Incremental Topological Sorting in Phonological Learning[*]

Cao Yu

1 Introduction

In Optimality Theory (OT; Prince and Smolensky 2004), the grammar of a language is characterized as a (strict) partial order on a set of constraints on input-output (IO) maps that are universal to all languages. One of the major tasks of learning thus lies in determining that partial order, or *ranking*.

A linearly ordered set of constraints $c_1 < \ldots < c_n$ sends an IO map f to a *violation vector* $(v_1, \ldots, v_n)$ (Prince 2002), where v_i counts the violation of f with respect to c_i. Given the lexicographic ordering, a legitimate constraint ranking should ensure that for any given input, the actual IO map be assigned the least violation vector. In this sense, the actual IO map is *optimal*, a *winner*. Thus whenever a winner map for some input is *known*, it informs us about the constraint ranking with an instruction as such: for any alternative map of the same input, which is a *loser*, its violation vector $(u_1, \ldots, u_n)$ is greater than that $(v_1, \ldots, v_n)$ of the winner. Prince shows that this is equivalent to saying that in the coordinate-wise difference $(\ldots, d_i = v_i - u_i, \ldots)$, each positive d_i (loser-favoring) is preceded by some negative d_j (winner-favoring). If we follow Prince, writing `L` for positive coordinates, `W` for negative ones, and `e` for zeros, we obtain an *elementary ranking condition* (ERC; see Tesar 2014, sec. 3.1 for illustrations). To repeat, an ERC informs the constraint ranking with the instruction that each `L`-valued constraint is preceded, or *dominated*, by some `W`-valued constraint.

A large part of phonological learning concerns itself with constructing winner-looser pairs that provide informative ERCs. Strictly speaking, once all informative ERCs are found, the learning proper is done. But to see the ranking instructions given by the ERCs are consistent—the corresponding directed graph is acyclic (DAG; see Cormen et al. 2009), and to be able to use it in practice, we still need to extend it into a linear order, or at least a *stratified* linear order, i.e. a linear order on the partitions, called *strata* (Tesar 2014, sec. 5.3), of the constraint set. From Zorn's Lemma it follows that such an extension always exists (see Shen and Vereshchagin 2000), but what we are interested in is finding one.

The problem was first solved by Tesar (1995) with his Recursive Constraint Demotion (RCD), an algorithm that deals specifically with ERCs. The goal of this paper, however, is to relocate the problem in a more general context: extending a partial order into a linear order, a classic graph theoretic problem known as *topological sorting* (Cormen et al. 2009; Knuth 1997). The motivation is twofold:

- since topological sorting has wide application in various network settings, viewing the problem in its abstract form allows us to see its computational essence at a closer distance, and

[*]I thank Prof. Bruce Tesar for encouraging me to embark on this project, and Prof. Alexander Shen for referring me to online topological sorting, and Prof. David J. Pearce for sharing his thoughts on the issue under investigation.

hopefully its connection with other real-world problems that feature the same computational essence;

- more importantly, recent developments in *online topological sorting* (Haeupler et al. 2008; Pearce and Kelly 2004 a.o.) have promising results in efficiently maintaining a linear order as partial ordering information accumulates over time. Since phonological learning usually uses the existing ranking information to discover more informative ERCs (e.g. Error-driven Learning; Tesar 2014), conceptualizing the problem in the general context of topological sorting enables us to explore how and how well the standard online results may address the dynamics of phonological learning.

The organization of the paper is as follows. Section 2 and 3 discuss in turn the static and online, or more precisely, incremental topological sorting that receive the partial ordering instructions given by ERCs. For the static case, a trivial modification of Kahn's (1962) algorithm well serves the purpose. We will see the modified Kahn's (MK) algorithm is essentially the same as RCD. For the incremental case, the idea of *selective updating* is incorporated into MK. In Section 4, the efficiency of the online MK is measured against experimental data. Different selective and trivial updating strategies are compared. Generally speaking, compared to trivial updating (e.g. MRCD; Tesar 2014), a fine-tuned online MK fares better when the input ERCs are structured in such a way that the ordering instruction of each ERC is not entailed by those of the previous ERCs. Section 5 concludes.

