Dynamic Programming Algorithms in Semiring and Hypergraph Frameworks

Liang Huang
Department of Computer and Information Science
University of Pennsylvania
lhuang3@cis.upenn.edu

November 27, 2006

Contents

Abstract

Dynamic Programming (DP) is an important class of algorithms widely used in many areas of speech and language processing. Recently there have been a series of work trying to formalize many instances of DP under algebraic or graph-theoretic frameworks. This report surveys two such frameworks, namely semirings and directed hypergraphs, and draws connections between them. We formalize two particular types of DP algorithms under these frameworks: the Viterbi-style fixed-order algorithms and the Dijkstra-style best-first algorithms. Wherever relevant, we also discuss typical applications of these algorithms in Natural Language Processing.

1 Introduction

Many algorithms in speech and language processing can be viewed as instances of *dynamic programming* (DP) (Bellman, 1957; Cormen et al., 2001). We survey two unifying frameworks for formalizing these algorithms: when the underlying representation of the search space is a (directed) graph as in finite-state machines, we use the algebraic-path framework based on semirings (Mohri, 2002); when the search space is hierarchically branching as in context-free grammars, we use directed hypergraphs (Gallo, Longo, and Pallottino, 1993) with weight functions as a generalization of graphs and semirings. In particular, we study two important types of DP algorithms under these frameworks: the Viterbi (1967)-style fixed-order algorithms, and the Dijkstra (1959)-style best-first algorithms. This report focuses on *optimization problems* where one aims to find the best solution of a problem (e.g. shortest path or highest probability derivation) but other problems will also be discussed.

2 Semirings

The definitions in this section follow Kuich and Salomaa (1986) and Mohri (2002).

Definition 1. A *monoid* is a triple $(A, \otimes, \overline{1})$ where $\otimes$ is a closed *associative binary operator* on the set A, and $\overline{1}$ is the identity element for $\otimes$, i.e., for all $a \in A, a \otimes \overline{1} = \overline{1} \otimes a = a$. A monoid is *commutative* if $\otimes$ is commutative.

Definition 2. A *semiring* is a 5-tuple $R = (A, \oplus, \otimes, \overline{0}, \overline{1})$ such that

1. $(A, \oplus, \overline{0})$ is a commutative monoid.
2. $(A, \otimes, \overline{1})$ is a monoid.
3. $\otimes$ distributes over $\oplus$: for all a, b, c in A,

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),$$
$$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).$$

4. $\overline{0}$ is an *annihilator* for $\otimes$: for all a in A, $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.

Table 1 shows some widely used examples of semirings and their applications.

Definition 3. A semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ is *commutative* if its multiplicative operator $\otimes$ is commutative.

Semiring	Set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$	intuition/application
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1	logical deduction, recognition
Viterbi	$[0,1]$	max	$\times$	0	1	prob. of the best derivation
Inside	$\mathbb{R}^{+} \cup \{+\infty\}$	$+$	$\times$	0	1	prob. of a string
Real	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$	0	shortest-distance
Tropical	$\mathbb{R}^{+} \cup \{+\infty\}$	min	$+$	$+\infty$	0	with non-negative weights
Counting	$\mathbb{N}$	$+$	$\times$	0	1	number of paths

Table 1: Examples of semirings

For example, all the semirings in Table 1 are commutative.

Definition 4. A semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ is *idempotent* if for all a in A, $a \oplus a = a$.

Idempotence leads to a comparison between elements of the semiring.

Lemma 1. Let $(A, \oplus, \otimes, \overline{0}, \overline{1})$ be an idempotent semiring, then the relation $\leq$ defined by

$$(a \leq b) \Leftrightarrow (a \oplus b = a)$$

is a partial ordering over A, called the natural order over A.

However, for optimization problems, a partial order is often not enough since we need to compare arbitrary pair of values, which requires a total ordering over A.

Definition 5. A semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ is *totally-ordered* if its natural order is a total ordering.

An important property of semirings when dealing with optimization problems is *monotonicity*, which justifies the *optimal subproblem property* in dynamic programming (Cormen et al., 2001) that the computation can be factored (into smaller problems).

