A* Beats Dynamic Programming

Pedro Felzenszwalb
University of Chicago
pff@cs.uchicago.edu

David McAllester
TTI at Chicago
mcallester@tti-c.org

Abstract

We generalize Dijkstra's shortest path algorithm, A* search, and abstraction heuristics to the problem of computing lightest weight derivations for general weighted logic programs. This generalization gives an essentially mechanical way of improving the ef£ciency of a wide variety of dynamic programing algorithms. We discuss examples in natural language processing and computer vision. As a demonstration we develop a novel algorithm for £nding contours in images. The logic programming framework also allows a clean approach to the pipeline problem — the problem of passing information back and forth between various levels of processing in perceptual inference.

Introduction

Dynamic programming (Bellman 1957) is one of the most fundamental techniques for designing ef£cient algorithms. It is known that some dynamic programming problems can be solved using Dijkstra's shortest path algorithm (Dijkstra 1959) while avoiding the computation of unnecessary values. This idea can be pushed further by using A* search (Hart, Nilsson, & Raphael 1968) which is one of the most successful paradigms in arti£cial intelligence. More recently admissible heuristics derived from abstractions have been used to greatly improve the power of A* search (Holte *et al.* 1994; Culberson & Schaeffer 1998). Here we bring these ideas together within the formal setting of bottom-up logic programming. We show, in essence, that for every dynamic programming algorithm there exists a Dijkstra's shortest path version, an A* version, and a mechanical way of constructing admissible heuristics from abstractions.

This work was originally motivated by a dynamic programming algorithm for £nding optimal embeddings of ¤exible models in images (Felzenszwalb 2005). This algorithm, as with most dynamic programming algorithms, can be formulated as inference rules for using table entries to £ll in more table entries. In general one gives a name P to each table of a dynamic programming algorithm and models a table entry by an assertion $P(x_1, , \ldots, x_n)$. Given a representation of tables as assertions, a dynamic programming algorithm can be formulated as inference rules for £lling in tables. In the case of £nding optimal embeddings of models in images we realized that the inferences performed by these rules could be done in increasing order of cost yielding a "Dijkstra lightest derivation" algorithm for this problem. Furthermore, it is possible to formulate a notion of admissible heuristic and give an A* version of the algorithm.

Statistical parsing systems have long used Dijkstra lightest derivation methods involving a priority queue so that derivations are enumerated in order of increasing cost (Caraballo & Charniak 1996). More recently, heuristic £gures of merit have been replaced in some parsers by admissible A* heuristics (Klein & Manning 2003). Some of our results can be viewed as a generalization of A* statistical chart parsing to arbitrary dynamic programming problems.

Abstractions provide a way of automatically constructing admissible heuristics for A* search. A simple example is solving the Rubik's cube. Suppose we ignore the edge and center pieces and consider solving only the corners. The shortest solution for the corners is a lower bound on the shortest solution for all pieces. In general, any function on a state space de£nes a new state space where each new state represents a set of the original states. Each concrete move, such as the rotation of a face in Rubik's cube, yields an abstract move between abstract states. The set of abstract states together with abstract moves de£nes an abstract search space and least cost solutions in the abstract space de£ne lower bounds — admissible heuristics — on the cost of solutions in the concrete space. Admissible heuristics derived from abstraction can be viewed as a re£nement of abstraction spaces in planning (Sacerdoti 1977).

Here we show that the process of deriving an admissible heuristic from an abstraction can be generalized to any abstraction function on the assertions used in a weighted logic program. Any given abstraction function on assertions, and any given set of inference rules on the concrete (not abstracted) assertions, de£nes a set of abstract inference rules derived by "projecting" the concrete rules onto the abstract assertions. The cost of a lightest derivation using abstract assertions and abstract rules gives a lower bound on the weight of a concrete derivation. The use of abstract assertions and abstract derivations seems related to the notion of proof planning in automated reasoning (Bundy 1988).