2 Disjunctive Topological Sorting

Standard (static) topological sorting takes as its input a set E of *directed edges* $(u_1, v_1), (u_2, v_2)...$, where u_i, v_i belong to the object set V. An edge (u_k, v_k) gives the ordering instruction that u_k precedes v_k. Kahn's (1962) algorithm finds a linear order on V which is compatible with the ordering instructions of all edges in E within a linear time complexity (Knuth 1997, sec. 2.2.3). This is how it works.

Kahn's algorithm We start by looking for an object x in V that is not preceded by anything (i.e. some x that never appears on the right hand side of any edge). x can be taken to be the first in the order. We then remove x from V, and from E all edges in which x appears as the first coordinate. We can do so because the removed edges all give instructions of the form "x precedes y". The reduced V and E again present a partial order for which a linear order is to be found. We therefore repeat the procedure until all the objects of V have been sorted. Whenever each object in V is preceded by something, we detect a cycle among the directed graph given by the original E and V. In the latter case, a linear order does not exist.

Turning to ERCs, we notice that the ordering instructions they give are slightly different: each ERC instructs that each `L`-valued constraint is preceded by *some* `W`-valued constraint, but there can be multiple `W` in a single ERC. Thus for a set of ERCs, we can translate their instructions into a set E of edges $(U_1, v_1), (U_2, v_2), ...$, where v_i belongs to V (the constraint set) and U_i is a nonempty subset of V. An edge (U_k, v_k) gives the instruction that at least one $u \in U_k$ precedes v_k:

$$\bigvee_{u \in U_k} u < v_k.$$

Namely, now our input edges each encodes a disjunction.

For lack of a better name, we may call topological sorting of the partial order given by disjunctive edges *disjunctive topological sorting*. Kahn's algorithm can be easily modified to handle such cases (Alexander Shen p.c.).

Modified Kahn's (MK) algorithm For a given set V of objects and a set E of disjunctive edges, we still look for an object x that is not preceded by anything and rank it as the first in the order. We remove x from V as before, but from E we now remove all the edges whose first coordinate contains x. We can do so because the removed edges all give instructions of the form "x or something else precede y". We then repeat the procedure until all the objects of V have been sorted. As before, the nonexistence of such an unpreceded x implies an unsolvable cycle. Furthermore, if each time instead of a single x, we look for the set X of all unpreceded objects, rank X as a whole, and remove all $x \in X$ from V, and from E all edges whose first coordinates contain some $x \in X$, we will end up with a stratified linear order, in the sense described on page 1.

In MK, it takes $O(|V|)$ to scan for the unpreceded objects at the beginning, after which we can keep track of unpreceded objects upon edge deletion and pop an unpreceded object in $O(1)$. Also, if for each object we restore the edges whose first coordinates contain that object, we can find the edges to be deleted in $O(1)$. But for a disjunctive edge (U, v) there will be $|U|$ attempts of deletion (Adayah p.c.), thus the overall time complexity of MK would be

$$O(|V| + \sum_{(U,v)\in E} |U|),$$

which is commensurable with $|V|$ times the number of ERCs.

```
function sort($V$, $E$)
    $P \leftarrow \{v \in V | (U, v) \in E\}$                // the set of preceded objects
    $Q \leftarrow V \backslash P$                           // the set of unpreceded objects
    linear ← empty list
    while $|Q| > 0$ do
        append Q to linear
        $E \leftarrow \{(U, v) \in E | U \cap Q = \varnothing\}$          // remove the relevant edges
        $P \leftarrow \{v \in V | (U, v) \in E\}$     // update the set of preceded objects
        $V \leftarrow V \backslash Q$                            // remove the relevant objects
        $Q \leftarrow V \backslash P$                 // update the set of unpreceded objects
    end
    if $|P| = 0$ then return linear else return error message
```

Algorithm 1: MK, generating a stratified linear order for a set V of objects and a set E of disjunctive edges.

