Speeding up Stochastic Dynamic Programming with Zero-Delay Convolution

Brian C. Dean

M.I.T., Cambridge, MA 02139, USA,
bdean@theory.lcs.mit.edu

Abstract

We show how a technique from signal processing known as zero-delay convolution can be used to develop more efficient dynamic programming algorithms for a broad class of stochastic optimization problems. This class includes several variants of discrete stochastic shortest path, scheduling, and knapsack problems, all of which involve making a series of decisions over time that have stochastic consequences. We also correct a flaw in the original analysis [8] of the zero-delay convolution algorithm.

1 Introduction

By allowing for uncertainty in the input data for a problem, stochastic optimization problems are capable of modeling much more realistic scenarios than their deterministic counterparts. Unfortunately this generality comes at a price, as stochastic problems tend to be much more computationally intensive to solve. If we focus on discrete probability distributions, many stochastic optimization problems can be solved using dynamic programming (DP), although for large instances straightforward DP algorithms often do not run fast enough to be useful in practice. In this paper we advocate the use of a technique from signal processing known as zero-delay convolution to substantially speed up DP algorithms for a broad class of stochastic optimization problems. In the literature many clever techniques have been proposed for speeding up classes of DP problems — see [7] for a good exposition of many of these results. However, there seem to be few such universal techniques that have specific application to stochastic problems.

The class of problems we consider includes the stochastic shortest path problem and several variants of stochastic scheduling and knapsack problems. In the stochastic shortest path problem, the travel times along edges in a graph are described by independent, integer-valued random variables, whose distributions are provided as input. Given a source node s, a destination node d, and a deadline T, and wish to find a route from s to t that has maximum probability of arriving within T time units. The problem is "adaptive" in the sense that once we follow an edge we realize its travel time, and depending on this outcome we have the option to select a different route for the remainder of the journey. Zero-delay convolution will allow us to reduce the running time for solving this problem from $O(mT^2)$ to $O(mT \log^2 T)$ for an m-edge graph.

There are many ways to generalize the classical knapsack problem by introducing randomness. We introduce one such variant here, which is also an example of a stochastic scheduling problem, and several more in Section 4. Suppose again that we have a fixed deadline of T time units, and a collection of n types of tasks we might choose to perform. Each task type has an associated value as well as an integer-valued duration described by a discrete probability distribution, where durations of different tasks are independent random variables. At the beginning of time we select some task type to perform, thereby receiving its associated value and realizing its duration. Again, we have an adaptive problem where based on the duration required by the first task we can select a second task type (multiple tasks of the same type may be selected), and so on. The process stops when the deadline is reached, and no value is received from the final interrupted task. One can think of this problem either as a type of stochastic scheduling problem with a fixed deadline, or alternatively as a stochastic knapsack problem where items sizes are random variables. Zero-delay convolution reduces the DP running time for this problem and many other variants from $O(nT^2)$ to $O(nT \log^2 T)$.

The remainder of this paper is structured as follows. In Section 2 we review the zero-delay convolution technique in the context of dynamic programming, and we correct a flaw in the original analysis of the technique in [8]. We then study stochastic shortest path problems in Section 3 and stochastic scheduling and knapsack problems in Section 4.