Our work is related to the work on logical algorithms in (Ganzinger & McAllester 2002). Many algorithms, especially dynamic programming algorithms, are naturally represented as inference rules. However, in order to use inference rules formulations as a foundation for algorithm design one must give a precise model of the run time of a set of inference rules (a logic program). This is non-trivial as many logic program compilers use sophisticated indexing and uni-

£cation algorithms and the order of run time of a logic program can depend, for example, on the indexing constructed by the compiler. The logical algorithms work shows that it is possible to give relatively simple models of the run time of logic programs that are consistent with compilation to standard random access computers. We plan to explore this idea in the future.

Admissible heuristics derived from abstractions are often stored in tables that are fully computed in advance. This approach is limited to tables that remain £xed over different problem instances, or small tables if the heuristic must be recomputed for each instance. However it is possible to construct heuristic values on the ¤y using a search over abstract solutions that is done simultaneously with the search for concrete solutions. A hierarchy of heuristics can be constructed each guiding the search for solutions at the level below (Holte 1996). Here we show that when using a hierarchy of abstractions all of these searches can be done in parallel using a single priority queue interleaving the computation. The priorities on this queue represent lower bounds on the cost of concrete solutions of the overall problem.

A fundamental problem in statistical AI systems is the "pipeline problem". In the general case we have a concatenation of systems where each stage is feeding information to the next. In vision, for example, we might have an edge detector feeding edges to a system for £nding contours which in turn feed contours into a system for image segmentation which feeds segments into an object recognition system. Because of the need to build and compose modules with clean interfaces, pipelines are often forced to make hard decisions at module boundaries. For example, an edge detector typically constructs a Boolean array which indicates at each image point whether or not an edge was detected at that point. But there is general recognition that the presence of an edge at a certain point can depend on the context around that point. People can see an edge at places where the image gradient is zero if, at higher cognitive level, it is clear that there is actually an object boundary at that point. Speech recognition systems often return n-best lists which may or may not contain the actual utterance. We would like the speech recognition system to be able to take higher level level information into account and avoid the hard decision of exactly what strings to output in its n-best list. We show here that if all stages of the pipeline are written as lightest inference logic programs with pattern-based admissible heuristics then the entire pipeline, as well as the computation of heuristic values, can be run simultaneously in a way that allows each stage to be controlled by heuristic values derived form the entire downstream pipe. In this case partial results obtained by high-level processing can guide low-level processing.

Our main example in this paper is a new algorithm for £nding salient contours in images. A salient contour is a curve that optimizes a score based on its length (longer contours are better), its shape (smooth contours are better) and the image gradient along its path. Most methods use a saliency score that is expressed in terms of subscores based on local contour properties (Montanari 1971). In this case the problem of £nding the most salient contour of length up to L can be solved ef£ciently using dynamic programming — $O(nL)$ time for an image with n pixels. However, these models have dif£culty capturing non-local shape constraints. For example, they can not be used to look for lines that are almost straight. Here we consider a different model for contours in which long distance shape constraints are expressible. A dynamic programming algorithm for this model is too slow for practical use. However, we show that there is an A* version of the algorithm that is fast in practice. We feel that the A* version of this contour £nding algorithm, and the pattern-derived heuristic function in particular, would be very dif£cult to formulate without the general framework introduced here.

Formal Preliminaries

The lightest derivation problem can be formalized as follows. Let Σ be a set of statements and R be a set of inference rules of the following form,

$$\begin{array}{l} A_1 = w_1 \\ \quad \vdots \\ A_n = w_n \\ \hline C = g(w_1, \ldots, w_n) \end{array}$$

Here the antecedents A_i and the conclusion C are statements in Σ, the w_i are real valued variables and g is a real valued function. Intuitively, the rule says that if there are derivations of the antecedents A_i with weights w_i then we can derive the conclusion C with weight $g(w_1, \ldots, w_n)$. The goal is to compute the lightest derivations of a set of statements. Note that a derivation of C can be represented by a tree rooted at a rule $A_1, \ldots, A_n/C$ with n children, where the i-th child is a derivation of A_i. The leaves of this tree are rules with no antecedents.