Definition 6. Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is *monotonic* if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c)$$

$$(a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

Lemma 2. Let $(A, \oplus, \otimes, \overline{0}, \overline{1})$ be an idempotent semiring, then its natural order is monotonic.

In the following section, we mainly focus on totally-ordered semirings (whose natural order is monotonic).

Another (optional) property is *superiority* which corresponds to the *non-negative weights* restriction in shortest-path problems. When superiority holds, we can explore the vertices in a best-first order as in the Dijkstra algorithm (see Section 3.2).

Definition 7. Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is *superior* if for all $a, b \in A$

$$a \leq a \otimes b, \qquad b \leq a \otimes b.$$

Intuitively speaking, superiority means the combination of two elements always gets worse (than each of the two inputs). In shortest-path problems, if you traverse an edge, you always get worse cost (longer path). In Table 1, the Boolean, Viterbi, and Tropical semirings are superior while the Real semiring is not.

Lemma 3. Let $(A, \oplus, \otimes, \overline{0}, \overline{1})$ be a superior semiring with a partial order $\leq$ over A, then for all $a \in A$

$$\overline{1} \leq a \leq \overline{0}.$$

Proof. For all $a \in A$, we have $\overline{1} \leq \overline{1} \otimes a = a$ by superiority and $\overline{1}$ being the identity of $\otimes$; on the other hand, we have $a \leq \overline{0} \otimes a = \overline{0}$ by superiority and $\overline{0}$ being the annihilator of $\otimes$. $\square$

This property, called *negative boundedness* in (Mohri, 2002), intuitively illustrates the direction of optimization from $\overline{0}$, the initial value, towards as close as possible to $\overline{1}$, the best possible value.

3 Dynamic Programming on Graphs

Following Mohri (2002), we next identify the common part shared between these two algorithms as the generic shortest-path problem in graphs.

Definition 8. A *(directed) graph* is a pair $G = (V, E)$ where V is the set of vertices and E the set of edges. A *weighted (directed) graph* is a graph $G = (V, E)$ with a mapping $w : E \mapsto A$ that assigns each edge a weight from the semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$.

Definition 9. The *backward-star* $BS(v)$ of a vertex v is the set of incoming edges and the *forward-star* $FS(v)$ the set of outgoing edges.

Definition 10. A *path* π in a graph G is a sequence of consecutive edges, i.e. $\pi = e_1 e_2 \cdots e_k$ where e_i and e_{i+1} are connected with a vertex. We define the *weight* (or *cost*) of path π to be

$$w(\pi) = \bigotimes_{i=1}^{k} w(e_i) \tag{1}$$

We denote $P(v)$ to be the set of all paths from a given source vertex s to vertex v. In the remainder of the section we only consider *single-source* shortest-path problems.

Definition 11. The *best weight* $\delta(v)$ of a vertex v is the weight of the best path from the source s to v:[1]

$$\delta(v) = \begin{cases} \overline{1} & v = s \\ \bigoplus_{\pi \in P(v)} w(\pi) & v \neq s \end{cases} \tag{2}$$

For each vertex v, the current estimate of the best weight is denoted by $d(v)$, which is initialized in the following procedure:

procedure INITIALIZE(G, s)
  for each vertex $v \neq s$ **do**
    $d(v) \leftarrow \overline{0}$
  $d(s) \leftarrow \overline{1}$

The goal of a shortest-path algorithm is to repeatedly update $d(v)$ for each vertex v to some better value (based on the comparison $\oplus$) so that eventually $d(v)$ will converge to $\delta(v)$, a state we call *fixed*. For example, the generic update along an incoming edge $e = (u, v)$ for vertex v is[2]

$$d(v) \oplus= d(u) \otimes w(e) \tag{3}$$

Notice that we are using the current estimate of u to update v, so if later on $d(u)$ is updated we have to update $d(v)$ as well. This introduces the problem of *cyclic updates*, which might cause great inefficiency. To alleviate this problem, in the algorithms presented below, we will *not* trigger the update until u is fixed, so that the $u \to v$ update happens at most once.