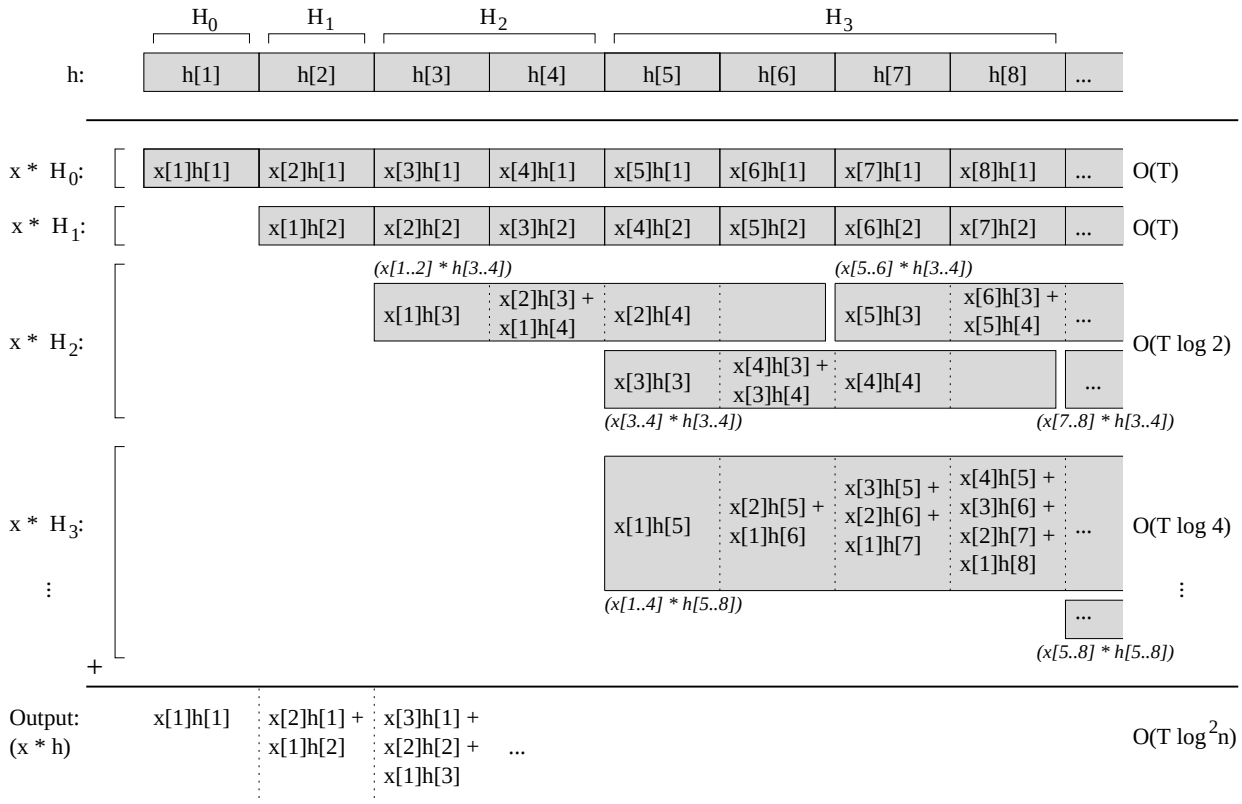

Figure 1: Illustration of the zero-delay convolution algorithm. We divide h into blocks $H_0, H_1, H_2, \ldots$ of exponentially increasing size (except for the first two, both of size 1). Each of these blocks is then convolved, in parallel, against x, using buffering and block convolution. The output, $x * h$ (we denote by $*$ the convolution operation), is obtained by summing these partial results. Running times are specified in the rightmost column, summing to $O(T \log^2 n)$. Applied to our example of computing $A[1 \ldots T]$, we would take each output element, increment it, and feed it back in as the next input element.

2 Zero-Delay Convolution

We illustrate the technique of zero-delay convolution applied to DP using a simple example. Let $h[1 \ldots n]$ be the probability distribution of a discrete random variable X, so $h[i] = \Pr[X = i]$. What is the expected number of independent samples of X one can draw until their sum reaches some threshold $T \geq n$? We can answer this question using a simple dynamic program. Letting $A[j]$ be the expected number of independent samples required to reach a sum of j, we have

$$A[j] = 1 + \sum_{i=1}^{n} A[j-i]h[i], \tag{1}$$

where $A[j \leq 0] = 0$ as a base case. Straightforward computation of $A[1 \ldots T]$ by direct application of (1) requires $O(nT)$ time. To improve this, we think of (1) as a special type of convolution between two sequences A and h. In a standard convolution problem we are given as input two pre-specified sequences to convolve. However, in this case the sequence A is actually generated in an on-line fashion, with each successive element of A depending on the results of the convolution so far. From a signal processing standpoint, we can picture this as a discrete-time system with impulse response h (so any signal fed through the system is convolved with h) where each output element is immediately incremented and fed back as the next element of the input signal.