We will always assume that for each rule in R the weight of the conclusion is at least as large as the weights of the antecedents $g(w_1, \ldots, w_n) \geq w_i$. This implies that in a derivation tree the weights of the conclusions increase in a path from a leaf to the root. We will also assume that g is monotone — increasing the weight of an antecedent can not decrease the weight of the conclusion. This implies that subtrees of lightest derivation trees are lightest derivation trees themselves.

Most dynamic programming algorithms can be seen as solving a lightest derivation problem. In dynamic programming there is an ordering of the statements $(B_1, \ldots, B_k)$ such that for any rule with conclusion B_i the antecedents are statements that come before B_i in the ordering. Using this ordering lightest derivations can be computed sequentially by iterating from $i = 1$ to k. At each step the lightest derivation of B_i is obtained by minimizing over all rules that can be used as a £nal derivation for B_i.

Rules for CKY chart parsing are shown in Figure 1. We assume that we are given a weighted context free grammar in Chomsky normal form, i.e., a weighted set of productions of the form $X \rightarrow y$ and $X \rightarrow YZ$ where X, Y and Z are nonterminal symbols and y is a terminal symbol. The input string is given by a sequence of terminals $(s_1, \ldots, s_n)$. The £rst set of rules state that if the grammar contains a produc-

(1) $$\frac{}{\textit{phrase}(X, i, i+1) = w(X \to s_i)}$$

(2) $$\frac{\begin{array}{l}\textit{phrase}(Y, i, j) = w_1 \\ \textit{phrase}(Z, j, k) = w_2\end{array}}{\textit{phrase}(X, i, k) = w_1 + w_2 + w(X \to YZ)}$$

Figure 1: The derivation rules for CKY chart parsing.

tion $X \to s_i$ then there is a phrase of type X generating the i-th entry of the input with weight $w(X \to s_i)$. The second set of rules state that if the grammar contains a production $X \to YZ$ and there is phrase of type Y from i to j and a phrase of type Z from j to k then there is an, appropriately weighted, phrase of type X from i to k. Let S be the start symbol of the grammar. The goal of parsing is to £nd the lightest derivation of *phrase*$(S, 1, n+1)$. A dynamic programming solution can be obtained by picking an ordering of the statements such that *phrase*(X, i, j) comes before *phrase*(Y, k, l) when $|i - j| > |k - l|$. The order among different phrases of the same length is irrelevant.

Dijkstra Lightest Derivation

We de£ne a bottom-up logic programming language in which we can easily express the algorithms we wish to discuss. A program is de£ned by a set of rules with priorities and a goal statement. We encode the priority of a rule by writing it along the line separating the antecedents and the conclusion,

$$\frac{\begin{array}{l}A_1 = w_1 \\ \quad\vdots \\ A_n = w_n\end{array}}{C = g(w_1, \ldots, w_n)}\, p(w_1, \ldots, w_n)$$

The execution of a set of prioritized rules P with goal statement *goal* is de£ned by the algorithm in Figure 2. We keep track of a set $\mathcal{S}$ and a priority queue $\mathcal{Q}$ of assignments of the form $(B = w)$. Initially $\mathcal{S}$ is empty and $\mathcal{Q}$ contains the conclusions of all rules with no antecedents at the priorities given by those rules. We iteratively remove the lowest priority assignment $(B = w)$ from $\mathcal{Q}$. If B already has an assigned weight in $\mathcal{S}$ then the assignment is ignored. Otherwise we add $(B = w)$ to $\mathcal{S}$ and "expand it" — every assignment derivable from $(B = w)$ and other assignments in $\mathcal{S}$ using a single rule in P is added to $\mathcal{Q}$ at the priority speci£ed by the rule. The program stops as soon as a weight is assigned to *goal*.