Comparison with RCD One may refer to Tesar 1995, 2014 for a detailed presentation of RCD. For our purpose, it suffices to notice that an unpreceded object/constraint corresponds to a constraint that prefers no losers. So it becomes clear that MK is essentially the same as RCD. Indeed, for a given set of ERCs, the stratified version of MK generates the same stratified linear order as RCD does. The two algorithms also share the same time complexity.

3 Incremental Topological Sorting

The static topological sorting discussed in the last section takes a fixed set E of edges (and a set V of objects, which is usually not given but implied) as its input and generates a linear order compatible with the partial ordering instructions given by E. In many real-world applications (e.g. phonological learning), however, edges are not given once and for all but added over time. Gaining more attention in recent years, research on incremental topological sorting aims to maximize the efficiency of linear order maintenance by minimizing the update one has to perform.

A trivial solution is to run topological sorting from the scratch upon each edge addition. Doing so does not take any advantage of the computation efforts we have already paid; we blindly update the whole existing linear order. A boost in efficiency arises once we are more selective about what to update. Indeed, the insight of *selective updating* has been explored by almost every incremental (standard) topological sorting algorithm on the market (see Haeupler et al. 2008 for a review). The following presentation is taken from Pearce and Kelly 2004, an influential representative of this vein of study.

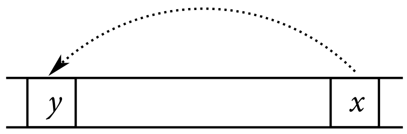

Figure 1: The affected region $\mathsf{AR}_{x,y}$.

Selective updating Let V be a set of objects and E a set of edges. Let $<$ be a linear order on V which is compatible with the partial order given by E. A new edge $(x, y) \notin E$ is said to *invalidate* $<$ if we have y precedes x in $<$, i.e. $y < x$. An invalidating edge (x, y) induces an *affected region*

$$\mathsf{AR}_{x,y} \overset{\text{def}}{=} \{k \in V | y \leq k \leq x\}. \tag{1}$$

Of particular interest are two subsets of $\mathsf{AR}_{x,y}$: R_f (*forward*), consisting of y and all the objects that can be reached from y, given E_t, the transitive closure of E,

$$\mathsf{R}_f \overset{\text{def}}{=} \left\{k \in \mathsf{AR}_{x,y} | k = y \vee (y, k) \in E_t\right\},$$

and R_b (*backward*), consisting of x and all the objects from which x can be reached, given E_t,

$$\mathsf{R}_b \overset{\text{def}}{=} \left\{k \in \mathsf{AR}_{x,y} | k = x \vee (k, x) \in E_t\right\}.$$

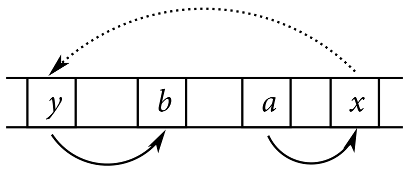

Figure 2: R_f and R_b.

Now we consider the partial order given by the updated edge set $E' = E \cup \{(x, y)\}$. It can be shown that $\mathsf{R}_f \cup \mathsf{R}_b$ covers all the objects misplaced by $<$, or more precisely,

$$(a, b) \in E'_t \wedge b < a \Rightarrow b \in \mathsf{R}_f \wedge a \in \mathsf{R}_b. \tag{2}$$

Indeed, consider the chain $a = t_0, t_1, ..., t_{n-1}, t_n = b$, where $(t_i, t_{i+1}) \in E'$. Since $b < a$, $(a, b) \notin E_t$, which means the chain cannot be made up exclusively of edges in E; it must contain (x, y). Thus either $(a, b) = (x, y)$, or $(y, b), (a, x) \in E_t$ and thus $y < b < a < x$.