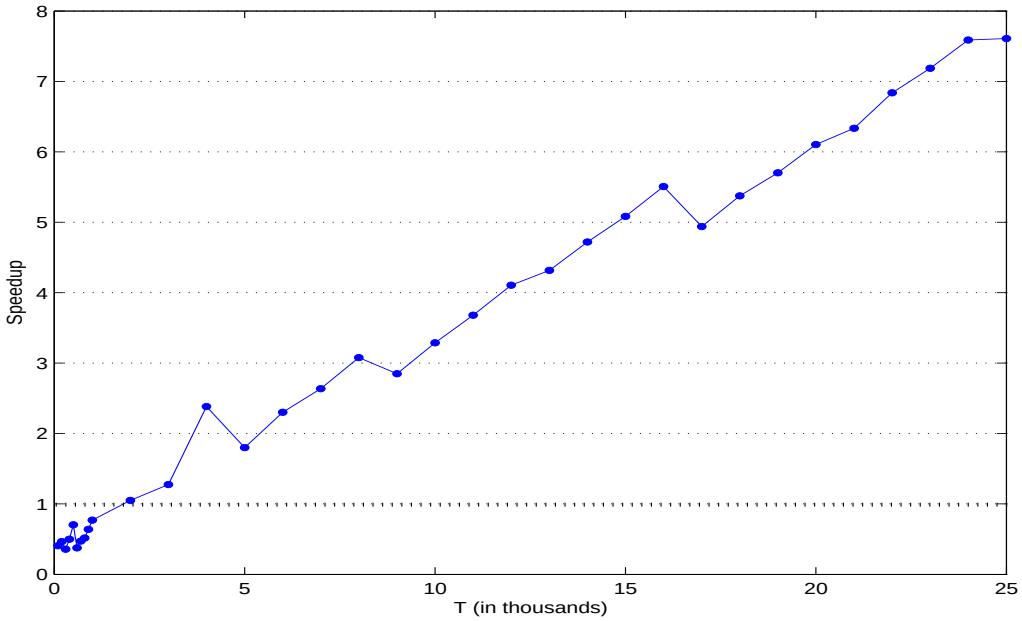

Figure 2: Speedup obtained by using zero-delay convolution. The small drops in performance for zero-delay convolution occur right after we cross a power of 2 in T, since this causes the zero-delay algorithm to perform more quite a few more FFTs on larger subarrays.

A fundamental computation performed by discrete-time signal processing devices is the convolution of a long input sequence $x[1 \ldots T]$ with the impulse response $h[1 \ldots n]$ of the system. Since the Fast Fourier Transform (FFT) can convolve two length-n signals in $O(n \log n)$ time, the usual approach is to buffer x into T/n length-n blocks and to convolve each of these in sequence against h using the FFT. This requires only $O(\log n)$ time per output element, but it has the unpleasant side effect of introducing some input/output delay due to the buffering of x; that is, we must wait for n elements of x to arrive at the system before we can produce any output. Input/output delay is undesirable for obvious reasons in many signal processing applications, and it is absolutely unacceptable for our problem since we cannot advance the input signal until the complete result of the convolution thus far is determined.

In 1995 Gardner [8] introduced a zero-delay convolution technique that eliminates input/output delay at the expense of only a small running time penalty, producing each output element in $O(\log^2 n)$ amortized time (this is incorrectly analyzed as $O(\log n)$ time in Gardner's paper). Assume for simplicity and without loss of generality that n is a power of 2, and break the impulse response h into blocks of exponentially increasing size (except the first two, which are both of size 1), as shown in Figure 1. We then launch separate convolutions that move forward in parallel between each block and the entire input sequence. By adding together the results of these sub-convolutions, we obtain the final convolution of x with h. We convolve each block of h against x using the standard buffering approach: for a block of size B we buffer x into T/B blocks of size B and convolve them each in sequence using the FFT, which requires a total of $O(\frac{T}{B} \times B \log B) = O(T \log B)$ time. Note that the block decomposition of h is designed so that buffering does not contribute to input/output delay. For example, in Figure 1 when we convolve x with H_3 we initially buffer the values $x[1 \ldots 4]$, but the result of convolving these buffered elements against H_3 is not used in computing elements $1 \ldots 4$ of the output. The total running time spent convolving x with all of the blocks of h is

$$\sum_{i=1}^{\log n} O(T \log \frac{n}{2^i}) = O(T \log^2 n),$$

which amortizes to $O(\log^2 n)$ time per output element. Applying this technique to our original problem of computing of $A[1 \ldots T]$, we obtain an $O(T \log^2 n)$ algorithm.