We can compute the lightest derivation of a statement *goal* under a set of rules R using an algorithm similar to Dijkstra's shortest path. The algorithm is de£ned in terms of rules with priorities as follows,

De£nition 1 (Dijkstra's lightest derivation) *Given R, de£ne a set of rules with priorities $\mathcal{D}(R)$ by setting the priority of each rule to be the weight of its conclusion.*

Algorithm *Run*$(P, \textit{goal})$
1. $\mathcal{S} \leftarrow \emptyset$
2. Initialize $\mathcal{Q}$ with conclusions of no-antecedent rules at the priorities speci£ed by those rules.
3. **repeat**
4. Remove lowest priority element $(B = w)$ from $\mathcal{Q}$.
5. **if** $(B = w') \notin \mathcal{S}$
6. $\mathcal{S} \leftarrow \mathcal{S} \cup \{(B = w)\}$
7. **if** $B = \textit{goal}$ **stop**
8. Insert conclusions derivable from $(B = w)$ and other assignments in $\mathcal{S}$ using a single rule into $\mathcal{Q}$ at the priorities speci£ed by the rule.

Figure 2: Running a set of rules with priorities.

Let $\ell(B)$ denote the weight of the lightest derivation of B using the rules in R when B is derivable and in£nity otherwise. It can be shown that if *goal* is derivable then $\mathcal{D}(R)$ will £nd its lightest derivation. Moreover if $(B = w) \in \mathcal{S}$ then $w = \ell(B)$ and $\ell(B) \leq \ell(\textit{goal})$. This means that all assignments in $\mathcal{S}$ represent lightest derivations and only statements B such that $\ell(B) \leq \ell(\textit{goal})$ will be expanded. The main advantage of $\mathcal{D}(R)$ over a dynamic programming solution to the lightest derivation problem is that $\mathcal{D}(R)$ only expands a subset of all statements.

A* Lightest Derivation

Let h be a heuristic function assigning a cost to each statement in Σ. We say that h is *monotone* if for every rule $A_1, \ldots, A_n/C$ in R,

$$\ell(A_i) + h(A_i) \leq g(\ell(A_1), \ldots, \ell(A_n)) + h(C). \qquad (1)$$

We can compute the lightest derivation of *goal* under the rules in R using a type of A* search that takes into account the value of a monotone heuristic function. As before the algorithm is de£ned in terms of rules with priorities,

De£nition 2 (A* lightest derivation) *Given R and h, de£ne a set of rules with priorities $\mathcal{A}(R)$ by setting the priority of each rule to be the weight of its conclusion plus the heuristic value, $g(w_1, \ldots, w_n) + h(C)$.*

It can be shown that if *goal* is derivable then $\mathcal{A}(R)$ will £nd its lightest derivation. Moreover if $(B = w) \in \mathcal{S}$ then $w = \ell(B)$ and $\ell(B) + h(B) \leq \ell(\textit{goal}) + h(\textit{goal})$. As in standard A* search $\mathcal{A}(R)$ avoids expanding statements that are not promising according to the heuristic function.

In standard A* search good heuristic functions can be automatically constructed using abstractions. This idea can also be applied to our situation. In what follows we consider only problems de£ned by sets of rules where the weight of a conclusion is always the sum of the weights of the antecedents plus a non-negative value v. We denote such a rule by $A_1, \ldots, A_n \to_v C$. We can think of v as the weight of the rule. Note that the weight of a derivation using these types of rules is just the sum of the weights of the rules that appear in the derivation tree.

Let *context*(B) be a derivation of *goal* with a hole that can be £lled by a derivation of B. Each context has a weight

which is the sum of weights of the rules in it. It can also be seen as the weight it assigns to *goal* assuming that B can be derived with zero weight. We de£ne $\ell(\mathit{context}(B))$ to be the weight of a lightest context for B. This is analogous to the distance from a node to the goal in standard search problems. Note that it is possible that $\ell(B)$ is in£nite while $\ell(\mathit{context}(B))$ is £nite. Lightest contexts can be computed by solving the following problem. For each rule $A_1, \ldots, A_n \rightarrow_v C$ in R we de£ne,

$$\frac{\mathit{context}(C) = w}{\mathit{context}(A_i) = v + \sum_{j \neq i} \ell(A_j) + w}$$

The base case is the context of *goal* which by de£nition equals zero. This can be captured by a rule with no antecedents, $\rightarrow_0 \mathit{context}(\mathit{goal})$.