3.1 Viterbi Algorithm for DAGs

In many NLP applications, the underlying graph exhibits some special structural properties which lead to faster algorithms. Perhaps the most common

[1] By convention, if $P(v) = \emptyset$, we have $\delta(v) = \overline{0}$.

[2] Here we adopt the C notation where $a \oplus= b$ means the assignment $a \leftarrow a \oplus b$.

of such properties is *acyclicity*, as in Hidden Markov Models (HMMs). For acyclic graphs, we can use the Viterbi (1967) Algorithm [3] which simply consists of two steps:

1. topological sort
2. visit each vertex in the topological ordering and do updates

The pseudo-code of the Viterbi algorithm is presented in Algorithm 1.

Algorithm 1 Viterbi Algorithm.

1: **procedure** VITERBI(G, w, s)
2:     topologically sort the vertices of G
3:     INITIALIZE(G, s)
4:     **for** each vertex v in topological order **do**
5:         **for** each edge $e = (u, v)$ in $BS(v)$ **do**
6:             $d(v) \oplus = d(u) \otimes w(e)$

The correctness of this algorithm (that $d(v) = \delta(v)$ for all v after execution) can be easily proved by an induction on the topologically sorted sequence of vertices. Basically, at the end of the outer-loop, $d(v)$ is fixed to be $\delta(v)$.

This algorithm is widely used in the literature and there have been some alternative implementions.

Variant 1. If we replace the backward-star $BS(v)$ in line 5 by the forward-star $FS(v)$ and modify the update accordingly, this procedure still works (see Figure 2 for pseudo-code). We refer to this variant the *forward-update* version of Algorithm 1[4]. The correctness can be proved by a similar induction (that at the beginning of the outer-loop, $d(v)$ is fixed to be $\delta(v)$).

Algorithm 2 Forward update version of Algorithm 1.

1: **procedure** VITERBI-FORWARD(G, w, s)
2:     topologically sort the vertices of G
3:     INITIALIZE(G, s)
4:     **for** each vertex v in topological order **do**
5:         **for** each edge $e = (v, u)$ in $FS(v)$ **do**
6:             $d(u) \oplus = d(v) \otimes w(e)$

[3] Also known as the Lawler (1976) algorithm in the theory community, but he considers it as part of the folklore.

[4] This is *not* to be confused with the *forward-backward* algorithm (Baum, 1972). In fact both forward and backward updates here are instances of the forward phase of a forward-backward algorithm.

Variant 2. Another popular implemention is *memoized recursion* (Cormen et al., 2001), which starts from a target vertex t and invokes recursion on sub-problems in a top-down fashion. Solved sub-problems are memoized to avoid duplicate calculation.

The running time of the Viterbi algorithm, regardless of which implemention, is $O(V + E)$ because each edge is visited exactly once.

It is important to notice that this algorithm works for all semirings as long as the graph is a DAG, although for non-total-order semirings the semantics of $\delta(v)$ is no longer "best" weight since there is no comparison. See Mohri (2002) for details.

Example 1 (Counting)**.** Count the number of paths between the source vertex s and the target vertex t in a DAG.

Solution Use the counting semiring (Table 1).

Example 2 (Longest Path)**.** Compute the longest (worst cost) paths from the source vertex s in a DAG.

Solution Use the semiring $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$.

Example 3 (HMM Tagging)**.**

3.2 Dijkstra Algorithm

The Dijkstra (1959) algorithm does not require acyclicity; however, it requires superiority of the semiring.

Algorithm 3 Dijkstra Algorithm.