In order to develop a sense of how well the zero-delay convolution algorithm performs in practice, Figure 2 shows the results of a simple computational experiment. Zero-delay convolution was implemented in C using the well-known FFTW3.0.1 Fast Fourier Transform library of Frigo and Johnson, and compared against the "naïve" convolution approach for the sample problem above, setting $n = T$ (so the theoretical running times are $O(T \log^2 T)$ for zero-delay convolution and $O(T^2)$ for the naïve approach). The zero-delay approach seems to become superior for $T > 2000$ and improve from there. For problems that are finely-discretized in the time dimension, a value of T in the mid thousands is entirely conceivable in practice.

3 The Stochastic Shortest Path Problem

The input to the stochastic shortest path (SSP) problem consists of a graph $G = (N, A)$ with $n = |N|$ nodes and $m = |A|$ directed arcs, where the length of each edge $(i, j) \in A$ is a discrete positive-integer-valued random variable l_{ij} whose distribution we denote $L_{ij}[\cdot]$, so $L_{ij}[t] = \mathbf{Pr}[l_{ij} = t]$. We are interested in finding an optimal path from a source node s to a destination node d.

Since the length of every $s \rightsquigarrow d$ path is a random variable (obtained by convolving together the distributions of the edge lengths along the path), it is not immediately clear what should be called an "optimal" path. Several reasonable objective functions have been proposed in the literature — see [13] and [1] for a detailed account. Some of the more prominent choices include (i) the path of minimum expected length, (ii) the path having maximum probability of being the shortest [15], (iii) the path whose expected deviation (or squared deviation) from the shortest path length is minimized [10], and (iv) the path having maximum probability of arrival by a specified deadline of T time units [6]. Of these objectives, only (i) appears to be computationally manageable as it is equivalent to a deterministic shortest path problem where every edge length distribution is replaced with its expectation. The author is not aware of any complexity results for (ii)-(iii), and (iv) can be shown to be #P-hard, even if edge lengths have Bernoulli distributions (only two possible lengths), using a result of Kleinberg et al. [11].

In this paper we focus on objective (iv): maximizing the probability of arrival by a specified deadline T. Although the computation of a static optimal path according to this objective seems difficult, a DP solution exists if we consider a dynamic, or "adaptive", variant: rather than selecting an entire path apriori, our algorithm initially selects only the first edge to follow. After following this edge and realizing its travel time, the algorithm choose a second edge, and so on. Hall [9] calls such an approach a "time-adaptive route choice" algorithm. We note that there are instances of the SSP problem in which the best adaptive strategy gives an exponentially larger probability (in the size of the graph, n) over the best "non-adaptive" static path of successfully arriving by the deadline (Figure 3).

3.1 Dynamic Programming Formulation

Let $P_i[t]$ denote the probability of arriving at d within t time units if we are currently at node i and henceforth follow an optimal (adaptive) route. Likewise let $P_{ij}[t]$ denote the optimal arrival probability if we depart along edge (i, j) at time t. We then have

$$P_i[t] = \begin{cases} 1 & \text{if } i = d \\ \max\{P_{ij}[t] : (i,j) \in A\} & \text{if } i \neq d \end{cases} \tag{2}$$

$$P_{ij}[t] = \sum_{\tau=1}^{t} L_{ij}[\tau] P_j[t - \tau] \tag{3}$$

where $P_i[t < 0] = P_{ij}[t < 0] = 0$ as a base case. A straightforward DP algorithm that repeatedly applies (2) and (3) for increasingly larger values of t spends $O(mT)$ total time evaluating (2) and $O(mT^2)$ total time evaluating (3). However, notice that (3) is of the same form as (1): we are convolving the fixed distribution

