Electronic Colloquium on Computational Complexity, Report No. 42 (2018)

To appear in: *Operations Research Letters*

Incremental versus Non-Incremental Dynamic Programming☆

Stasys Jukna[1]

Institute of Computer Science, Goethe University Frankfurt, Robert-Mayer-Str. 11–15, 60325 Frankfurt am Main, Germany

Abstract

Many dynamic programming algorithms for discrete optimization problems are *pure* in that they only use min/max and addition operations in their recursions. Some of them, in particular those for various shortest path problems, are even *incremental* in that one of the inputs to the addition operations is a variable. We present an explicit optimization problem such that it can be solved by a pure DP algorithm using a polynomial number of operations, but any incremental DP algorithm for this problem requires a super-polynomial number of operations.

Key words: Dynamic programming, Complexity lower bounds

1. Introduction

A 0-1 optimization problem is specified by giving a set of ground elements, and a family of its subsets, called *feasible solutions*. Given an assignment of nonnegative weights to the ground elements, the goal is to compute the maximum or the minimum weight of a feasible solution, the latter being the sum of weights of the elements in this solution.

Dynamic programming (DP) is a popular technique to solve optimization problems. Many fundamental DP algorithms are *pure* in that they only use min/max and addition operations in their recursion equations. Such an algorithm is *incremental* if one of the two inputs of every addition operation is a variable. In the DP literature, incremental DP algorithms are also called *monadic* while the non-incremental ones are called *polyadic* (see, e.g., [9]).

Notable examples of incremental DP algorithms are the well-known Bellman–Ford DP algorithm for the shortest s-t path problem [1, 3], the Held–Karp DP algorithm for the traveling salesman problem [4], and others. Say, Bellman and Ford consider subproblems $f_l(j)$ = the minimum weight of a path from s to the node j consisting of at most l edges. The basis cases are $f_1(j) = x_{s,j}$ (weights of edges from s to nodes j), and the recursion is

$$f_{l+1}(j) = \min\Big\{f_l(j),\ \min_i\{f_l(i) + x_{i,j}\}\Big\}. \tag{1}$$

☆Research supported by the DFG grant JU 3105/1-1 (German Research Foundation).

Email addresses: stjukna@gmail.com (Stasys Jukna)

[1]Affiliated with Institute of Data Science and Digital Technologies, Faculty of Mathematics and Informatics, Vilnius University, Lithuania.

ISSN 1433-8092

A prominent example of a non-incremental pure DP algorithm is that of Floyd [2] and Warshall [10] for the all-pairs shortest paths problem: it uses the recursion

$$f_k(i,j) = \min\{f_{k-1}(i,j), f_{k-1}(i,k) + f_{k-1}(k,j)\}\ ,$$

where $f_k(i,j)$ stands for the length of a shortest path from i to j that only uses nodes $1,\dots,k$ as inner nodes. Hence, a natural question arises:

- Can non-incremental pure DP algorithms be substantially faster than incremental DP algorithms?

In this note, we use a recent construction of Hrubes and Yehudayoff [5] to answer this question *affirmatively*: there are explicit optimization problems that can be solved by pure DP algorithms using a polynomial number of operations, but any incremental DP has to apply a super-polynomial number of operations to solve them.

To define the corresponding optimization (minimization and maximization) problems, consider the complete binary tree $T_d = (V,E)$ of depth d; hence, T_d has $|V| = n := 2^{d+1} - 1$ nodes. A *labeling* of T_d is a mapping $h : V \to \mathbb{Z}_n$ from the nodes of T_d to the additive group $\mathbb{Z}_n = \{0,1,\dots,n-1\}$ of integers modulo n. A labeling h is *legal* if it is additive in the following sense: if u is a node with sons v and w in T_d, then $h(u) = h(v) + h(w)$. Let H denote the set of all legal labelings of T_d. Note that the labels of leaves in every legal labeling determine the labels of all other nodes. So, there are $|H| = n^{2^d} = n^{(n+1)/2}$ legal labelings.