Therefore, in order to repair the existing linear order invalidated by (x, y), only part of the affected region $\mathsf{AR}_{x,y}$ needs reorganization. Different incremental topological sorting algorithms vary along the techniques for identifying the subarea of the affected region to be reorganized, and the techniques for updating the identified subarea.

Turning to disjunctive topological sorting, we unfortunately find that when a new disjunctive edge (U, v) invalidates an existing linear order $<$ compatible with an edge set E (i.e. $v < u$ for every $u \in U$; discussion below abstains from stratifies linear orders for sake of simplicity), it is impossible to identify a fixed affected region like (1): we now have a set of potential affected regions

$$\{\mathsf{AR}_{u,v} | u \in U\} \tag{3}$$

to choose from, but we do not know which.

Even worse, it is not guaranteed that we can reorganize any $\mathsf{AR}_{u,v}$ from (3) with the edge (u, v) added to repair $<$. For an arbitrary $u \in U$, consider an analogy of R_b, a backward chain $w_0 = u, w_1, w_2, ...$, where $w_i \in \mathsf{AR}_{u,v}$ and w_{i+1} is the only object that satisfies some ordering requirement of w_i, i.e.

$$X \cap \{x \in V | x < w_i\} = \{w_{i+1}\} \text{ for some } (X, w_i) \in E.$$

Consider also an analogy of R_f, a forward chain $z_0 = v, z_1, z_2, ...$, where $z_i \in \mathsf{AR}_{u,v}$, z_i is the only object that satisfies some ordering requirement of z_{i+1}, and no other objects preceded by z_{i+1} in $\mathsf{AR}_{u,v}$ can potentially satisfy that requirement, i.e.

$$\left.\begin{array}{l} X \cap \{x \in V | x < z_{i+1}\} = \{z_i\} \text{ and} \\ X \cap \{x \in \mathsf{AR}_{u,v} | z_{i+1} < x\} = \varnothing \end{array}\right\} \text{ for some } (X, z_{i+1}) \in E.$$

Whenever a backward chain *meets* a forward chain, in the sense that an object appears in both chains, (u, v) witnesses a cycle in $\mathsf{AR}_{u,v}$.

The solution may well be found in an affected region whose right boundary lies in between two objects of U or beyond the final object of U (referring to the order $<$), and we do not know which disjunct of (U, v) works for that region. Therefore to incorporate the idea of selective updating into disjunctive topological sorting, we can try out affected regions whose right boundaries range from the initial object of U to the final object of $<$, and stick to the disjunctive edge (U, v).

Incremental MK After adding an invalidating edge (U, v) into E, we start with the affected region $\mathsf{AR}_{u,v}$, where u is the initial object in U (i.e. $u < u'$ for all $u' \in U$ other than u). To update the order internal to $\mathsf{AR}_{u,v}$, we set the local edge set as follows:

$$E_{\mathsf{AR}} = \{(\mathsf{AR}_{u,v} \cap X, x) | x \in \mathsf{AR}_{u,v} \wedge (X, x) \in E \wedge X \cap \{y \in V | y < v\} = \varnothing\}. \tag{4}$$

That is, for each object $x \in \mathsf{AR}_{u,v}$ we examine all its ordering requirements, i.e. edges in E that have x as its second coordinate. If an edge (X, x) is taken care of by an object that precedes v (and thus the whole $\mathsf{AR}_{u,v}$), it will be satisfied no matter how we reorganize $\mathsf{AR}_{u,v}$; it is then excluded from E_{AR}. Otherwise, some objects in $\mathsf{AR}_{u,v}$ must satisfy (X, x), since the existing order $<$ is compatible with E before edge addition. We then take intersection $\mathsf{AR}_{u,v} \cap X$, which includes both objects that actually satisfy (X, x) and those which can potentially satisfy (X, x) but is currently ordered after x. The edge $(\mathsf{AR}_{u,v} \cap X, x)$ is then added to E_{AR}. Constructing E_{AR} in such a way ensures that we can repair the order $<$ by reorganizing $\mathsf{AR}_{u,v}$ to the satisfaction of the ordering instructions of E_{AR}.