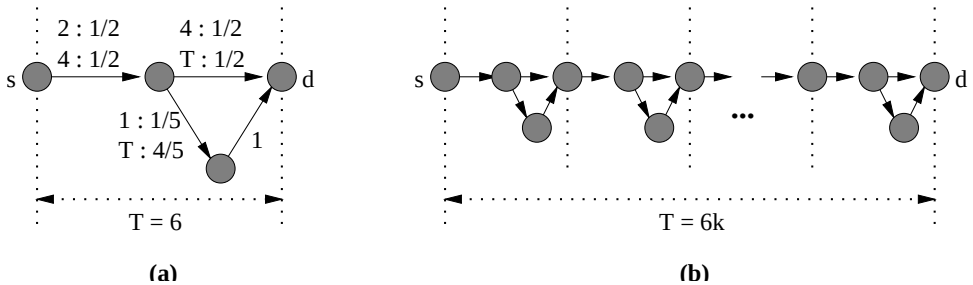

Figure 3: (a) An instance of the stochastic shortest path problem for which a better probability of arrival by a deadline T is possible if adaptive routing is allowed. The best non-adaptive path (the straight edges across the top of the graph) arrives on time with probability 1/4, where the best adaptive route arrives on time with probability at least 7/20. By concatenating k copies of (a), we see in (b) an instance where the non-adaptive and adaptive probabilities are $(1/4)^k$ and at least $(7/20)^k$, so our adaptiv ity gap is exponentially large factor in the size of the graph.

$L_{ij}[\cdot]$ against a sequence $P_j[\cdot]$ that is appearing one element at a time as we increment t. We can therefore use one instance of Gardner's zero-delay convolution algorithm for every edge to evaluate (3) in a total of only $O(mT \log^2 T)$ time. Moreover, we *must* employ a type of zero-delay convolution due to the interleaved structure of the formulation.

3.2 Some Extensions

If we assign a cost c_{ij} to every edge $(i, j) \in A$, we can consider the adaptive route choice problem whose goal is to minimize the expected cost of traveling from s to d, where this cost becomes equal to some large penalty M if we take longer than T units of time. The DP formulation for the adaptive variant of this problem is similar to the one above; if we redefine $P_i[t]$ and $P_{ij}[t]$ to be optimal expected path costs, we obtain

$$P_i[t] = \begin{cases} 0 & \text{if } i = d \\ \min\{P_{ij}[t] : (i,j) \in A\} & \text{if } i \neq d \end{cases} \quad (4)$$

$$P_{ij}[t] = c_{ij} + \sum_{\tau=1}^{t} L_{ij}[\tau] P_j[t-\tau], \quad (5)$$

where $P_i[t < 0]$ and $P_{ij}[t < 0]$ are as a base case set to some large specified penalty M. This variant can also be solved in $O(mT \log^2 T)$ time using zero-delay convolution.

We can make the edge costs time-dependent by replacing c_{ij} with a function $c_{ij}[t]$. By splitting d into two nodes d' and d connected by a zero-length edge with time-dependent cost $c_{d'd}[t] = f[t]$ we can impose an penalty $f[t]$ based on the arrival time at the destination — for example, one might wish to impose a soft deadline $T' < T$ after which arrival is acceptable but penalized based on the extent to which the soft deadline is violated.

Finally, we have been assuming that edge lengths are strictly positive in order to ensure the acyclicity of our DP formulation. This assumption can be relaxed, however. Let G' denote the subgraph of G consisting of edges (i, j) for which $L_{ij}[0] > 0$. If G' is acyclic we can still solve the problem with no increase in asymptotic running time by processing nodes within each "time level" in a topological ordering. If G' is not acyclic, then we do incur a penalty in running time as it is necessary to solve a generalized shortest path problem within each time level as the DP algorithm progresses.

3.3 Sparse Distributions

If we describe the probability distributions L_{ij} that comprise the input to an instance as length-T vectors, the running time of $O(mT \log^2 T)$ is only a polylogarithmic factor larger than the $O(mT)$ time required to read the entire input. However, in practice we are likely to find instances in which many edges have either fixed deterministic lengths or very "sparse" length distributions. If any distribution L_{ij} has $o(\log^2 T)$ non-zero elements we will find it more efficient to evaluate (3) directly for this edge rather than to use zero-delay convolution. Furthermore, if the distribution of L_{ij} does not range from 1 to T but rather fits into a smaller range $[r, R]$, we can reduce the $\log^2 T$ factor for that edge in the zero-delay convolution running time to $\log R \log \frac{R}{r}$. These observation also apply to the stochastic knapsack problem (described in the next section) with sparse distributions.