The optimization (minimization or maximization) problem on H is: given an assignment of nonnegative weights $x_{u,i}$ to the points (u,i) of the grid $V \times \mathbb{Z}_n$, compute the minimum or the maximum weight $\sum_{u\in V} x_{u,h(u)}$ of a legal labeling h. Note that the number of ground elements in this case is $|V \times \mathbb{Z}_n| = n^2$.

Theorem. *Both optimization problems on H can be solved by pure DP algorithms using $O(n^3)$ operations, but any incremental DP algorithm for either of these two problems must perform at least $n^{\Omega(\log n)}$ operations.*

The lower bound here holds even when incremental DP algorithms are only required to solve the problem correctly on all input weightings $x \in \{0,1\}^n$ (in the case of maximization) or on all input weightings $x \in \{0,1,n+1\}^n$ (in the case of minimization). Moreover, the lower bound is already on the number of *unbounded* fanin min/max operations.

The upper bound is given by Lemma 1, and the lower bound by Lemma 7.

2. Upper bound

The upper bound $O(n^3)$ was proved by Hrubes and Yehudayoff [5] for monotone arithmetic circuits computing the corresponding (to H) multivariate polynomial, but their argument gives, with almost no changes, also a pure DP algorithm.

Lemma 1. *Each of the two optimization problems on H can be solved by a pure, non-incremental DP algorithm using $O(n^3)$ operations.*

Proof. We only consider the minimization problem on H; the case of the maximization problem is the same. So, our goal is to efficiently solve, by a pure DP algorithm, the following minimization problem:

$$f_n(x) = \min_{h\in H} \sum_{u\in V} x_{u,h(u)}\,.$$

For $i \in \mathbb{Z}_n$, let $F^{(d)}_{v,i}$ denote the minimization problem defined as f_n, except that h is now restricted to range over legal labelings with $h(v) = i$, where v is the root of T_d. Hence, $f_n = \min_{i \in \mathbb{Z}_n} F^{(d)}_{v,i}$. Observe that

$$F^{(d)}_{v,i} = \min\left\{F^{(d-1)}_{u,j} + F^{(d-1)}_{w,k} + x_{v,i} \colon j,k \in \mathbb{Z}_n, j+k=i\right\} ,$$

where u and w are the left and right sons of the root v. Hence, if $L(d)$ denotes the number of operations required to compute the minimum of all $F^{(d)}_{v,i}$ with $i \in \mathbb{Z}_n$, then we have a recursion $L(d) \leq 2 \cdot L(d-1) + O(n^2)$, which resolves to $L(d) = O(n^2 2^d) = O(n^3)$. So, the resulting pure DP algorithm uses only $O(n^3)$ operations to compute f_n, as desired. $\square$

Our goal is now to show that every incremental DP algorithm solving any of the two optimization problems on H must use at least $n^{\Omega(\log n)}$ operations. After some preparations, this will be done in Lemma 7.

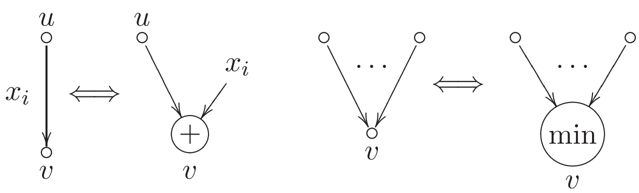
3. Tropical branching programs

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of nonnegative integers. Every finite set $A \subset \mathbb{N}^n$ of vectors (of feasible solutions) *defines* two natural optimization problems: given an assignment $x \in \mathbb{R}^n_+$ of nonnegative weights, compute the minimum or the maximum weight $\langle a, x\rangle = a_1 x_1 + \cdots + a_n x_n$ of a feasible solution $a \in A$.