We therefore run MK over $\mathsf{AR}_{u,v}$ and E_{AR}. If a linear order can be found, we replace the relevant part of $<$ with the newly found order and we are done. But if a cycle is detected, we move on to the object u' next to u, and try to update $\mathsf{AR}_{u',v}$ as above. We repeat the process until $<$ is successfully updated, or the final object of $<$ is tried and fails. In the latter case we conclude that (U, v) is incompatible to the original E.

```
function update($V$, $E$, $<$, $(U, v)$)
    if $<$ is incompatible with $(U, v)$ then
        $u \leftarrow$ the initial object of $U$
        flag $\leftarrow$ false
        repeat
            set $E_{\mathsf{AR}}$ as in (4)
            sub $\leftarrow$ sort($\mathsf{AR}_{u,v}$, $E_{\mathsf{AR}}$)          // MK, non-stratified version
            if no error in sub then
                update $<$ with sub
                flag $\leftarrow$ true
            else $u \leftarrow$ the $<$-successor of $u$
        until flag or $u$ is undefined
            // the successor of the final object of < is undefined
        if flag then return $<$ else return error message
    else return $<$                                        // no update needed
```

Algorithm 2: Incremental MK: updating $<$ when adding (U, v) to the edge set E.

Selective updating is incorporated into the incremental algorithm above in a happy-go-lucky fashion. When a solution is found in an affected region whose right boundary is near the initial object of U, the first coordinate of the invalidating edge, Incremental MK can update the existing linear order quite efficiently. This is because we have a reduced object set (i.e. the affected region) and a reduced edge set for topological sorting. But when a solution is found as the right boundary of the affected region on trial almost reaches the end of $<$, it is far from clear whether the reduction in the sizes of the object set and the edge set still makes a difference, not to mention the computation costs wasted over the failed trials. One might wonder in practice, how frequently the solution would fall around the middle ground. Therefore, while a worst-case time bound is hard to measure, it seems safe to conclude that the merit of incremental MK should lie in its amortized complexity (per edge addition). This is to be tested against experimental data in the next section.

4 Experimental Study

This section reports an experimental comparison of the efficiencies of different selective and trivial order updating strategies in incremental topological sorting.

Strategies to compare We may identify six different order updating strategies in the context of phonological learning. First, we distinguish between strategies which process an ERC as a whole and those which break an ERC into separate edge additions. We then distinguish between trivial (unselective) updating and selective updating. Finally, since failed trials in searching for a minimal affected region which provides a solution is the major cost of the incremental algorithm laid out in Algorithm 2, we may test whether trading off the size of the affected region against the number of trials pays off. Thus when updating selectively, we distinguish between the strategies which push the

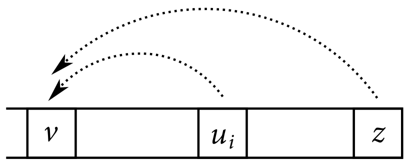

Figure 3: For an invalidating edge (U, v), restrict AR trials to those with a right boundary $u_i \in U$, and $\mathsf{AR}_{z,v}$, where z is final object the current order <.

right boundary of the affected region one object at a time, and those which shift the right boundary from one precedence disjunct to the next, i.e. among the objects in the first coordinate U of the invalidating edge (U, v), all the way up to the final object of the existing order <. Combining these

		per informative ERC	per informative edge
trivially update		A1	B1
selectively update	enlarge AR with step length 1	A2	B2
	enlarge AR with step length n	A3	B3

Table 1: Six order updating strategies. An ERC or edge is *informative* if its ordering instruction is not satisfied by the existing linear order.

options gives rise to the six updating strategies in Table 1.