1: **procedure** DIJKSTRA(G, w, s)
2:     INITIALIZE(G, s)
3:     $Q \leftarrow V[G]$ ▷ prioritized by d-values
4:     **while** $Q \neq \emptyset$ **do**
5:         $v \leftarrow$ EXTRACT-MIN(Q)
6:         **for** each edge $e = (v, u)$ in $FS(v)$ **do**
7:             $d(u) \oplus= d(v) \otimes w(e)$
8:             DECREASE-KEY(Q, u)

3.2.1 A* Algorithm for State-Space Search

In many real-world applications (especially AI search), there is a specific target vertex t and the motivation to use Dijkstra algorithm is that it can finish as soon as t is extracted out of the priority queue (line 5) because superiority guarantees the optimality of $d(t)$ at that point. The efficiency then depends (empirically) on how fast t is extracted. A very common technique to speed up this process is the A* algorithm (Hart, Nilsson, and Raphael, 1968) where we prioritize the queue using a combination

$$d(v) + \hat{h}(v)$$

of the known cost $d(v)$ from the source vertex, and an estimate $\hat{h}(v)$ of the (future) cost from v to the target t. In case the estimate is *admissible*, namely, no worse than the true future cost $h(v)$, we can prove that the optimality of $d(t)$ when t is extracted still holds.

4 Hypergraphs

Hypergraphs, as a generalization of graphs, have been extensively studied since 1970s as a powerful tool for modeling many problems in Discrete Mathematics. In this report, we use *directed hypergraphs* (Gallo, Longo, and Pallottino, 1993) to abstract a hierarchically branching search space for dynamic programming, where we solve a big problem by dividing it into (more than one) sub-problems. Classical examples of these problems include matrix-chain multiplication, optimal polygon triangulation, and optimal binary search tree (Cormen et al., 2001).

Definition 12. A *(directed) hypergraph* is a pair $H = \langle V, E \rangle$ with a set $\mathbf{R}$, where V is the set of *vertices*, E is the set of *hyperedges*, and $\mathbf{R}$ is the set of *weights*. Each hyperedge $e \in E$ is a triple $e = \langle T(e), h(e), f_e \rangle$, where $h(e) \in V$ is its *head vertex* and $T(e) \in V^*$ is an ordered list of *tail* vertices. f_e is a *weight function* from $\mathbf{R}^{|T(e)|}$ to $\mathbf{R}$.

Note that our definition differs slightly from the classical definitions of Gallo, Longo, and Pallottino (1993) and Nielsen, Andersen, and Pretolani (2005) where the tails are *sets* rather than *ordered lists*. In other words, we allow multiple occurrences of the same vertex in a tail and there is an ordering among the components. We also allow the head vertex to appear in the tail creating a self-loop which is ruled out in (Nielsen, Andersen, and Pretolani, 2005).

Definition 13. We denote $|e| = |T(e)|$ to be the *arity* of the hyperedge[5]. If $|e| = 0$, then $f_e() \in \mathbf{R}$ is a constant (f_e is a *nullary* function) and we call $h(e)$ a *source vertex*. We define the *arity* of a hypergraph to be the maximum arity of its hyperedges.

[5] The arity of e is different from its *cardinality* defined in (Gallo, Longo, and Pallottino, 1993; Nielsen, Andersen, and Pretolani, 2005) which is $|T(e)| + 1$.

A hyperedge of arity one degenerates into an edge, and a hypergraph of arity one is standard graph.

Similar to the case of graphs, in many applications presented below, there is also a distinguished vertex $t \in V$ called *target vertex*.

We can adapt the notions of backward- and forward-star to hypergraphs.

Definition 14. The *backward-star* $BS(v)$ of a vertex v is the set of incoming hyperedges $\{e \in E \mid h(e) = v\}$. The *in-degree* of v is $|BS(v)|$. The *forward-star* $FS(v)$ of a vertex v is the set of outgoing hyperedges $\{e \in E \mid v \in T(e)\}$. The *out-degree* of v is $|FS(v)|$.

Definition 15. The *graph projection* of a hypergraph $H = \langle V, E, t, \mathbf{R} \rangle$ is a directed graph $G = \langle V, E' \rangle$ where

$$E' = \{(u, v) \mid \exists e \in BS(v), \text{s.t. } u \in T(e)\}.$$

A hypergraph H is *acyclic* if its graph projection G is acyclic; then a *topological ordering* of H is an ordering of V that is a topological ordering in G.