Let *abs* be a an abstraction mapping statements in Σ to a set of abstract statements. The abstraction can be used to obtain an abstract lightest derivation problem with rules $\mathit{abs}(R)$ as follows. For each rule in $A_1, \ldots, A_n \rightarrow_v C$ in R we de£ne an abstract rule,

$$\mathit{abs}(A_1), \ldots, \mathit{abs}(A_n) \rightarrow_v \mathit{abs}(C).$$

Note that $\ell(\mathit{abs}(C)) \leq \ell(C)$. If we let the goal of the abstract problem be $\mathit{abs}(\mathit{goal})$ we can use abstract contexts to de£ne a heuristic function for A*. To see that $\ell(\mathit{context}(\mathit{abs}(C)))$ is a monotone heuristic function consider a rule $A_1, \ldots, A_n \rightarrow_v C$ in R. For any such rule there is a corresponding abstract rule and by the de£nition of context we have,

$$\ell(\mathit{context}(\mathit{abs}(A_i))) \leq v + \sum_{j \neq i} \ell(\mathit{abs}(A_j)) + \ell(\mathit{context}(\mathit{abs}(C))).$$

We know that $\ell(\mathit{abs}(A_i)) \leq \ell(A_i)$ which is enough to see that $\ell(\mathit{context}(\mathit{abs}(C)))$ is monotone.

In practice we can compute lightest abstract derivations and contexts using dynamic programming or Dijkstra's lightest derivation. If the set of abstract statements is small this is signi£cantly more ef£cient than using dynamic programming or Dijkstra's lightest derivation on the concrete problem. Another option is to build a single program that computes abstract weights and abstract contexts at the same time that we are computing lightest derivations of the concrete problem. This idea is explored in the next section.

Hierarchical A* Lightest Derivation

We de£ne an abstraction hierarchy to be a sequence of disjoint sets of statements $\Sigma_0, \ldots, \Sigma_m$ plus a single abstraction function *abs*. For $i < m$ the abstraction maps Σ_i onto Σ_{i+1} and Σ_m contains a single statement $\perp$ with $\mathit{abs}(\perp) = \perp$. Since *abs* is onto we have $|\Sigma_{i+1}| \leq |\Sigma_i|$ with $|\Sigma_m| = 1$. We denote by abs_i the abstraction from Σ_0 to Σ_i obtained by composing *abs* with itself i times.

As an example consider the eight puzzle where abs_i replaces tiles numbered i or less with a black tile. For $A \in \Sigma_i$ we have that $\mathit{abs}(A)$ replaces the tile numbered $i+1$ with a black tile. In this example $|\Sigma_i|$ declines exponentially as

START1: $$\frac{}{\mathit{context}(\perp) = 0}\, v_{\min}$$

START2: $$\frac{\mathit{goal}_i = w}{\mathit{context}(\mathit{goal}_i) = 0}\, w$$

UP: $$\frac{\mathit{context}(\mathit{abs}(C)) = w_c \quad A_1 = w_1 \quad \vdots \quad A_n = w_n}{C = v + w_1 + \cdots + w_n}\, v + w_1 + \cdots + w_n + w_c$$

DOWN: $$\frac{\mathit{context}(C) = w_c \quad A_1 = w_1 \quad \vdots \quad A_n = w_n}{\mathit{context}(A_i) = v + w_c + w_1 + \cdots + w_n - w_i}\, v + w_c + w_1 + \cdots + w_n$$

Figure 3: The hierarchical algorithm $\mathcal{H}(R)$. Here $v_{\min}$ is the smallest weight associated with a no-antecedent rule in R. START2 rules are de£ned for $1 \leq i < m$. UP and DOWN rules are de£ned for each rule $A_1, \ldots, A_n \rightarrow_v C \in \mathit{abs}_i(R)$ with $0 \leq i < m$ for UP and $1 \leq i < m$ for DOWN.

i increases. This makes it much more ef£cient to compute derivations and contexts in Σ_i as i increases. On the other hand contexts in Σ_i provide better heuristic for solving the eight puzzle as i decreases.