4 Stochastic Knapsack and Scheduling Problems

The classical knapsack problem involves a set of n items with sizes $s_1 \ldots s_n$ and values $v_1 \ldots v_n$, where our goal is to maximize the value we can pack into a knapsack of capacity C. There are two common variants of the problem: one in which multiple copies of items can be placed in the knapsack, and the so-called 0/1 knapsack problem in which at most one copy of each item is allowed. In addition to the more common packing formulation, both of these variants can be rephrased as covering problems, where items have costs $c_1 \ldots c_n$ rather than values, and we wish to compute a minimum-cost collection of items whose total size is at least C. If items sizes are integral, all of these problems can be solved in $O(nC)$ time with simple DP algorithms.

Stochastic knapsack problems involve item sizes and/or item values/costs that are independent random variables with known distributions. In this section we will focus primarily on the case where sizes are integer-valued random variables and values/costs are deterministic, since this is the case where zero-delay convolution applies. As an example, let us consider the stochastic "multiple copies allowed" knapsack packing problem. We interpret our n items as different types of tasks, where the duration of each task is an independent random variable of known distribution, and each task type, if performed by a specified deadline T, offers us a deterministic value. Our solution to the problem will be an adaptive scheduling policy, where at the outset we select only the first task to perform. After realizing the duration of the first task, we select a second task type (multiple tasks of the same type may be selected) based on the amount of time remaining before the deadline, and so on, until eventually we run out of time and receive no value for the final interrupted task. Our goal is to compute a policy that maximizes the expected value we receive. Although we motivated this problem as a stochastic knapsack problem, we could just have easily approached it as a type of stochastic scheduling problem. Stochastic scheduling problems involve optimally sequencing a set of tasks with probabilistic durations — see [17, 3] for a thorough discussion. To date, the only results in the stochastic scheduling literature that focus on the knapsack objective (maximizing the expected value of tasks scheduled by a fixed deadline) consider only special classes of probability distributions, typically only those for which optimal analytic solutions are possible.

As another example, consider the stochastic "multiple copies allowed" knapsack cover problem. Derman et al. [5] describe an excellent motivation for this problem in which we have a machine (e.g. an automobile) that must be kept running for T units of time and that depends on some critical component (e.g. a battery) to run. We have n types of components from which to choose, each with a deterministic cost and each having a lifetime described by a random variable of known distribution. Our task is to devise an optimal adaptive policy that selects a sequence of components over time that keeps the machine running at minimum expected cost. Derman et al. [5] study instances of this problem in which component durations are exponentially distributed, as well as instances in which one type of component is available in limited supply and the remaining types are unlimited.

Regarding previous results from the literature for the stochastic knapsack problem, the first dynamic programming formulation for the stochastic knapsack problem appears in Steinberg and Parks [16], who consider problems where sizes are deterministic and costs are random variables, with a goal of packing the knapsack

with a collection of items that maximizes the probability of achieving some target value V (we consider this variant in Section 4.4). Several heuristics have been proposed in the literature for solving this problem (e.g. branch-and-bound, preference-order dynamic programming), and item values are typically assumed to be normally-distributed (see Carraway et al. [2] for a more detailed historical account). Previous literature on the stochastic knapsack problem appears to be entirely focussed on the non-adaptive case where a subset of items is determined apriori, which seems to be much more computationally demanding than the corresponding adaptive variant we study in this paper, where items are sequentially chosen for placement in the knapsack based on the accumulated size (or cost) thus far.

