Shortest Path Discovery Problems: A Framework, Algorithms and Experimental Results

Csaba Szepesvári
Computer and Automation Research Institute of the Hungarian Academy of Sciences
1111 Budapest XI. Kende u. 13-17.
e-mail: szcsaba@sztaki.hu

Abstract

In this paper we introduce and study Shortest Path Discovery (SPD) problems, a generalization of shortest path problems: In SPD one is given a directed edge-weighted graph and the task is to find a the shortest path for fixed source and target nodes such that initially the edge-weights are unknown, but they can be queried. Querying the cost of an edge is expensive and hence the goal is to minimize the total number of edge cost queries executed. In this article we characterize some common properties of sound SPD algorithms, propose a particular algorithm that is shown to be sound and effective. Experimental results on real-world OCR task demonstrate the usefulness of the approach whereas the proposed algorithm is shown to yield a substantial speed-up of the recognition process.

Introduction

Shortest path problem appear in many subproblems of artificial intelligence, e.g. in symbolic planning, parsing, vision, or speech recognition. Depending on the nature of additional assumptions made on the underlying search graph one gets fundamentally different problem types: the graph may be finite or infinite, or one might have additional initial knowledge about the solutions in the form of estimates of the cost-to-go function like in heuristic search, or the search might be interleaved with execution like in real-time search, etc. Recently there has been a growing interest in incremental search (or dynamic search) where one considers a sequence of related search problems and the goal is to speed up the overall process by reusing solutions to previous instances (Ramalingam & Reps 1996).

Less attention have been paid to another important aspect of search, namely that obtaining the costs of the edges might be a non-trivial process and the cost of this process may dominate the total cost of the algorithm. As an example consider segmentation lattice based character recognition systems when a path from the source node to the target node in the segmentation lattice represents a possible segmentation, the edges of the lattice are associated with certain segments (parts) of the image to be recognized and where the edge costs are e.g. proportional to the negated log-likelihood of a given segment representing some (any) character. Provided that one has upper and lower bounds on the widths of characters the segmentation lattice will have a particularly simple structure with the number of its edges growing at most linearly with the width of the image area to be recognized. The number of edges in such a graph is typically in the range of 1.5-5.0 times the number of characters on the image. When the recognizer must work even with degraded quality images or images that contain heavy clutter then the OCR component often needs to check many alternative interpretations of a single input image corresponding to alternative "binarizations". In such a case the time needed to recognize a single character for a difficult input can be substantially more expensive than to solve the full shortest path problem. In such cases the cost of finding the shortest path is clearly dominated by the cost of obtaining the costs associated with the edges.

In segmentation based speech recognition the situation is entirely analogous. Another example comes from hierarchical (symbolic or non-symbolic) planning when calculating the cost associated with an "edge" requires the solution of a subproblem. Thus, hierarchical planning algorithms might also benefit from minimizing the total number of edge-cost evaluations. A related example is the exploration of unsafe environments where one is interested in minimizing the number of dangerous exploration steps. One particular case is when robots are used to explore a minefield. In this case it is not the computational cost, but the probability of loosing the robot that is to be minimized. This example can again be casted as a SPD problem.

The article is organized as follows: In the next section we introduce the algorithmic framework and derive some common properties of SPD algorithms. Next we present an algorithm that we call "Greedy SPD" that in each step queries the cost of edges along the shortest path found using the current best estimate of the cost function. It is shown that this algorithm is sound and that it is not possible to uniformly improve the performance of this algorithm. In Section results of some experiments are presented on the task of recognizing vehicle license plates, where the proposed algorithm is shown to yield substantial savings in the total running time of recognition. Relation to previous work is discussed in Section , whilst conclusions are drawn in Section .

Shortest Path Discovery Problems

Framework

Consider the shortest path problem $P = (V, E, c, s, t)$, where $G = (V, E)$ is a finite directed graph, V, E are the set of vertices and edges, respectively, $c : E \to \mathbb{R}$ is a function assigning costs (weights) to the edges of G, and where $s, t \in V$ are the source and target nodes, respectively. We further assume that the graph G has at least one directed path from s to t.