The strategy A1 is that of MRCD (Tesar 2014), i.e. updating an edge set E with the added ERC, and running MK (equivalent to RCD) over the updated E and the object set V, which remains unchanged. The strategies A2 and A3 process an ERC as a whole. To ensure the affected region on trial includes all `L`-valued objects, they modify Algorithm 2 by locating the left boundary of the affected region to the leftmost `L`-valued object, and the starting point of right boundary trials to the first object in U which is preceded by all `L`-valued objects. Here U is the set of `W`-valued objects of the added ERC, the common first coordinate of the edges in that ERC.

Test Data The strategies A1-3 and B1-3 are tested against two sets of randomly generated *consistent* ERC lists, that is, there exists a partial order and thus a linear order compatible with the ordering instructions of the ERCs in a list. The consistency is needed because the efficiency of cycle detection (which is usually quick, though) is out of concern of the current study.

One of the two data sets is *unstructured*, in the sense that the ordering instruction of each ERC in the list may or may not be entailed by those of the previous ERCs. The other is *forward informative*, or *forward non-redundant*, in the sense that ordering instruction of each ERC in the list is not entailed by those of the previous ERCs. The current study compares not only updating strategies, but also their responses to differently structured inputs. The unstructured ERC lists, which represent the most general situations, are contrasted with forward informative ones, which reflect the reality of phonological learning (e.g. Error-driven Learning and Output-driven Learning; see Tesar 2014).

The following procedure is repeated to construct an unstructured list. Without loss of generality, for a given number n of objects (i.e. constraints), we can regard the objects as natural numbers in the range $[1, n]$ and generate ERCs which follow the usual order $1 < ... < n$. To do so, we randomly choose a nonempty subset S_W of $[1, n]$ whose members we label as `W`. We then choose a random subset S_L of $[1, n] \backslash S_W$ whose greatest element does not exceed that of S_W. We label the members of S_L as `L` and the rest as `e`.

To construct a forward informative list, we can use the above procedure to find a random ERC, and check whether adding its edges into the existing set of edges leads MK to a different stratified linear order. We keep the ERC if it does, or keep trying until an informative ERC is found. Our way of construction ensures that each ERC brings the stratified linear order compatible with the edges accumulated up to that ERC closer to the trivial stratified linear order $\{1\} < ... < \{n\}$. Thus unlike the unstructured cases, the number of forward informative ERCs is finite. It is imaginable that our way of construction is expensive. In fact, as the stratified linear order becomes finer, the time it takes to find the next informative ERC grows very quickly. Thus to construct a list of a decent size (which is bounded anyway) in a reasonable amount of time, a special technique can be useful: we divide $[1, n]$ into four tiers of equal length:

$$[1, \lfloor n/4 \rfloor], [\lfloor n/4 \rfloor + 1, 2 \times \lfloor n/4 \rfloor], [2 \times \lfloor n/4 \rfloor + 1, 3 \times \lfloor n/4 \rfloor], [3 \times \lfloor n/4 \rfloor + 1, n],$$

and find informative ERCs for each tier (i.e. restricting `W`'s and `L`'s to one tier). We then do the same for divisions of $[1, n]$ into three tiers, two tiers, and one tier (i.e. leaving the whole range undivided).

Table 2 summarizes the data points constructed with the above methods. A pair x-y stands for an input consisting of y ERCs over x objects.

unstructured	80-4000, 80-8000, 160-4000, 160-8000, 240-4000, 240-8000
informative	240-310, 240-319, 320-4330, 320-383, 400-356, 400-389

Table 2: Data points used in the experiment.

Methods The experiment measures each strategy's execution time at the data points in Table 2. The average execution time of ten runs in a Python environment (3.4.4 on win32) is recorded as the performance of a strategy at a data point. Whenever an execution instance runs over ten minutes, the average processing time of an ERC up to that point is used to estimate the overall running time.