A natural mathematical model for incremental DP algorithms is that of *tropical branching programs* (tropical BP). Such a program is a directed acyclic graph G with one zero indegree node s (the source node) and one zero outdegree node t (the target node); multiple edges joining the same pair of nodes are allowed. Every edge is either unlabeled or is labeled by one of the variables $x_1, \ldots, x_n$. The *size* of a BP is the total number of its nodes.

Every tropical BP *solves* some optimization problem (minimization or maximization) in a natural way: along every s-t path, the sum of all labels of its edges is computed and the optimization problem solved by the program is the minimum or the maximum of all these sums.

Another way to view at how a tropical BP computes, and to see the connection with incremental DP algorithms, is to view each *edge* as adding its label x_i (or constant 0, if the edge is unlabeled) to the already computed value, and to view each *node* (except for the source node s) as performing a min or max operation:



Every tropical BP also *creates* some set $B \subset \mathbb{N}^n$ of vectors in a natural way. Namely, every path p *creates* the vector $b = (b_1, \ldots, b_n)$, where $b_i \in \mathbb{N}$ is the number of times the i-th variable x_i appears as a label of an edge of p. In particular, if the path p has no labeled edges, then the created vector is the all-0 vector (of length n). The set of vectors created at a node $u \neq s$ is the set of vectors created by paths from s to u. The set of vectors created by the entire branching program is the set of vectors created at the target node t.

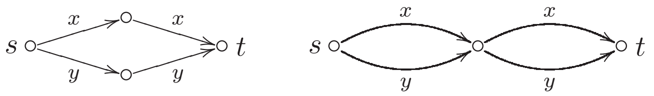


Figure 1: Two BPs solving the same minimization problem $f(x, y) = \min\{2x, 2y\}$ given by the set $A = \{(2, 0), (0, 2)\}$ of feasible solutions. The set of vectors created by the first BP is $B = A$, but the second BP saves one node by creating a *different* set $B = \{(2, 0), (1, 1), (0, 2)\}$. Both sets A and B define the same minimization problem, because $A \subseteq B$ and vector $(1, 1)$ is a convex combination $\frac{1}{2}(2, 0) + \frac{1}{2}(0, 2)$ of vectors in A.

If $b \in B$ is the vector created by an s-t path, then on every input weighting $x \in \mathbb{R}^n_+$, the sum $\langle b, x\rangle = b_1x_1 + \cdots + b_nx_n$ is computed along this path. Hence, the BP solves an optimization problem defined by B, i.e., computes either the minimum or the maximum of $\langle b, x\rangle$ over all vectors $b \in B$.

If, however, we only know that the BP G *solves* a given optimization problem defined by some set $A \subset \mathbb{N}^n$ of feasible solutions, then the set of vectors B created by G does not need to *coincide* with A (see Fig. 1): we only know that the set B must define the same optimization problem as A. And indeed, there *are* sets A of feasible solutions such that it is exponentially easier to *solve* an optimization problem on A than to *create* the set A itself.

Example. Consider the shortest s-t paths problem. Here we have $m = \binom{n}{2}$ ground-elements (edges of a complete graph K_n on n vertices), and $A \subset \{0, 1\}^m$ is the set of characteristic vectors of s-t paths in K_n. The minimization problem on A can be solved by a tropical BP corresponding to the Bellman–Ford shortest s-t path dynamic programming algorithm (1) using only $O(n^2)$ nodes and $O(n^3)$ edges. The set $B \subset \mathbb{N}^m$ of vectors created by this program corresponds not to *paths* but rather to *walks* from node s to node t of length at most $n - 1$; in particular, some vectors of B have entries larger than 1. Still, the BP *solves* the shortest s-t path problem correctly, because every s-t path is also an s-t walk, and every s-t walk contains an s-t path (recall that we only consider nonnegative weights). On the other hand, the results of Jerrum and Snir [6] imply that tropical BPs with $2^{\Omega(n)}$ nodes are necessary to *create* the set A itself. □

Fortunately, if a set A is homogeneous, then it is no more difficult to create A than to solve an optimization problem on A. A set of vectors is *homogeneous* if all its vectors have the same degree, where the *degree* of a vector is just the sum of its entries.