In the classical shortest path problem we are interested in finding the path connecting s and t with the least cost, where the cost of a path $\pi = (\pi_1, \ldots, \pi_n)$ is measured as the sum of the costs of the edges of the path:

$$c(\pi) = \sum_i c(\pi_i), \quad \pi_i \in E.$$

We shall denote by $\Pi(V, E, s, t)$ the set of paths in G connecting s and t, whilst $\Pi^*(V, E, c, s, t)$ shall denote the set of solutions of the shortest-path problem (V, E, c, s, t).

In the case of SPDs one is given a shortest path problem instance P and it is assumed that the costs of edges are initially unknown, but there exists a method `query` that can be used to query the cost of edges of G. The goal is to obtain a solution of P with the least number of query calls. Algorithms that solve SPDs take a SPD problem instance (possibly extended with additional information) and return solution candidates.

Definition 1 *An algorithm $\mathcal{A}$ for solving SPDs is* correct *if for all shortest path problem instances P when $\mathcal{A}$ terminates then it returns a solution of P. If an algorithm is correct and terminates in finite time for all problem instances then we call it* sound.

We shall call an edge *known* when its cost have been queried previously. The set of known edges is uniquely determined by the corresponding characteristic function $k : E \to \{0, 1\}$ that assigns 1 (0) to known (resp. unknown) edges. In what follows we shall refer to functions mapping the set of edges into the set $\{0, 1\}$ as *knowledge-functions*.

In what follows we shall assume that one is given an initial estimate of the edge costs, c_0 and a pair (P, c_0) shall be called an SPD problem instance. The hope is that when c_0 is a good approximation of c then it may be used in place of c to guess optimal paths and ultimately to avoid querying the cost of some edges. When $c_0 \leq c$ we say that c_0 is an *admissible* estimate of c.[1]

From the point of view of input-output behavior an SPD algorithm may be represented as a function mapping SPD instances (P, c_0) into paths of the graph underlying P. With a slight abuse of notation we shall use the same symbol to denote the algorithm and the function that it implements. Given a shortest path problem P and an initial cost estimate c_0 we let $k_{\mathcal{A}}(P, c_0)$ denote the knowledge function that corresponding to the set of known edges at the time when algorithm $\mathcal{A}$ terminates.

[1]Here and in what follows $\leq$ will be used to denote the usual partial ordering of functions, too: $f \leq g$ iff $f(x) \leq g(x)$ holds for all x from the common domain of f and g.

The optimization criterion we are interested in can be defined as follows:

Definition 2 *Let $\mathcal{A}$ be a sound algorithm and let (P, c_0) be an SPD instance. Let $k = k_{\mathcal{A}}(P, c_0)$ and define $\|k\|_1 = \sum_{e \in E} k(e)$. We say that $\mathcal{A}$ is* optimally effective *on the SPD instance (P, c_0) if for any other sound algorithm $\mathcal{A}'$ it holds that $\|k\|_1 \leq \|k'\|_1$, where $k' = k_{\mathcal{A}'}(P, c_0)$, i.e., the number of edges queried by $\mathcal{A}'$ is not smaller than the number of edges queried by $\mathcal{A}$.*

We say that an SPD algorithm $\mathcal{A}$ is optimally effective if it is sound and if it is optimally effective on all problem instances (P, c_0), where c_0 is any admissible initial cost estimate for the problem P.

The Fundamental Property of Correct SPD Algorithms

Fix a problem instance P. Let $k : E \to \{0, 1\}$ be an arbitrary knowledge function and define $c_k : E \to \mathbb{R}$ by

$$c_k(e) = \begin{cases} c(e), & \text{if } k(e) = 1; \\ c_0(e), & \text{if } k(e) = 0. \end{cases} \tag{1}$$

Note that c_k gives the "best" available estimate of the cost of edges given the knowledge function k.

Before giving the main result of this section we shall need one more definition. Let P be a shortest path problem. We call an edge e of P a *bottleneck* if all paths connecting s and t go through it. A graph is bottleneck free when it has no bottleneck edges. *In what follows we always assume that the graphs that we work with are bottleneck free.* Note that the cost of bottleneck edges need never be queried. Further, the identification of bottleneck edges does not require the knowledge of edge costs. Hence, the above assumption yields in no loss generality.