4.1 Weight Functions and Semirings

Definition 16. A function $f : \mathbf{R}^m \mapsto \mathbf{R}$ is *monotonic* with regarding to $\preceq$, if for all $i \in 1..m$

$$(a_i \preceq a'_i) \Rightarrow f(a_1, \cdots, a_i, \cdots, a_m) \preceq f(a_1, \cdots, a'_i, \cdots, a_m).$$

Definition 17. A hypergraph H is *monotonic* if there is a total ordering $\preceq$ on $\mathbf{R}$ such that every weight function f in H is monotonic with regarding to $\preceq$. We can borrow the additive operator $\oplus$ from semiring to define a comparison operator

$$a \oplus b = \begin{cases} a & a \preceq b, \\ b & \text{otherwise.} \end{cases}$$

In this paper we will assume this monotonicity, which corresponds to the optimal substructure property in dynamic programming (Cormen et al., 2001).

Definition 18. A function $f : \mathbf{R}^m \mapsto \mathbf{R}$ is *superior* if the result of function application is worse than each of its argument:

$$\forall i \in 1..m,\ a_i \preceq f(a_1, \cdots, a_i, \cdots, a_m).$$

A hypergraph H is superior if every weight function f in H is superior.

4.2 Derivations

To do optimization we need to extend the notion of paths in graphs to hypergraphs. This is, however, not straightforward due to the assymmetry of the head and the tail in a hyperedge and there have been multiple proposals in the literature. Here we follow the recursive definition of *derivations* in (Huang and Chiang, 2005). See Section 6 for the alternative notion of hyperpaths.

Definition 19. A *derivation* D of a vertex v in a hypergraph H, its size $|D|$ and its weight $w(D)$ are recursively defined as follows:

- If $e \in BS(v)$ with $|e| = 0$, then $D = \langle e, \epsilon \rangle$ is a derivation of v, its size $|D| = 1$, and its weight $w(D) = f_e()$.
- If $e \in BS(v)$ where $|e| > 0$ and D_i is a derivation of $T_i(e)$ for $1 \leq i \leq |e|$, then $D = \langle e, D_1 \cdots D_{|e|} \rangle$ is a derivation of v, its size $|D| = 1 + \sum_{i=1}^{|e|} |D_i|$ and its weight $w(D) = f_e(w(D_1), \ldots, w(D_{|e|}))$.

The ordering on weights in $\mathbf{R}$ induces an ordering on derivations: $D \preceq D'$ iff $w(D) \preceq w(D')$.

We denote $\mathscr{D}(v)$ to be the set of derivations of v and extend the *best weight* in definition 11 to hypergraph:

Definition 20. The *best weight* $\delta(v)$ of a vertex v is the weight of the best derivation of v:

$$\delta(v) = \begin{cases} \overline{1} & v \text{ is a source vertex} \\ \bigoplus_{D \in \mathscr{D}(v)} w(D) & \text{otherwise} \end{cases} \tag{4}$$

4.3 Related Formalisms

Hypergraphs are closely related to other formalisms like AND/OR graphs, context-free grammars, and deductive systems (Shieber, Schabes, and Pereira, 1995; Nederhof, 2003).

In an AND/OR graph, the OR-nodes correspond to vertices in a hypergraph and the AND-nodes, which links several OR-nodes to another OR-node, correspond to a hyperedge. Similarly, in context-free grammars, nonterminals are vertices and productions are hyperedges; in deductive systems, items are vertices and instantied deductions are hyperedges. Table 2 summaries these correspondences. Obviously one can construct a corresponding hypergraph for any given AND/OR graph, context-free grammar, or deductive system. However, the hypergraph formulation provides greater

hypergraph	AND/OR graph	context-free grammar	deductive system
vertex	OR-node	symbol	item
source-vertex	leaf OR-node	terminal	axiom
target-vertex	root OR-node	start symbol	goal item
hyperedge	AND-node	production	instantiated deduction
$(\{u_1, u_2\}, v, f)$		$v \xrightarrow{f} u_1\ u_2$	$\dfrac{u_1 : a \quad u_2 : b}{v : f(a, b)}$

Table 2: Correspondence between hypergraphs and related formalisms.

modeling flexibility than weighted deductive systems of Nederhof (2003): in the former we can have a separate weight function for each hyperedge, where as in the latter, the weight function is defined for a deductive (template) rule which corresponds to many hyperedges.