Lemma 2. *Let $A \subset \{0, 1\}^n$ be homogeneous. If an optimization problem on A can be solved by a tropical BP of size ℓ, then the set A can be also created by a BP of size at most ℓ.*

To prove this lemma, we will need some known structural properties of the sets of feasible solutions of equivalent optimization problems.

If two sets of vectors define the same optimization problem, how are these two sets then related? To answer this question, say that a set $B \subset \mathbb{N}^n$ lies *above* (resp., *below*) a set $A \subset \mathbb{N}^n$ if for every vector $b \in B$ there is at least one vector $a \in A$ such that $b_i \geq a_i$ (resp., $b_i \leq a_i$) holds for all $i = 1, \ldots, n$. A set A is an *antichain* if no two of its vectors are comparable under $\leq$.

Lemma 3 ([7, 8])**.** *Let $A \subset \{0, 1\}^n$ be an antichain, and $B \subset \mathbb{N}^n$ be some finite set of vectors. Then:*

(i) $\max_{b\in B} \langle b,x\rangle = \max_{a\in A} \langle a,x\rangle$ *holds for all weightings* x *in* $\{0,1\}^n$ *if and only if* $A \subseteq B$ *and* B *lies below* A.

(ii) $\min_{b\in B} \langle b,x\rangle = \min_{a\in A} \langle a,x\rangle$ *holds for all weightings* x *in* $\{0,1,n+1\}^n$ *if and only if* $A \subseteq B$ *and* B *lies above* A.

A simple proof of this lemma is given in [7, Lemmas 6,7]; in the proof of item (ii), the weight $+\infty$ was used but, as shown in [8, Appendix A], one can use the weight $n+1$ instead. The "if" directions here are trivial, and hold even for all nonnegative weightings $x \in \mathbb{R}^n_+$. What is interesting is the "only if" direction: relatively small sets of weightings x are sufficient to capture the *structural* relation between sets A and B. Note that the assumption that A must consist of only 0-1 vectors is here important (cf. Fig. 1).

Proof of Lemma 2. Let $A \subset \{0,1\}^n$ be homogeneous of degree d, and consider first the *maximization* problem on A. Let G be tropical $(\max,+)$ branching program solving this problem, and let $B \subset \mathbb{N}^n$ be the set of vectors created by G. Our goal is to show that some subprogram of G must create the set A.

By Lemma 3, we know that $A \subseteq B$, and every vector of B must be contained in at least one vector of A. These two properties imply that A is the subset of all vectors in B of *largest* degree (which is the degree d of A).

Define the *length* of a path in G to be the number of labeled edges along this path. Hence, the length of a path coincides with the degree of the vector created by this path. Apply iteratively the following transformation to the BP G until possible: remove an edge (u,v) if the longest path from the source node s to node v going through this edge is shorter than the longest path from s to v.

Since vectors in $A \subseteq B$ have the largest degree among all vectors in the set B created by the program G, none of the vectors of A will be lost during this removal. By applying such a removal as long as possible, we will eventually obtain the desired BP creating the set A.

The argument in the case of the *minimization* problem on A is almost the same. By Lemma 3, the set A is then exactly the subset of vectors in B of *smallest* degree (which is the degree d of A). Thus, we now remove an edge (u,v) if the shortest path from s to v going through this edge is longer than the shortest path from s to v. ☐

The *sumset* $X+Y$ of two sets of vectors $X,Y \subset \mathbb{N}^n$ is the set of all vectors $x+y$ with $x \in X$ and $y \in Y$. Such a sumset is *k-balanced* if all vectors of X have the same degree k.