It should be fairly clear that a correct algorithm must query the cost of edges of the path that it returns. Further, it is also clear that a correct algorithm must return a shortest path given its best current estimate of the costs. The following theorem formalizes these observations and further it shows that the reverse statement holds as well:

Theorem 1 (Fundamental property of SPD problems) *Let $\mathcal{A}$ be an SPD algorithm that is designed to work with admissible initial cost estimates. The following statements are equivalent:*

1. *$\mathcal{A}$ is correct.*
2. *For all shortest path problems $P = (V, E, c, s, t)$ and admissible initial cost estimates c_0 if k represents the state of edge-cost knowledge at the time when the algorithm stops and π is the path returned by $\mathcal{A}$ then $k(e) = 1$ for all edges e of π and $\pi \in \Pi^*(V, E, c_k, s, t)$.*

Proof. Assume that (1) holds, but (2) does not hold. If (2) does not hold then either (i) $k(e) = 0$ for some $e \in \pi$, or (ii) $\pi \notin \Pi^*(V, E, c_k, s, t)$. In case (i) consider a modified problem P' that is equal to P in all respects except that $c(e)$ is replaced by some sufficiently large cost g. Since algorithm $\mathcal{A}$ has no knowledge of $c(e)$ it will return the same solution π. It should be clear that if $g > \max_\pi c(\pi)$ then π cannot be optimal in P'. Hence algorithm $\mathcal{A}$ cannot be correct, which is a contradiction. Therefore we may assume that (i) does not hold, i.e., $k(e) = 1$ for all edges of π. Now assume that case (ii) holds true. Let $\pi' \in \Pi^*(V, E, c_k, s, t)$. Let us now define a problem instance P' equivalent to P in all respects except that $c(e)$ is redefined such that it is equal to $c_0(e)$ for all edges e of π' such that $k(e) = 0$. If no such edges exist then we let $P' = P$. Again, algorithm $\mathcal{A}$ will still return π on P'. Therefore, by (i) we may assume that $k|_\pi \equiv 1$ (i.e. $k(e) = 1$ for all $e \in \pi$). Then $c(\pi') = c_k(\pi') < c_k(\pi) = c(\pi)$, which is a contradiction. This finishes the proof of (1)⇒(2).

Let us now consider the case (2)⇒(1): assume that (2) holds and let π be the path returned by $\mathcal{A}$ on P. Take any other path π' of P. Then $c(\pi') \geq c_k(\pi') \geq c_k(\pi) = c(\pi)$ since $k|_\pi \equiv 1$. This shows that π is indeed an optimal path of P. □

The importance of this theorem is that it shows that an optimally effective algorithm can and should stop when the shortest path w.r.t. the best estimate of the cost has no unknown edges. This observation motivates the following definition:

Definition 3 *Fix an SPD instance (P, c_0). Then a knowledge function $k : E \to \{0, 1\}$ is called* terminating *if there exists a shortest path π for the problem (V, E, c_k, s, t) such that $k|_\pi \equiv 1$.*

It should be clear that if k is terminating then for any $k' : E \to \{0, 1\}$ satisfying $k' \geq k$, k' is also terminating.

A Greedy Algorithm for Solving SPD Problems

Any SPD algorithm uniquely determines a sequence of knowledge functions. Naturally, the sequence of knowledge functions $\{k_n\}$ that an algorithm produces satisfies $k_n \leq k_{n+1}$. Actually, any reasonable algorithm would produce functions such that $k_n(e) < k_{n+1}(e)$ holds for at least one edge e, i.e., in each stage the algorithm queries the cost of at least one edge. Also, if k_n is terminating it does not make any sense to query some more edges, hence reasonable SPD algorithms should terminate whenever they reach any terminating knowledge function. Therefore the behavior of reasonable SPD algorithms can be represented as a walk on the directed acyclic graph $G_K = (V_K, E_K, k_0, T)$, where V_K is the set of all knowledge functions ($V_K = 2^E$), $(k_1, k_2) \in E_K$ iff $k_1 < k_2$, k_0 is the source node that satisfies $k_0 \equiv 0$ and T is the set of terminating knowledge functions associated with (P, c_0). The walk starts at k_0 and finishes whenever a node of T is reached. It follows that any algorithm that constructs such a walk on G_K is sound (termination within $|E|$ time steps follows since $k_n < k_{n+1}$ and the graph G is assumed to be finite). The cost associated with an SPD algorithm that constructs the walk $k_0, k_1, \ldots, k_n$ is $\|k_n\|_1$. Hence, the performance of SPD algorithms can be lower bounded by $\min_{k \in T} \|k\|_1$.

