DTU Library

Subsequence Automata with Default Transitions

Bille, Philip; Gørtz, Inge Li; Skjoldjensen, Frederik Rye

[Link back to DTU Orbit](#)

Subsequence Automata with Default Transitions

Philip Bille   Inge Li Gørtz   Frederik Rye Skjoldjensen

Technical University of Denmark
{phbi,inge,fskj}@dtu.dk

Abstract

Let S be a string of length n with characters from an alphabet of size σ. The *subsequence automaton* of S (often called the *directed acyclic subsequence graph*) is the minimal deterministic finite automaton accepting all subsequences of S. A straightforward construction shows that the size (number of states and transitions) of the subsequence automaton is $O(n\sigma)$ and that this bound is asymptotically optimal.

In this paper, we consider subsequence automata with *default transitions*, that is, special transitions to be taken only if none of the regular transitions match the current character, and which do not consume the current character. We show that with default transitions, much smaller subsequence automata are possible, and provide a full trade-off between the size of the automaton and the *delay*, i.e., the maximum number of consecutive default transitions followed before consuming a character.

Specifically, given any integer parameter k, $1 < k \leq \sigma$, we present a subsequence automaton with default transitions of size $O(nk \log_k \sigma)$ and delay $O(\log_k \sigma)$. Hence, with $k = 2$ we obtain an automaton of size $O(n \log \sigma)$ and delay $O(\log \sigma)$. On the other extreme, with $k = \sigma$, we obtain an automaton of size $O(n\sigma)$ and delay $O(1)$, thus matching the bound for the standard subsequence automaton construction. Finally, we generalize the result to multiple strings. The key component of our result is a novel hierarchical automata construction of independent interest.

1   Introduction

Let S be a string of length n with characters from an alphabet of size σ. A *subsequence* of S is any string obtained by deleting zero or more characters from S. The *subsequence automaton* (often called the *directed acyclic subsequence graph*) is the minimal deterministic finite automaton accepting all subsequences of S. Baeza-Yates [1] initiated the study of subsequence automata. They presented a simple construction using $O(n\sigma)$ size (size denotes the total number of states *and* transitions) and showed that this bound is optimal in the sense that there are subsequence automata of size at least $\Omega(n\sigma)$. They also considered variations with encoded input strings and multiple strings. Subsequently, several researchers have further studied subsequence automata (and its variants) [2, 3, 4, 5, 6, 7, 8, 9]. See also the surveys by Troníček [10, 11]. The general problem of *subsequence indexing*, not limited to automata based solutions, is investigated by Bille et al. [12].

In this paper, we consider subsequence automata in the context of *default transitions*, that is, special transitions to be taken only if none of the regular transitions match the current character, and which do not consume the current character. Each state has at most one default transition and hence the automaton remains deterministic. The key point of default transitions

is to reduce the size of standard automata at the cost of introducing a *delay*, i.e., the maximum number of consecutive default transition followed before consuming a character. For instance, given a pattern string of length m the classic Knuth-Morris-Pratt (KMP) [13] string matching algorithm may be viewed as an automaton with default transitions (typically referred to as *failure transitions*). This automaton has size $O(m)$, whereas the standard automaton with no default transitions would need $\Theta(m\sigma)$ space. The delay of the automaton in the KMP algorithm is either $O(m)$ or $O(\log m)$ depending on the version. Similarly, the Aho-Corasick string matching algorithm for multiple strings may also be viewed as an automaton with default transitions [14]. More recently, default transitions have also been used extensively to significantly reduce sizes of deterministic automata for regular expression [15, 16]. The main idea is to effectively enable states with large overlapping identical sets of outgoing transitions to "share" outgoing transitions using default transitions.