5 Dynamic Programming on Hypergraphs

Since hypergraphs with weight functions are generalizations of graphs with semirings, we can extend the algorithms in Section 3 to the hypergraph case.

5.1 Generalized Viterbi Algorithm

The Viterbi Algorithm (Section 3.1) can be adapted to acyclic hypergraphs almost without modification (see Algorithm 4 for pseudo-code).

Algorithm 4 Generalized Viterbi Algorithm.

procedure General-Viterbi(H)
    topologically sort the vertices of H
    Initialize(H)
    for each vertex v in topological order **do**
        for each hyperedge e in $BS(v)$ **do**
            e is $(\{u_1, u_2, \cdots, u_{|e|}\}, v, f_e)$
            $d(v) \oplus = f_e(d(u_1), d(u_2), \cdots, d(u_{|e|}))$

The correctness of this algorithm can be proved by a similar induction. Its time complexity is $O(V + E)$ since every hyperedge is visited exactly once (assuming the arity of the hypergraph is a constant).

The forward-update version of this algorithm, however, is not as trivial as the graph case. This is because the tail of a hyperedge now contains several vertices and thus the forward- and backward-stars are no longer symmetric. The naive adaption would end up visiting a hyperedge many times. To ensure that a hyperedge e is fired only when all of its tail vertices have been fixed to their best weights, we maintain a counter $r[e]$ of the remaining vertices yet to be fixed (line 5) and fires the update rule for e when $r[e] = 0$ (line 9). This method is also used in the Knuth algorithm (Section 5.2).

Algorithm 5 Forward update version of Algorithm 4.

1: **procedure** GENERAL-VITERBI-FORWARD(H)
2:     topologically sort the vertices of H
3:     INITIALIZE(H)
4:     **for** each hyperedge e **do**
5:         $r[e] \leftarrow |e|$ ▷ counter of remaining tails to be fixed
6:     **for** each vertex v in topological order **do**
7:         **for** each hyperedge e in $FS(v)$ **do**
8:             $r[e] \leftarrow r[e] - 1$
9:             **if** $r[e] == 0$ **then**   ▷ all tails have been fixed
10:                 e is $(\{u_1, u_2, \cdots, u_{|e|}\}, h(e), f_e)$
11:                 $d(h(e)) \oplus = f_e(d(u_1), d(u_2), \cdots, d(u_{|e|}))$

5.1.1 CKY Algorithm

The most widely used algorithm for parsing in NLP, the CKY algorithm (Kasami, 1965), is a specific instance of the Viterbi algorithm for hypergraphs. The CKY algorithm takes a context-free grammar G in Chomsky Normal Form (CNF) and essentially intersects G with a DFA D representing the input sentence to be parsed. The resulting search space by this intersection is an acyclic hypergraph whose vertices are items like (X, i, j) and whose hyperedges are instantiated deduction like $(\{(Y, i, k)(Z, k, j)\}, (X, i, j), f)$ for all $i < k < j$ if there is a production $X \rightarrow YZ$. The weight function f is simply

$$f(a, b) = a \otimes b \otimes w(X \rightarrow YZ).$$

The Chomsky Normal Form ensures acyclicity of the hypergraph but there are multiple topological orderings which result in different variants of the CKY algorithm:

1. bottom-up CKY
2. left-to-right CKY
3. left-corner CKY

5.2 Knuth Algorithm

Knuth (1977) generalizes the Dijkstra algorithm to what he calls the *grammar problem*, which essentially corresponds to the search problem in a monotonic superior hypergraph (see Table 2). However, he does not provide an efficient implementation nor analysis of complexity. Graehl and Knight (2004) present an implementation that runs in time $O(V \log V + E)$ using the method described in Algorithm 5 to ensure that every hyperedge is visited only once (assuming the priority queue is implemented as a Fibonaaci heap; for binary heap, it runs in $O((V + E) \log V)$).