4.1 Packing and Covering with Multiple Item Copies Permitted

If multiple copies of items are allowed, this simplifies the state space of DP subproblems, since all that matters is how much room remains in the knapsack and not the set of items that have already been placed in the knapsack. Consider first the knapsack packing problem. Let $V[j]$ denote the optimal expected value one can pack into a capacity-j knapsack, and let $V_i[j]$ denote the optimal expected value where item i is the next to be added. We then have a DP formulation based on the following recurrences:

$$V[j] = \max_{i=1\ldots n} \{V_i[j]\}. \tag{6}$$

$$V_i[j] = v_i \mathbf{Pr}[s_i \leq j] + \sum_{j'=1}^{j} \mathbf{Pr}[s_i = j']V[j - j'] \tag{7}$$

Computation of $V[1 \ldots C]$ by straightforward evaluation of (6) and (7) requires $O(nC^2)$ time, and this is improved to $O(nC \log^2 C)$ time if we employ n interleaved instances of zero-delay convolution to evaluate (7) for each item. The DP formulation for the covering problem variant is quite similar, and the running time improvement achieved using zero-delay convolution is the same. Although (6) and (7) assume that item sizes are positive integers, it is not difficult to write a slightly more complicated set of recurrences (with no impact on running time) that accommodates items with $\mathbf{Pr}[s_i = 0] > 0$. It is worth noting that one can also express the packing variant as stochastic shortest path problem, in the same way that a classical knapsack problem can be expressed as a standard shortest path problem.

4.2 The Stochastic and Dynamic Knapsack Problem

We can extend the DP formulation above to solve a problem variant known as the stochastic and dynamic knapsack problem [14, 12], in which items arrive in an on-line fashion. Returning to the scheduling interpretation of the stochastic knapsack problem, suppose $A_i[t]$ denotes the probability that a copy of task type i arrives when precisely t units of time remain before our deadline. At this point in time if we are currently not processing any task we may choose to accept the arriving task and begin processing it. Otherwise, if we decline the task it is cannot be recalled later (although another task of the same type might arrive later). Papastavrou et al. [14] describe numerous applications of stochastic and dynamic knapsack problems. Note that only packing problems make sense in this framework, and that an optimal adaptive policy might involve "idle" time. To easily accommodate idle time we introduce a dummy task with unit size, zero value, and unit arrival probability. It is therefore guaranteed that at least one task arrives at every point in time, so our goal is to compute a policy that selects the best possible arriving task every time the current task reaches completion. We can write a DP formulation for this problem by replacing (6) with

$$V[j] = \max_{i=1\ldots n} \{V_i[j]A_i[j]\}. \tag{8}$$

Zero-delay convolution provides an $O(nC \log^2 C)$ algorithm that optimally solves this problem. The resulting policy should be employed as follows: every time a task is completed at time t we examine the set S of incoming tasks and select the task maximizing $V_i[t]$ over $i \in S$.

For this problem variant as well as the previous non-dynamic packing and covering variants we can accomodate items with time-varying values/costs at no additional running time expense. We can also extend

the formulations for these variants to allow the final interrupted item to generate some pro-rated amount of value based on the amount of progress on the item prior to its termination.

4.3 The 0/1 Case

A very interesting phenomenon occurs when we consider the 0/1 knapsack problem. In this case we restrict our focus to the packing variant, as the covering variant is not well-posed in the 0/1 case since we could run out of items before successfully covering the knapsack. Whereas in the deterministic case the 0/1 problem is no harder (from a DP running time standpoint) than the "multiple items allowed" variant, in the stochastic case the 0/1 problem seems to be significantly more difficult. The natural DP algorithm for the deterministic 0/1 problem computes an optimal "ordered" solution: it builds a subset of items to include in the knapsack sequentially — to compute the best subset of items $1 \dots i$ to include in the knapsack, we decide whether to insert item i after first optimally scheduling a subset of items $1 \dots i-1$. The items comprising the resulting optimal solution are implicitly ordered by their index in the problem input, but this is of no concern in the deterministic case since the order of the items in the knapsack is of no consequence. However, in the stochastic case the optimal adaptive policy might not involve inserting items in order of their input indices.

It does not seem that there exists a DP formulation for computing the optimal adaptive policy whose state space is polynomial in n and C. However, it turns out that for any ordering of the items, the expected value of the optimal "ordered" policy (constrained to insert items in an order consistent with their indices in the input) is within a constant factor of the expected value of the optimal unordered policy [4]. An optimal ordered adaptive policy can be computed using DP, although it does not require zero-delay techniques (the standard FFT suffices).