In general we consider a lightest derivation problem with statements in Σ_0, rule set R and goal statement *goal*. The abstraction can be used to de£ne a set of problems with statements in Σ_i, rules $\mathit{abs}_i(R)$ and goal statement $\mathit{goal}_i = \mathit{abs}_i(\mathit{goal})$. For any statement B in the abstraction hierarchy we de£ne $\ell(B)$ and $\ell(\mathit{context}(B))$ in terms of these problems. Note that we always have $\ell(\mathit{abs}(B)) \leq \ell(B)$ and $\ell(\mathit{context}(\mathit{abs}(B))) \leq \ell(\mathit{context}(B))$.

The hierarchical A* lightest derivation algorithm $\mathcal{H}(R)$ is de£ned by the set of rules with priorities in Figure 3. The rules labeled START1 and START2 compute the context of the goal in each abstract problem. Rules labeled UP compute derivations of statements in Σ_i using the contexts of their abstractions in Σ_{i+1} as a heuristic. Rules DOWN derive contexts in each abstract level using derivations at that level. Intuitively the algorithm starts by deriving the context of $\perp$, it can then derive statements in Σ_{m-1}. Once the lightest derivation of goal_{m-1} is found the algorithm starts computing contexts of statements in Σ_{m-1} and so on.

The correctness of the hierarchical algorithm $\mathcal{H}(R)$ can be proved from a more general theorem. Consider an arbitrary set of prioritized rules M with statements in Γ. De£ne $\ell(\Phi, M)$ to be the weight of a lightest derivation of Φ using the rules in M ignoring priorities. It is not hard to show that $\ell(\mathit{goal}, \mathcal{H}(R))$ equals $\ell(\mathit{goal})$. We now state general condi-

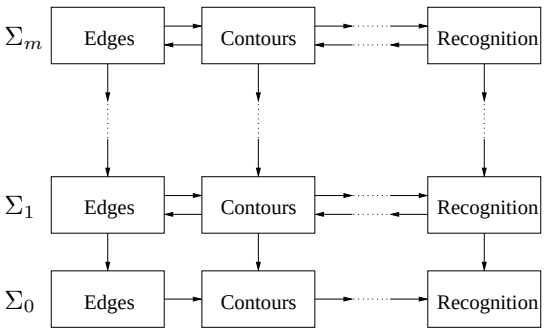

Figure 4: A vision system with several levels of processing. Forward arrows represent the normal ¤ow of information from one stage of processing to the next. Downward arrows represent the in¤uence of contexts. Backward arrows represent the computation of contexts.

tions under which executing a set of prioritized rules M with the algorithm in Figure 2 correctly computes $\ell(\textit{goal}, M)$.

A function $q : \Gamma \times \mathbb{R} \rightarrow \mathbb{R}$ will be called a *proper priority function* for M if it satis£es the following conditions:

- q represents priority — in every instance of a rule in M with antecedents of the form $\Psi_i = \ell(\Psi_i, M)$ and conclusion $\Phi = w$ we have that $q(\Phi, w)$ equals the priority of the rule instance.
- Inference never reduces priority — in every instance of a rule in M with antecedents of the form $\Psi_i = \ell(\Psi_i, M)$ and conclusion $\Phi = w$ we have that $q(\Phi, w) \geq q(\Psi_i, \ell(\Psi_i, M))$ for all antecedents Ψ_i.
- Priority is monotonic in weight — if $w < w'$ then $q(\Phi, w) < q(\Phi, w')$.

Theorem 1 *Let M be a set of rules with priorities. If there exists a proper priority function for M and* goal *is derivable then the algorithm in Figure 2 will £nd a lightest derivation of* goal. *Moreover if $(\Phi = w) \in \mathcal{S}$ then $w = \ell(\Phi, M)$ and $q(\Phi, \ell(\Phi, M)) \leq q(\text{goal}, \ell(\text{goal}, M))$.*

For the rule set $\mathcal{H}(R)$ we can de£ne the priority function as follows,

$$q(A, w) = w + \ell(\textit{context}(\textit{abs}(A))) \quad (2)$$

$$q(\textit{context}(A), w) = w + \ell(A) \quad (3)$$

The correctness of $\mathcal{H}(R)$ and a bound on the number of statements $\mathcal{H}(R)$ expands follows from the following theorem which we state without proof.