This is actually an underestimate, since a pilot experiment shows that the average processing time is non-decreasing as the running time grows.

Results As shown in Firgures 4 and 5, for both unstructured and forward informative ERC lists, selective updating which searches for a minimal affected region by exhaustion (i.e. pushing the right boundary of the affected region one object at a time, as in A2 and B2) is never a good idea: the computation becomes extremely expensive for a large number of objects (e.g. data point 240-8000).

Updating by ERCs are generally more efficient than breaking ERCs into individual edges, which can be seen by comparing A strategies with their corresponding B strategies, especially A1 *vs.* B1. We have two exceptions. One is A2, which falls way behind other updating-by-ERC strategies; the other is B3, an updating-by-edge strategy which fares equally well as those well behaving updating-by-ERC strategies. The contrast between A2 and B3 confirms the observation above, and further suggests that the number of trials in searching for an affected region weighs much more than processing unit (per ERC vs. per edge) in determining the efficiency of selective updating.

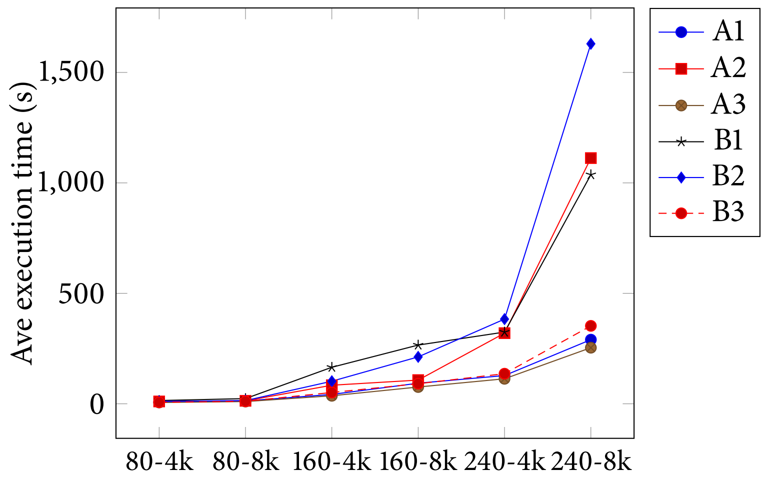

Figure 4: Performance of A1-3 and B1-3 at unstructured data points.

We observe that the selective updating strategies A3 and B3 behave similarly as the trivial updating strategy A1 for unstructured ERC lists; given the data points in Figure 4, they even share a similar rising trend as the input size grows. The advantage of selective updating (strengthened by a proper affected region searching technique) appears in processing structured ERC lists. While the running time of A1 rises slowly from 27.9s at point 240-310 to 122.41s at point 400-389, the running time of A3 and B3 maintain a level below 30s over the same range. Still, from the data points in Figure 5 it is hard to tell whether the running time growth rates of A3 and B3 in response to growth in the input size are of the same order as that of A1.

Finally, it is true across all six strategies that having the number of objects fixed, the execution time grows as the number of ERCs does, provided they are unstructured. But when they *are* structured, the same observation does not hold. For a given number of objects, more ERCs do not necessarily translate into longer execution time. This is most sharply depicted by the final decline in the curves of A2 and B2. It seems that it is the quality of ERCs that matters.