Algorithm 6 Knuth Algorithm.

procedure KNUTH(H)
    INITIALIZE(H)
    $Q \leftarrow V[H]$ ▷ prioritized by d-values
    for each hyperedge e **do**
        $r[e] \leftarrow |e|$
    while $Q \neq \emptyset$ **do**
        $v \leftarrow$ EXTRACT-MIN(Q)
        for each edge e in $FS(v)$ **do**
            e is $(\{u_1, u_2, \cdots, u_{|e|}\}, h(e), f_e)$
            $r[e] \leftarrow r[e] - 1$
            if $r[e] == 0$ **then**
                $d(h(e)) \oplus = f_e(d(u_1), d(u_2), \cdots, d(u_{|e|}))$
                DECREASE-KEY($Q, h(e)$)

5.2.1 A* Algorithm for Hypergraph

We can also extend the A* algorithm to hypergraphs when the weight functions factor to semiring operations. A specific case of this algorithm is the A* parsing of Klein and Manning (2003) where they achieve significant speed up using some carefully designed heuristic functions.

6 Extensions and Discussions

In most of the above we focus on optimization problems where one aims to find the best solution. Here we consider two extensions of this scheme: *non-optimization problems* where the goal is often to compute the summation or closure, and *k-best problems* where one also searches for the 2nd, 3rd, through kth-best solutions. Both extensions have many applications in NLP. For the former, algorithms based on the Inside semiring (Table 1), including the forward-backward algorithm (Baum, 1972) and Inside-Outside algorithm (Baker, 1979; Lari and Young, 1990) are widely used for unsupervised training with the EM algorithm (Dempster, Laird, and Rubin, 1977). For the latter, since NLP is often a pipeline of several modules, where the 1-best solution from one module might *not* be the best input for the next module, and one prefers to postpone disambiguation by propogating a k-best list of candidates (Collins, 2000; Gildea and Jurafsky, 2002; Charniak and Johnson, 2005; Huang and Chiang, 2005). The k-best list is also frequently used in discriminative learning to approximate the whole set of candidates which is usually exponentially large (Och, 2003; McDonald, Crammer, and Pereira, 2005).

6.1 Beyond Optimization Problems

We know that in optimization problems, the criteria for using dynamic programming is *monotonicity* (definitions 6 and 16). But in non-optimization problems, since there is no comparison, this criteria is no longer applicable. Then when can we apply dynamic programming to a non-optimization problem?

Cormen, Leiserson, and Rivest (1990) develop a more general criteria of *closed semiring* where $\oplus$ is idempotent and infinite sums are well-defined and present a more sophisticated algorithm that can be proved to work for all closed semirings. This definition is still not general enough since many non-optimization semirings including the Inside semiring are not even idempotent. Mohri (2002) solves this problem by a slightly different definition of *closedness* which does not assume idempotence. His generic single-source algorithm subsumes many classical algorithms like Dijkstra, Bellman-Ford (Bellman, 1958), and Viterbi as specific instances.

It remains an open problem how to extend the *closedness* definition to the case of weight functions in hypergraphs.

6.2 k-best Extensions

The straightforward extension from 1-best to k-best is to simply replace the old semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ by its k-best version $(A^k, \oplus^k, \otimes^k, \overline{0}^k, \overline{1}^k)$ where each element is now a vector of length k, with the ith component represent the ith-best value. Since $\oplus$ is a comparison, we can define $\oplus^k$ to be the top-k elements of the $2k$ elements from the two vectors, and $\otimes^k$ the top-k elements of the k^2 elements from the cross-product of two vectors:

$$a \oplus^k b = \oplus'_k(\{a_i \mid 1 \leq i \leq k\} \cup \{b_j \mid 1 \leq j \leq k\})$$

$$a \otimes^k b = \oplus'_k\{a_i \otimes b_j \mid 1 \leq i, j \leq k\}$$

where $\oplus'_k$ returns the *ordered list* of the top-k elements in a set. A similar construction is obvious for the weight functions in hypergraphs.