Surprisingly, no non-trivial bounds for subsequence automata with default transitions are known. Naively, we can immediately obtain an $O(n\sigma)$ size solution with $O(1)$ delay by using the standard subsequence automaton (without default transitions). At the other extreme, we can build an automaton with $n + 1$ states (each corresponding to a prefix of S) with a standard and a default transition from the state corresponding to the ith prefix to the state corresponding to the $i + 1$st prefix (the standard transition is labeled $S[i + 1]$). It is straightforward to show that this leads to an $O(n)$ size solution with $O(n)$ delay. Our main result is a substantially improved trade-off between the size and delay of the subsequence automaton:

Theorem 1. *Let S be a string of n characters from an alphabet of size σ. For any integer parameter k, $1 < k \leq \sigma$, we can construct a subsequence automaton with default transitions of size $O(nk\log_k \sigma)$ and delay $O(\log_k \sigma)$.*

Hence, with $k = 2$ we obtain an automaton of size $O(n \log \sigma)$ and delay $O(\log \sigma)$. On the other extreme, with $k = \sigma$, we obtain an automaton of size $O(n\sigma)$ and delay $O(1)$, thus matching the bound for the standard subsequence automaton construction.

To obtain our result, we first introduce the *level automaton*. Intuitively, this automaton uses the same states as the standard solution, but hierarchically orders them in a tree-like structure and samples a selection of their original transitions based on their position in the tree, and adds a default transition to the next state on a higher level. We show how to do this efficiently leading to a solution with $O(n \log n)$ size and $O(\log n)$ delay. To achieve our full trade-off from Theorem 1 we show how to augment the construction with additional ideas for small alphabets and generalize the level automaton with parameter k, $1 < k \leq \sigma$, where large k reduces the height of the tree but increases the number of transitions. In the final section we generalize the result to *multiple* strings.

2 Preliminaries

A *deterministic finite automaton* (DFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a set of nodes called *states*, δ is a set of labeled directed edges between states, called *transitions*, where each label is a character from the alphabet Σ, $q_0 \in Q$ is the *initial* state and $F \subseteq Q$ is a set of *accepting* states. No two outgoing transitions from the same state have the same label. The DFA is *incomplete* in the sense that every state does not contain transitions for every character in Σ. The *size* of A is the sum of the number of states and transitions.
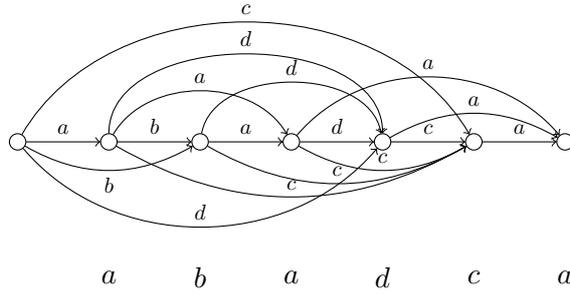
Figure 1: An example of an SA constructed from the string *abadca*.

We can think of A as an *edge-labeled directed graph*. Given a string P and a path p in A we say that p and P match if the concatenation of the labels on the transitions in p is P. We say that A *accepts* a string P if there is a path in A, from q_0 to any state in F, that matches P. Otherwise A *rejects* P.

A *deterministic finite automaton with default transitions* is a deterministic finite automaton AD where each state can have a single unlabeled default transition. Given a string P and a path p in AD we define a match between P and p as before, with the exception that for any default transition d in p the corresponding character in P cannot match any standard transition out of the start state of d. Definition of accepted and rejected strings are as before. The *delay* of AD is the maximum length of any path matching a single character, i.e., if the delay of AD is d then we follow at most $d - 1$ default transitions for every character that is matched in P.

A *subsequence* of S is a string P, obtained by removing zero or more occurrences of characters from S. The alphabet of the string S is denoted by $\Sigma(S)$. A *subsequence automaton* constructed from S, is a deterministic finite automaton that accepts string P iff P is a subsequence of S. A subsequence automaton construction is presented in [1]. This construction is often called the *directed acyclic subsequence graph* or DASG, but here we denote it SA. The SA has $n + 1$ states, all accepting, that we identify with the integers $\{0, 1, \ldots, n\}$. For each state s, $0 \leq s \leq n$, we have the following transitions:

- For each character α in $\Sigma(S[s + 1, n])$, there is a transition labeled α to the smallest state $s' > s$ such that $S[s'] = \alpha$.