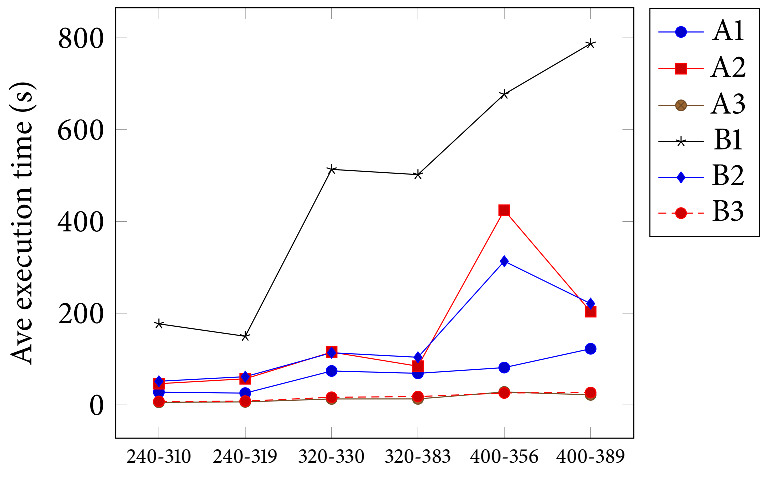

Figure 5: Performance of A1-3 and B1-3 at forward informative data points.

Discussion We have seen that a fine-tuned selective updating strategy fares at least as well as a trivial updating strategy when processing unstructured ERCs, and outdoes the latter if each ERC tells us something new. Naively searching for an affected region has been proved to be impractical; it weighs down selective updating when dealing with a large set of objects. Updating by ERCs and trying fewer affected regions (by pushing the right boundary of the latter further) point to the same effect: reducing the total runs of MK for each ERC addition. To this end, the technique employed A3 and B3—restricting trials of affected regions to those whose right boundaries is a precedence disjunct or the final object of the existing linear order—does a good job, but it is still not quite ideal.

It seems to be impossible to identify an affected region as small as possible in disjunctive topological sorting without paying certain computation cost. Thus for the future, the hope seems to be exploiting the information contained in a failed affected region trial, and using that as heuristics to choose the next trial more decisively: minimizing the boundary shift while maximizing the chance of finding a solution.

5 Conclusion

This paper has reformulated one of the most basic tasks in phonological learning, that of finding a (stratified) linear order compatible with given ERCs as disjunctive topological sorting. It has thereby paved the way for incorporating the idea of selective updating from recent developments in online topological sorting to deal with the incremental nature of phonological learning. The uncertainty introduced by disjunctive edges poses a serious challenge to identifying an affected region. An experimental study has suggested that the power of selective updating is best exercised when a balance is stricken between the size of the affected region we search for and the number of trials we pay for that. It remains to be explored what implications the current result might have for learning theory, and moreover, whether it might find application in other real-world problems.

References

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. Cambridge, MA: MIT Press, 3rd edition.

Haeupler, Bernhard, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert E. Tarjan. 2008. Faster algorithms for incremental topological ordering. In *International Colloquium on Automata, Languages, and Programming*, ed. Damgård I. Aceto L., Goldberg L.A., Halldórsson M.M., Ingólfsdóttir A., and Walukiewicz I., 421–433. Springer.

Kahn, Arthur B. 1962. Topological sorting of large networks. *Communications of the ACM* 5:558–562.

Knuth, Donald Ervin. 1997. *The art of computer programming: Fundamental algorithms*. Addison Wesley Longman, 3rd edition.

Pearce, David J., and Paul H. J. Kelly. 2004. A dynamic algorithm for topologically sorting directed acyclic graphs. In *International Workshop on Experimental and Efficient Algorithms*, ed. Ribeiro C.C. and Martins S.L., 383–398. Springer.

Prince, Alan. 2002. *Entailed ranking arguments*. Rutgers University, New Brunswick. ROA-500. Ms.

Prince, Alan, and Paul Smolensky. 2004. *Optimality theory constraint interaction in generative grammar*. Blackwell Publishing.

Shen, Alexander, and Nikolai Konstantinovich Vereshchagin. 2000. *Basic set theory*. American Mathematical Society.

Tesar, Bruce. 1995. Computational optimality theory. Doctoral Dissertation, University of Colorado.

Tesar, Bruce. 2014. *Output-driven phonology: Theory and learning*. Cambridge University Press.