Now, let us consider Algorithm 1 given below: In each iteration it computes a shortest path (ties should be broken in a deterministic manner) given its current knowledge and then queries the cost of all unknown edges along this path. Termination happens when the number of unknown edges belonging to the current shortest path equals to zero. It follows immediately from Theorem 1 that this algorithm is sound. The algorithm is greedy as it queries *all* the unknown edges. As an alternative one might decide to query edges along the best path one by one, hoping that querying the cost of some of these edges would already prove that the current solution is suboptimal. Then the ordering at which edges are queried becomes critical. In this article we focus only on the basic algorithm, leaving the study of its variants for further work. In what follows we shall call the set of edges that lie on some best path of (V, E, c_k, s, t) the *set of critical edges given k*. The rest of the edges are called non-critical.

Algorithm 1 The Greedy SPD Algorithm

```
 1: Input: $(V, E, c_0, s, t)$
 2: for all $e \in E$ do
 3:    $k(e) := 0$
 4: end for
 5: repeat
 6:    Let $\pi \in \Pi^*(V, E, c_k, s, t)$.
 7:    $u := 0$
 8:    for all $e \in \pi$ do
 9:       if $k(e) = 0$ then
10:          Call query($e$), $k(e) := 1$
11:          $u := u + 1$
12:       end if
13:    end for
14: until $u = 0$
15: Return $\pi$
```

We now show that the performance of sound algorithms that query the cost of non-critical edges cannot uniformly dominate the performance of (sound) algorithms that query the cost of critical edges only. In order to see this consider a sound algorithm $\mathcal{A}$ that is known to query non-critical edges and in order to derive a contradiction assume that $\mathcal{A}$ dominates all sound algorithms that query the cost of critical edges only. We may assume that $\mathcal{A}$ always stops as soon as possible. Let $E^* = E^*(V, E, c_k, s, t)$ denote the set of critical edges given k, $P = (V, E, c, s, t)$ and c_0. By assumption, for some SPD instance (P, c_0) algorithm $\mathcal{A}$ chooses to query edges outside of $E^*(V, E, c_k, s, t)$ at some stage of its execution. Let (P, c_0) be such an SPD instance and let n be the first stage when $\mathcal{A}$ queries some non-critical edges. Let the set of queried non-critical edges be U_n. Further, let $\pi \in \Pi^*(V, E, c_{k_n}, s, t)$ be a path such that $k|_\pi \not\equiv 0$. (By our assumption on $\mathcal{A}$ and Theorem 1 no path in $\Pi^*(V, E, c_{k_n}, s, t)$ can have all of its edges known at stage n.) Now define $P' = (V, E, c', s, t)$ such that

$$c'(e) = \begin{cases} c(e), & \text{if } k_n(e) = 1; \\ c_0(e), & \text{if } e \in \pi \text{ and } k_n(e) = 0; \\ \max(c(e), c(\pi) + 1), & \text{otherwise.} \end{cases}$$