4.4 Random Costs/Values and Deterministic Sizes

One can also consider stochastic knapsack problems in which item sizes are deterministic but costs/values are independent random variables with known discrete distributions. The objective in this case is similar to that of the stochastic shortest path problem: we seek an adaptive policy that maximizes the probability of delivering a solution of at least some specified target value (or alternatively, of at most some specified maximum cost). The DP formulation for these problems allows use of the standard FFT, so zero-delay techniques are not necessary. However, even using the FFT we still encounter factors of both V (target value) and C (knapsack capacity) in the running time, so these DP algorithms are rather computationally intensive.

References

[1] ANDREATTA, G., *Shortest path models in stochastic networks*, Advanced School in Stochastics and Combinatorial Optimization, Edited by G. Andreatta, F. Mason, and P. Serafini, CISM Udine, World Scientific Publishing, Singapore, 1987.

[2] CARRAWAY, R.L., SCHMIDT, R.L., and WEATHERFORD, L.R., *An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns,* Naval Research Logistics, Vol. 40, pp. 161-173, 1993.

[3] DEAN, B.C., *Approximation Algorithms for Stochastic Scheduling Problems*. PhD Thesis, Massachusetts Institute of Technology, 2005.

[4] DEAN, B.C., GOEMANS, M.X., and VONDRAK, J., *Approximating the stochastic knapsack problem: the benefit of adaptivity*, Proceedings of the 45th annual IEEE symposium on the Foundations of Computer Science (FOCS), pp. 208-217, 2004.

[5] DERMAN, C., LIEBERMAN,G.J., and ROSS,S.M., *A renewal decision problem*, Management Science, Vol. 24, No. 5, pp. 554-561, 1978.

[6] FRANK, H., *Shortest paths in probabilistic graphs,* Operations Research, Vol. 17, pp. 583-599, 1969.

[7] GALIL, Z. and PARK, K., *Dynamic Programming with Convexity, Concavity and Sparsity*, Theoretical Computer Science Vol. 92, pp. 49-76, 1992.

[8] GARDNER, W.G., *Efficient convolution without input-output delay*, Journal of the Audio Engineering Society, Vol. 43, No. 3, pp. 127-136, 1995.

[9] HALL, R., *The fastest path through a network with random time-dependent travel times*, Transportation Science, Vol. 20, No. 3, pp. 182-188, 1986.

[10] KAMBUROWSKI, J., *A note on the stochastic shortest route problem*, Operations Research, Vol. 33, pp. 696-698, 1985.

[11] KLEINBERG, J., RABINI, Y., and TARDOS, E., *Allocating bandwidth for bursty connections*, proceedings of the 29th annual ACM Symposium on the Theory of Computation (STOC), pp. 664-673, 1997.

[12] KLEYWEGT, A., PAPASTAVROU, J. D., *The dynamic and stochastic knapsack problem with random sized items*, Operations Research, Vol. 49, No. 1, pp. 26-41, 2001.

[13] LOUI, R., *Optimal paths in graphs with stochastic or multidimensional weights*, Communications of the ACM, Vol. 26, No. 9, pp. 670-676, 1983.

[14] PAPASTAVROU, J.D., RAJAGOPALAN, S., KLEYWEGT, A., *The dynamic and stochastic knapsack problem with deadlines*, Management Science, Vol. 42, No. 12, pp. 1706-1718, 1996.

[15] SIGAL, C.E., PRITSKER, A.A.B., and SOLBERG, J.J., *The stochastic shortest route problem,* Operations Research, Vol. 28, pp. 1122-1129, 1980.

[16] STEINBERG, E., and PARKS, M.S., *A preference order dynamic program for a knapsack problem with stochastic rewards*, The Journal of the Operational Research Society, Vol. 30, No. 2, pp. 141-147, 1979.

[17] UETZ, M., *Algorithms for deterministic and stochastic scheduling.* PhD Thesis, Institut für Mathematik, Technische Universität Berlin, 2001.