S.L. Lauritzen and D.J. Spiegelhalter. “Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems”. *Journal of the Royal Statistical Society*. B. Vol. 50. No. 2., pp 157-224. 1988.