It should be clear then that $\pi \in \Pi^*(V, E, c', s, t) \cap \Pi^*(V, E, c'_{k_n}, s, t)$ and π is the unique optimal path of P'. Now let us consider the behavior of $\mathcal{A}$ on (P', c_0). It is clear that at stage n the state of algorithm $\mathcal{A}$ will be identical to its state when it was run on (P, c_0). Therefore in time step n algorithm $\mathcal{A}$ will still decide to query the edges in U_n. However, the unique optimal path in P' is π which has no edges in U_n, hence $\mathcal{A}$ queries edges of U_n superfluously. Now, consider an algorithm $\mathcal{A}'$ that queries the cost of critical edges only. We may further assume that $\mathcal{A}'$ is such that it queries exactly the same set of edges up to stage n as algorithm $\mathcal{A}$ (such an algorithm indeed exists since n was the first time when $\mathcal{A}$ picked some non-critical edges). Without the loss of generality, we may assume that at stage n $\mathcal{A}'$ chooses to query all the unknown edges along the path π. It follows then by the construction of c' that $\mathcal{A}'$ terminates after the costs of these edges have been queried. Hence the cost of running $\mathcal{A}'$ on (P', c_0) will be strictly less than that of $\mathcal{A}$. This shows that it is not possible to give a "clever" (and sound) algorithm that queries non-critical edges and which would uniformly dominate those (sound) algorithms that query only critical edges. This result is summarized in the following proposition:

Proposition 1 *If $\mathcal{A}$ is a sound algorithm with a property such that for some SPD instances (P, c_0) $\mathcal{A}$ queries non-critical edges during its execution then there exist a sound algorithm $\mathcal{A}'$ and an SPD instance (P', c_0) such that $\mathcal{A}'$ queries only critical edges and $\|k_{\mathcal{A}'}(P', c_0)\|_1 < \|k_{\mathcal{A}}(P', c_0)\|_1$.*

Our next aim is to relax the condition $c_0 \le c$. The next proposition shows that this can be done by appropriately redefining c_k when a lower bound on the smallest non-zero difference of the cost of paths is known:

Proposition 2 *Let (P, c_0) be an SPD instance and let $k : E \to \{0, 1\}$. Assume that c is such that*

$$\min_{\pi,\pi' \in \Pi(V,E,s,t)} \{ |c(\pi) - c(\pi')| \,:\, c(\pi) - c(\pi') \ne 0 \} \ge 1. \tag{2}$$

Further, assume that $c_0(e) > 0$ holds for all $e \in E$ and define $c_k^+(e) = c_0(e) + k(e)c(e)L_0, \quad e \in E$, where $L_0 \ge \operatorname{argmax}_{\pi \in \Pi(V,E,s,t)} c_0(\pi)$. Assume that for some $\pi \in \Pi^*(V, E, c_k^+, s, t)$, $k|_\pi \equiv 1$ holds true. Then π is the solution of the SP problem (V, E, c, s, t).

Proof. Let P, c_0, π, k, L_0 be as defined above. Further, let π' be an arbitrary path from s to t in $G = (V, E)$. Then $c_0(\pi) + c(\pi)L_0 = c_k^+(\pi) \le c_k^+(\pi') \le c_0(\pi') + c(\pi')L_0$. Rearranging the terms yields $(c(\pi) - c(\pi'))\, L_0 \le c_0(\pi') - c_0(\pi) < L_0$, where we have used that $c_0(\pi') \le L_0$ and that $c_0(e) > 0$ holds for all edges $e \in E$. Dividing both sides of this inequality by L_0 gives $c(\pi) - c(\pi') < 1$ or $c(\pi) < c(\pi') + 1$. Hence from (2) we conclude that $c(\pi) \le c(\pi')$, which finishes the proof. □

Although the requirement that c_0 is a lower bound for c is dropped, we still have condition (2) which might be hard to ensure unless e.g. the set of possible costs is discrete which is the case when e.g. the costs are integer valued. The proposition below shows a way to overcome this difficulty. The price we pay is that the solutions returned will not be optimal anymore:

Proposition 3 *Fix an arbitrary $\epsilon > 0$. Let (P, c_0) be an SPD instance and let $k : E \to \{0, 1\}$. Assume that $c_0(e) > 0$ holds for all $e \in E$ and let $c_k^+(e) = c_0(e) + L_0 k(e)c(e)/\epsilon, \quad e \in E$, where L_0 is as before. Assume that for some $\pi \in \Pi^*(V, E, c_k^+, s, t)$, $k|_\pi \equiv 1$. Then π is an ϵ-optimal solution of the SP problem (V, E, c, s, t), i.e., for any path $\pi' \in \Pi(V, E, s, t)$, $c(\pi) < c(\pi') + \epsilon$.*