Lemma 4. *Let $A \subset \{0,1\}^n$ be homogeneous of degree d, and suppose that an optimization problem on A can be solved by a tropical branching program of size ℓ. Then for every $k = 1,\ldots,d$, the set A can be written as a union of at most ℓ k-balanced sumsets.*

Proof. Lemma 2 gives us a tropical BP G of size at most ℓ which not only solves the corresponding optimization problem on the set A (maximization or minimization), but even *creates* this set.

Associate with every node u of G the sumset $X_u + Y_u$, where X_u is the set of vectors created by paths from the source node s to u, and Y_u is the set of vectors created by paths from node u to the target node t. Then $X_u + Y_u$ is exactly the set of all vectors created by s-t paths going through node u. Since no vector outside A can be created, we have $X_u + Y_u \subseteq A$. Moreover, since A is homogeneous, all vectors in X_u must have the *same* degree d_u. Along the nodes u of every s-t path p, d_u takes all values $0,1,\ldots,d$: if $e = (u,v)$ is an edge of p, then $d_v = d_u$ if e is unlabeled, and $d_v = d_u + 1$ otherwise. So, for every $k = 1, \ldots, d$, the union of k-balanced sumsets $X_u + Y_u$ over all nodes u with $d_u = k$ gives us the desired covering of A. $\square$

4. Lower bound

Recall that a labeling $h : V \to \mathbb{Z}_n$ of the nodes of a complete binary tree T_d of depth d is *legal* if for every non-leaf u, $h(u)$ is the sum (modulo n) of the labels of all leafs of the sub-tree rooted in u. Let H be the set of all legal labelings. The following lemma from [5] gives the key combinatorial property of this set.

Let $V = V_0 \cup V_1$ be a partition of V. The composition of two partial labelings $h_0 : V_0 \to \mathbb{Z}_n$ and $h_1 : V_1 \to \mathbb{Z}_n$ is the labeling $h = h_0 \circ h_1$ which coincides with h_i on all nodes in V_i, $i = 0, 1$. Every two sets $H_0 \subseteq \mathbb{Z}_n^{V_0}$ and $H_1 \subseteq \mathbb{Z}_n^{V_1}$ of partial labelings define the *rectangle*

$$H_0 \circ H_1 = \{h_0 \circ h_1 \colon h_0 \in H_0 \text{ and } h_1 \in H_1\}$$

consisting of all possible compositions of these labelings. Such a rectangle is a *k-rectangle* if $|V_0| = k$.

The complete binary tree T_d has $n = 2^{d+1} - 1$ nodes. For an integer $0 \leq k \leq n$, take its binary expansion $a = (a_0, a_1, a_2, \ldots)$ with $a_i \in \{0, 1\}$, that is, $k = a_0 + 2a_1 + 4a_2 + 8a_3 + \cdots$, and let $\tau(k)$ denote the number of configurations "10" (one followed by zero) in the vector a. For example, $\tau(5) = \tau(1 + 4) = 2$ whereas $\tau(7) = \tau(1 + 2 + 4) = 1$. In particular, if we take $k = 1 + 4 + 16 + \cdots + 2^d = (2n + 1)/3$ (for even d) or $k = 1 + 4 + 16 + \cdots + 2^{d-1} = n/3$ (for odd d), then $\tau(k) \geq d/2$.

Lemma 5 (Hrubes and Yehudayoff [5]). *Let $1 \leq k < n$ be an integer. If a k-rectangle $H_0 \circ H_1$ consists of only legal labelings, that is, if $H_0 \circ H_1 \subseteq H$ holds, then*

$$|H_0 \circ H_1| \leq \frac{|H|}{n^{\tau(k)/8}}\,.$$

The *characteristic vector* of a labeling $h : V \to \mathbb{Z}_n$ of T_d is a vector in $\{0, 1\}^{n \times n}$ whose (u, i)-th position is 1 if and only if $h(u) = i$. Consider the set $A \subset \{0, 1\}^{n \times n}$ of characteristic vectors of all legal labelings of T_d. Note that this set A *is* homogeneous: every its vector has exactly $|V| = n$ ones.