Theorem 2 *The function q de£ned by (2) and (3) is a proper priority function for $\mathcal{H}(R)$.*

Figure 4 illustrates a hypothetical case where R represents the processing "pipeline" of a vision system. Statements from one stage of processing provide input to the next stage and the system ultimately derives statements about the objects in a scene. The hierarchical algorithm $\mathcal{H}(R)$ provides a mechanism by which a processing stage can be in¤uenced by later stages.

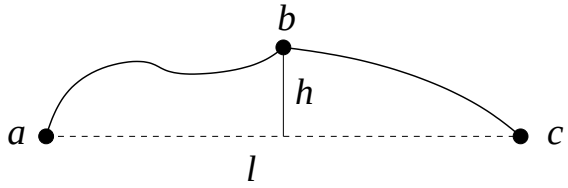

Figure 5: A curve with endpoints (a, c) is formed by composing curves with endpoints (a, b) and (b, c), where b is equally distant from a and c. The cost of the composition is proportional to $(h/l)^2$. This cost is scale invariant and encourages contours to be straight by pulling b toward the line containing a and c.

Finding Salient Contours in Images

Now we turn to the problem of £nding salient contours in images. Essentially we want to look for contours that are long, do not bend too much and tend to go along paths with high image gradient. Here we consider a new type of model that can be used to capture more global shape constraints than is possible with previous methods. The problem of £nding the most salient contour in an image using this model can be solved using dynamic programming, but this approach is too slow for practical use. On the other hand, our experiments show that using a heuristic derived from a simple abstraction we obtain a fast A* algorithm.

Let C_1 be a curve with endpoints a and b and C_2 be a curve with endpoints b and c. The two curves can be composed to form a curve C with endpoints a and c. We de£ne the cost of the composition to be the sum of the costs of C_1 and C_2 plus a "shape" cost which depends on the geometric arrangement of the points (a, b, c). Figure 5 illustrates the composition of two curves and the shape cost that we use. Besides the composition rule we assume that if a and b are neighboring locations in the image then there is a curve with endpoints a and b whose cost depends only on the image data. This forms the base case for creating longer curves. Rules for £nding the best curve between each pair of endpoints are shown below.

$$\frac{\textit{curve}(a, b) = w_1 \quad \textit{curve}(b, c) = w_2}{\textit{curve}(a, c) = w_1 + w_2 + \textit{shape}(a, b, c)}$$

$$\frac{||a - b|| < 2}{\textit{curve}(a, b) = D(a) + D(b)}$$

In practice we use a data cost $D(a)$ that is zero if a is on a ridge of the image gradient and a positive constant otherwise. Ridges of the image gradient are exactly the locations labeled as edges by the Canny edge detector (Canny 1986).

The costs just de£ned are not a good measure of saliency by themselves because they always prefer short curves over long ones. A saliency measure should prefer longer contours if they have good shape and are supported by the image data. We de£ne the saliency of a curve to be its cost minus the distance between its endpoints. In this case the problem of

£nding the most salient contour in the image can be solved by looking for the lightest derivation of *goal* using,

$$\frac{\textit{curve}(a,b) = w}{\textit{goal} = w - ||a-b|| + K}$$

Here K is any constant large as the maximum distance between endpoints to ensure that the weight of *goal* is at least the weight of *curve*(a,b). Computing the most salient contour using the rules de£ned above is a simple example of the pipeline problem. The naive way to £nd the most salient contour is to £rst compute the lightest curve between every pair of endpoints and then check which of those is most salient. In contrast, we can use the rules for computing lightest curves together with the rule for £nding the most salient contour to de£ne a single lightest derivation problem.

Using Dijkstra's lightest derivation is not enough for computing the most salient contour quickly. In particular K is relatively large and it causes too many short curves to be explored before the goal is derived. This can be avoided by using a heuristic function de£ned in terms of abstract contexts. If we partition the image into a grid of boxes there is a natural abstraction taking an image location to the box containing it. We can extend this abstraction to statements parameterized by image locations. For example, each abstract statement about curves is of the form *curve*(A,B) where A and B are boxes in the image. The lightest derivation of *curve*(A,B) gives a lower bound on the cost of any concrete curve from A to B.