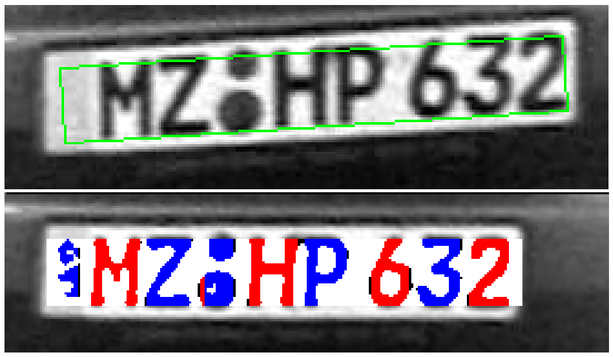
Figure 1: Original license plate image (with the detected license plate subimage framed) and the narrowest components found by the segmentation algorithm. Components are plotted in alternating colors/patterns. The segmentation boundaries are defined by the left and right edges of the narrowest components. Note that the segmentation algorithm successfully separated the letters 'H' and 'P'. However, the price is that it also cuts the stamp between the letters 'Z' and 'H' into two parts.

Proof. Let π, π' be as in the conditions of the proposition. Analogously to the previous result we obtain $(c(\pi) - c(\pi'))\, L_0/\epsilon \le c_0(\pi') - c_0(\pi) < L_0$. Dividing both sides by L_0 and rearranging the terms yields the desired inequality. □

Experiments

The validity of the approach suggested, as well as the efficiency of the proposed algorithm was tested on a specific optical character recognition (OCR) problem, namely, the problem of recognizing the character strings on vehicle license plates. The proposed algorithm was built into a commercial system. Here we report experiments with this system.

Assume that the license plate was already found by a detector algorithm and the task is to decipher the characters on the "straightened", height and orientation normalized and cleaned plate. The segmentation lattice is produced as follows: A segmentation algorithm that considers various features measured on the image such as the peak to valley ratio identifies the boundaries that may separate characters. Any position between two neighboring pixel columns is a potential candidate for such a boundary. The exact description of this algorithm is out of the scope of the present paper, however we note that this algorithm typically yields an oversegmentation of the image. An example output of this algorithm for a license plate image is given in Figure 1.

The next step is to build the segmentation lattice (or graph) that will be the input of our algorithms. This graph is built as follows: We shall distinguish two types of vertices of the graph that we shall call 'white' and 'black'. White vertices correspond to boundaries on the image identified by the segmentation algorithm, whilst a black vertex is added for each candidate character region. First the skeleton of the

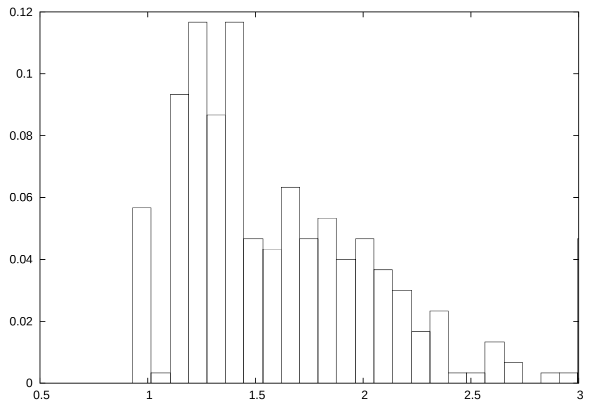

Figure 2: Histogram of the average number of black nodes per boundaries.

graph is built by adding all white vertices corresponding to all potential boundary points identified by the segmentation algorithm. No edges are added at this point. The source node, representing the 0th boundary position, and the target node representing the very last boundary position on the image are added in this step (they are white nodes). Next, all boundary point pairs that satisfy certain width constraints are considered. For a boundary pair whose corresponding vertices are v_1 and v_2, a black node, v, and the edges (v_1, v) and (v, v_2) are added to the graph. The reason of adding two edges instead of a single one is to separate the costs associated with the character region and that of the regions separating neighboring character regions. The distribution of the average number of black nodes per white nodes measured over a set of example images is shown below in figure 2.