If $x \leq a$ for some $a \in A$, then we say that vector x *touches* a node u of the tree T_d, if $x_{u,i} = 1$ holds for some $i \in \mathbb{Z}_n$. Note that every vector $a \in A$ touches every node in V (exactly once).

Lemma 6. *If $\emptyset \neq X + Y \subseteq A$, then all vectors in X as well as all vectors in Y touch the same sets of nodes, and these two sets of touched nodes form a partition of the set of all nodes of the tree.*

Proof. Assume contrariwise that there are two vectors $x \neq x' \in X$ and a node u in the tree T_d which is touched by x but not by x'. Take any vector $y \in Y$. Since u is not touched by x', and since $x' + y$ must be a characteristic vector of some legal labeling, the node u must be touched by vector y. Hence, there are $i, j \in \mathbb{Z}_n$ such that $x_{u,i} = y_{u,j} = 1$. Since the vector $x + y$ belongs to A, it must be a 0-1 vector. So, $i \neq j$. But this is impossible because no function can take two distinct values i and j on the same input u. $\square$

Lemma 7. *None of the two optimization problems on A can be solved by a tropical branching program having fewer than $n^{\Omega(\log n)}$ nodes.*

Proof. Let t be the minimum number of nodes in a tropical branching program solving either the maximization or the minimization problem on A, and let k be a natural number for which $\tau(k) \geq d/2$ holds (we know that such k exists, be the depth d even or odd). Since the set A is homogeneous, Lemma 4 implies that it can be written as a union of at most t k-balanced sumsets $X + Y \subseteq A$; being k-balanced implies that every vector of X has exactly k ones.

Lemma 6 implies that every vector in X must touch the same set V_0 of nodes, and every vector in Y must touch the same set V_1 of the remaining nodes of the tree T_d. Moreover, we have $|V_0| = k$ since every vector of X has exactly k ones. By Lemma 5, we have that $|X+Y| \leq |A|/n^{d/16}$. Thus, the number t of nodes in our BP must be at least $|A|/(|A|/n^{d/16}) = n^{d/16} = n^{\Omega(\log n)}$, as claimed. □

The lower bound in Lemma 7 is not far from the truth: $n^{O(\log n)}$ nodes are also sufficient. This follows from the construction of a DP algorithm in Lemma 1. It actually gives us a tropical *formula* with $n^{O(d)}$ gates, and tropical formulas are tropical branching programs of a special form, when the underlying graph is series-parallel.

References

[1] Bellman, R., 1958. On a routing problem. Quarterly of Appl. Math. 16, 87–90.
[2] Floyd, R., 1962. Algorithm 97, shortest path. Comm. ACM 5, 345.
[3] Ford, L., 1956. Network flow theory. Tech. Rep. P-923, The Rand Corp.
[4] Held, M., Karp, R., 1962. A dynamic programming approach to sequencing problems. SIAM J. on Appl. Math. 10, 196–210.
[5] Hrubes, P., Yehudayoff, A., 2016. On isoperimetric profiles and computational complexity. In: Proc. of 43rd ICALP (2016). Vol. 55 of Leibniz Int. Proc. in Informatics. pp. 89:1–89:12.
[6] Jerrum, M., Snir, M., 1982. Some exact complexity results for straight-line computations over semirings. J. ACM 29 (3), 874–897.
[7] Jukna, S., 2015. Lower bounds for tropical circuits and dynamic programs. Theory of Comput. Syst. 57 (1), 160–194.
[8] Jukna, S., 2016. Tropical complexity, Sidon sets and dynamic programming. SIAM J. on Discrete Math. 30 (4), 2064–2085.
[9] Kumar, V., Grama, A., Gupta, A., Karypis, G., 1994. Introduction to Parallel Computing. Benjamin–Cummings.
[10] Warshall, S., 1962. A theorem on boolean matrices. J. ACM 9, 11–12.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il