Now we can re-use the 1-best Viterbi Algorithm to solve the k-best problem in a generic way, as is done in (Mohri, 2002). However, some more sophisticated techniques that breaks the modularity of semirings results in much faster k-best algorithms. For example, the Recursive Enumeration Algorithm (REA) (Jiménez and Marzal, 1999) uses a lazy computation method on top of the Viterbi algorithm to efficiently compute the ith-best solution based on the 1st, 2nd, ..., $(i-1)$th solutions. A simple k-best Dijkstra algorithm is described in (Mohri and Riley, 2002).

For the hypergraph case, the REA algorithm has been adapted for k-best derivations (Jiménez and Marzal, 2000; Huang and Chiang, 2005). Applications of this algorithm include k-best parsing (McDonald, Crammer, and Pereira, 2005; Mohri and Roark, 2006) and machine translation (Chiang, 2007). It is also implemented as part of Dyna (Eisner, Goldlust, and Smith, 2005), a generic langauge for dynamic programming. The k-best extension of the Knuth Algorithm is studied by Huang (2005). A separate problem, k-shortest hyperpaths, has been studied by Nielsen, Andersen, and Pretolani (2005).

Eppstein (2001) compiles an annotated bibliography for k-shortest-path and other related k-best problems.

7 Conclusion

This report surveys two frameworks for formalizing dynamic programming and presents two important classes of DP algorithms under these frameworks. We focused on 1-best optimization problems but also discussed other scenarios like non-optimization problems and k-best solutions. We believe that a better understanding of the theoretical foundations of DP is benefitial for NLP researchers.

References

Baker, James K. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550.

Baum, L. E. 1972. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process. *Inequalities*, (3).

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

Bellman, Richard. 1958. On a routing problem. *Quarterly of Applied Mathematics*, (16).

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine-grained n-best parsing and discriminative reranking. In *Proceedings of the 43rd ACL.*

Chiang, David. 2007. Hierarchical phrase-based translation. In *Computational Linguistics.* To appear.

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*, pages 175–182.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms.* first edition. MIT Press.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms.* second edition. MIT Press.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38.

Dijkstra, Edsger W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, (1):267–271.

Eisner, Jason, Eric Goldlust, and Noah A. Smith. 2005. Compiling comp ling: Weighted dynamic programming and the dyna language. In *Proceedings of HLT-EMNLP.*

Eppstein, David. 2001. Bibliography on k shortest paths and other "k best solutions" problems. http://www.ics.uci.edu/∼eppstein/bibs/kpath.bib.

Gallo, Giorgio, Giustino Longo, and Stefano Pallottino. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201.

Gildea, Daniel and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

Graehl, Jonathan and Kevin Knight. 2004. Training tree transducers. In *HLT-NAACL*, pages 105–112.

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

Huang, Liang. 2005. k-best Knuth algorithm and k-best A* parsing. Unpublished manuscript.

Huang, Liang and David Chiang. 2005. Better k-best Parsing. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT-2005).*

Jiménez, Víctor and Andrés Marzal. 1999. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Algorithm Engineering*, pages 15–29.

Jiménez, Víctor M. and Andrés Marzal. 2000. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proc. of the Joint IAPR International Workshops on Advances in Pattern Recognition.*

Kasami, T. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA†.

Klein, Dan and Chris Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of HLT-NAACL.*

Knuth, Donald. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1).

Kuich, W. and A. Salomaa. 1986. *Semirings, Automata, Languages.* EATCS Monographs on Theoretical Computer Science, number 5. Berlin, Germany: Springer-Verlag.

Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Lawler, E. L. 1976. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd ACL.*

Mohri, Mehryar. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Mohri, Mehryar and Michael Riley. 2002. An efficient algorithm for the n-best-strings problem. In *Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP '02)*, Denver, Colorado, September.

Mohri, Mehryar and Brian Roark. 2006. Probabilistic context-free grammar induction based on structural zeros. In *Proceedings of HLT-NAACL*.

Nederhof, Mark-Jan. 2003. Weighted deductive parsing and Knuth's algorithm. 29(1):135–143.

Nielsen, Lars Relund, Kim Allan Andersen, and Daniele Pretolani. 2005. Finding the k shortest hyperpaths. *Computers and Operations Research*.

Och, Franz Joseph. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167.

Shieber, Stuart, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Viterbi, Andrew J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, April.