The cost associated with character separating regions is obtained by means of some simple and quick calculations (the cost depends on e.g. the saturation of the corresponding region, if there exists a path from the top of the region to the bottom that does not cross any black pixels on the binarized image, etc.). The initial cost c_0 of a character region is set to $1 + c(R)$, where $c(R)$ depends on some other simple measures (e.g. width, aspect ratio, stroke width, existence of a white strip through the region, etc.). The idea is to penalize regions that have obviously 'wrong' features as opposed to regions that have no such trivial problems. Hence this estimate of the cost is undoubtedly a crude estimate of the true cost, but the hope is that it still should be able to guide the search process. Note that calculating this crude cost estimate is also cheap.

The solution of the recognition problem is defined as the path from the source to the target node that gives the minimal total cost, where costs are now defined as a function of the output of an optical character recognition component. In our case the OCR is built to recognize all the characters of the fonts of European license plates plus some special symbols (flags, arms, etc.). It is also capable of tolerating positioning errors up to the extent of a few pixels in all directions. The recognition component works on an appropriately binarized and cleaned image, but if the underlying imgae is highly cluttered and/or noise then the running time of the OCR component can increase substantially as it will consider alternative binarizations. This is the price that one has to pay to make the OCR component robust against substantial image noise and clutter. The cost of running the OCR component can be particularly high when many alternative "interpretations" (alignments, binarizations) exist. As a result, when due to some bad luck a non-license plate image is sent to the recognizer then the whole system may slow down. Hence the goal in this particular application is not merely to achieve a good average running time, but rather to achieve a good worst-case running time.

In our experiments we compared the performance of the Greedy SPD algorithm of the previous section with that of the algorithm that queries the costs of all the edges before starting the search for the shortest path. We shall call this algorithm the ALLCOSTS algorithm. The experiments were run on a Pentium IV 2.4GHz computer. Greedy SPD was run in the "approximate" mode with $\epsilon = 1E - 5$.[2] For the purposes of these measurements we selected a one of our test sets of images that were used in assessing the performance of the various components of the system. This set consisted of 300 images that present both easy and difficult to recognize cases. We have measured factors like the number of vertices and edges of the segmentation graph, the total time to process the plate candidate, the time spent on calculating the costs, the number of queried edges and the number of iterations required by the Greedy SPD algorithm to return a solution.

Exploiting that the graph is acyclic, shortest path computations were implemented by the standard DAG shortest path algorithm that visits edges of the graph in reverse topological order. In the case of the Greedy SPD algorithm the topological ordering was obtained in the initialization phase since the structure of the graph is kept fixed in the process. Since the graphs are typically very small (typically having less than 50 vertices) we did not implement any further optimizations here (for larger graphs incremental search algorithms could prove to be useful). The ALLCOSTS algorithm used the same shortest path subroutines.

Figure 3 shows the scatter plot of the running time of the Greedy SPD algorithm plotted against that of the ALLCOSTS algorithm. It should be evident from the figure that Greedy SPD performs better in almost all cases than ALLCOSTS and it performs substantially better than ALLCOSTS in those situations when the running-time of ALLCOSTS is large (note the log scaling of the axes on the figure). As mentioned before the observed slow-down is due to the increased effort of the OCR component when it needs to process difficult to interpret regions. It can be concluded that in these cases Greedy SPD successfully cuts down the worst-case processing time, causing speed-ups in the order of 500-4000%. On the average the speed-up factor was found to be 260% on this set of images (it should be clear that all these numbers depend heavily on the set of images used). The cases when no speed-up was measured correspond to segmentation lattices that has a single path or just a very few

[2]The algorithm was not sensitive to the particular value of ϵ.

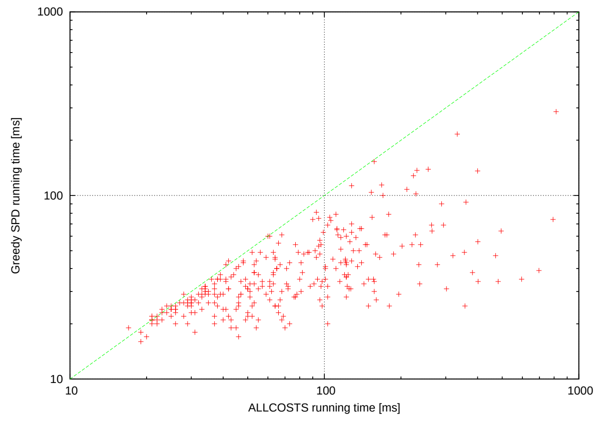

Figure 3: Scatter plot of the running time ALLCOSTS vs. the running time Greedy SPD. Note that both axis have log-scale.