A coarse heuristic function is able to capture that most short curves can not be extended to a long curve so A* concentrates its efforts in building curves in a sparse set of locations. Figure 6 illustrates some of results obtained by the A* algorithm and also which areas of the input image were heavily explored. In each case the most salient contour was found in about one second. In part (d) of Figure 6 we can see how the A* spent most of its time exploring the back of the penguin, exactly the location of the most salient contour. There are other places with good curves but the heuristic function can "see" that those curves will not be very long.

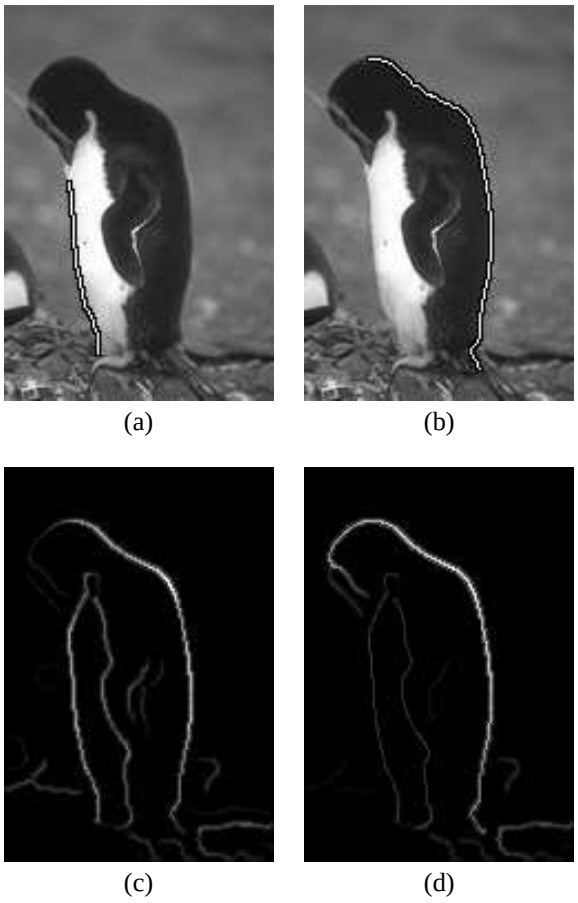

Figure 6: Images (a) and (b) show the most salient contours in a picture using different model parameters. The contour in (a) was obtained by penalizing curvature more aggressively than in (b). Images (c) and (d) represent how many times each possible endpoint was in a conclusion asserted by A* when £nding the optimal contours in (a) and (b) respectively.

References

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bundy, A. 1988. The use of explicit plans to guide inductive proofs. In *9th Conf. on Automated Deduction*.

Canny, J. 1986. A computational approach to edge detection. *IEEE Transactions on PAMI* 8(6).

Caraballo, S. A., and Charniak, E. 1996. Figures of merit for best-£rst probabilistic chart parsing. In *Empirical Methods in Natural Language Processing*.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3).

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerical Mathematics* 1.

Felzenszwalb, P. 2005. Representation and detection of deformable shapes. *IEEE Transactions on PAMI* 27(2).

Ganzinger, H., and McAllester, D. 2002. Logical algorithms. In *International Conf. on Logic Programming*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimal cost paths. *IEEE Trans. Syst. Science and Cybernetics* 4(2).

Holte, R.; Drummond, C.; Perez, M.; Zimmer, R.; and MacDonald, A. 1994. Searching with abstractions: A unifying framework and new high-performance algorithm. In *Canadian Arti£cial Intelligence Conf.*

Holte, R. 1996. Hierarchical a*: Searching abstraction hierarchies ef£ciently. In *AAAI*.

Klein, D., and Manning, C. 2003. A* parsing: Fast exact viterbi parse selection. In *HLT-NAACL*.

Montanari, U. 1971. On the optimal detection of curves in noisy pictures. *Commun. of the ACM* 14(5).

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior*. New York, NY: American Elsevier.