paths from the source to the target. Note that the running time of the OCR component on a given region was found to be 3 ms on average, whilst the average running time of the shortest path algorithm was found to be less than 0.3 ms on average. On the average 80% of the edge cost were evaluated, with this percentage decreasing for the more difficult cases. Note that due to the choice of c_0, Greedy SPD will typically succeed at avoiding to send the most difficult to analyze characters to the OCR system – that can take longer time to process. Indeed it turned out that the average recognition time per region for those regions that were not sent to the OCR component when Greedy SPD was running then was 8.3ms, almost 3 times the time required by the OCR component to process 'normal' regions.[3]

We have also tried the version of the Greedy SPD algorithm that queries only the cost of the last unknown edge of the optimal path obtained. We have found no significant differences between the performances of these two versions.

Related Work

Two recent branches of search research, real-time search and incremental search consider related, but still radically different aspects of search. Real-time (heuristic) search methods (e.g.(Korf 1990; Furcy & Koenig 2000)) interleave planning and plan execution. They are based on agent-centered search: the set of states considered in the search is restricted to those states that can be reached from the current state of the agent within a few number of action execution steps. Real-time search methods, like LRTA* (Korf 1990) iteratively refine their heuristic function that is used in the search. Their main advantage is that they can amortize the cost of obtaining the optimal cost-to-go functions over several search episodes and are thus capable of making decisions in "real-time" whilst still converging to optimality in the long run. Real-time search is similar to shortest path discovery in that both use heuristic functions to guide search and both can learn the true costs as the search proceeds. However, the optimality criterion used in SPD is radically different from that of used in real-time search (the total number of edge-costs queried vs. the number of suboptimal actions executed). Incremental search, e.g. (Ramalingam & Reps 1996; Frigioni, Marchetti-Spaccamela, & Nanni 2000) concerns the solution of a sequence of related search problems that are only slightly different from each other in ways that are communicated to the search algorithms. The SPD problem naturally gives rise to such a sequence of related search problems. However, in incremental search one is interested in quickly solving the actual search problem by reusing as much of the previous solutions as it is possible. Thus the optimality criteria of the two search problems are again rather different. It should be obvious that SPD algorithms might use incremental search algorithms as their subroutines when they need to recalculate the shortest path after the cost of some edges have been observed.

[3]Hence the cost assigned to evaluating an edge is not uniform as it was assumed in the previous sections. Extending the algorithms to such non-uniform cases looks like an important next step.

Conclusions

We have introduced a new class of search problems, called Shortest Path Discovery Problems. An SPD instance is given by an underlying shortest path problem and an initial estimate of the costs of edges. Algorithms may query the cost of edges at any time in any order. The goal is to execute the least possible number of queries and return an optimal solution to the original shortest path problem. We have given necessary and sufficient conditions for an algorithm to be a sound SPD algorithm. We proposed an algorithm, Greedy SPD, that greedily explores all the edges of the current best path. It was shown that this algorithm is sound and we have argued that it is not possible to design a sound algorithm that would explore 'non-critical' edges and which would dominate the performance of algorithms that explore only 'critical' edges. Greedy SPD was extended to use non-admissible initial estimates at the price of returning only approximately optimal solutions. The utility of the approach was shown in a real-world experiment where Greedy SPD was shown to provide a speed-up of ca. 250%. In the particular example studied larger speed-up factors were measured in the more difficult cases.

References

Frigioni, D.; Marchetti-Spaccamela, A.; and Nanni, U. 2000. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms* 34(2):251–281.

Furcy, D., and Koenig, S. 2000. Speeding up the convergence of real-time search. In *Proceedings of the National Conference on Artificial Intelligence*, 891–897.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Ramalingam, G., and Reps, T. 1996. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms* 21(2):267–305.