The SA has size $O(n\sigma)$ since every state can have at most σ transitions. An example of an SA is given in Figure 1.

A *subsequence automaton with default transitions* constructed from S, denoted SAD, is a deterministic finite automaton with default transitions that accepts string P iff P is a subsequence of S.

The next section explores different configurations of transitions and default transitions in SADs.

3 New Trade-Offs for Subsequence Automata.

We now present a new trade-off for subsequence automata, with default transitions. We will gradually refine our construction until we obtain an automaton that gives the result presented

in Theorem 1. In each construction we have $n + 1$ states that we identify with the integers $\{0, 1, \ldots, n\}$. Each of these states represents a prefix of the string S and are all accepting states. We first present the *level automaton* that gives the first non-trivial trade-off that exploits default transitions. The general idea is to construct a hierarchy of states, such that every path that only uses default transitions is guaranteed to go through states where the outdegree increases at least exponentially. The level automaton is a SAD of size $O(n \log n)$ and delay $O(\log n)$. By arguing that any path going through a state with outdegree σ will do so by taking a regular transition, we are able to improve both the size and delay of the level automaton. This results in the *alphabet-aware level automaton* which is a SAD of size $O(n \log \sigma)$ and delay $O(\log \sigma)$. Finally we present a generalized construction that gives a trade-off between size and delay by letting parameter k, $1 < k \leq \sigma$, be the base of the exponential increase in outdegree on paths with only default transitions. This SAD has size $O(nk \log_k \sigma)$ and delay $O(\log_k \sigma)$. With $k = 2$ we get an automaton of size $O(n \log \sigma)$ and delay $O(\log \sigma)$. In the other extreme, for $k = \sigma$ we get an automaton of size $O(n\sigma)$ and delay $O(1)$.

3.1 Level Automaton

The level automaton is a SAD with $n + 1$ states that we identify with the integers $\{0, 1, \ldots, n\}$. All states are accepting. For each state $i > 0$, we associate a level, level(i), given by:

$$\text{level}(i) = \max(\{x \mid i \bmod 2^x = 0\})$$

Hence, level(i) is the exponent of the largest power of two that divides i. The level function is in the literature known as the ruler function. We do not associate any level with state 0. Note that the maximum level of any state is $\log_2 n$. For a nonnegative integer s, we define $\bar{s}$ to be the smallest integer $\bar{s} > s$ such that level$(\bar{s}) \geq$ level$(s) + 1$.

The transitions in the level automaton are as follows: From state 0 we have a default transition to state 1 and a regular transition to state 1 with label $S[1]$. For every other state s, $1 \leq s \leq n$, we have the following transitions:

- A default transition to state $\bar{s}$. If no such state exist, the state s does not have a default transition.

- For each character α in $\Sigma(S[s + 1, \min(\bar{s}, n)])$, there is a transition labeled α to the smallest state $s' > s$ such that $S[s'] = \alpha$.

An example of the level automaton constructed from the string *abacbabcabad* and alphabet $\{a, b, c, d\}$ is given in Figure 2. The dashed arrows denote default transitions and the vertical position of the states denotes their level.

We first show that the level automaton is a SAD for S, i.e., the level automaton accepts a string iff the string is a subsequence of S. To do so suppose that P is a string of length m accepted by the level automaton and let $s_1, s_2, \ldots, s_m$ be the sequence of states visited with regular transitions on the path that accepts P. From the definition of the transition function, we know that if a transition with label α leads to state s', then $S[s'] = \alpha$. This means that $S[s_1]S[s_2] \ldots S[s_m]$ spells out a subsequence of S if the sequence $s_1, s_2, \ldots, s_m$ is strictly increasing. From the definition of the transitions, a state s only have transitions to states s' if $s' > s$. Hence, the sequence is strictly increasing.

For the other direction, we show that the level automaton simulates the SA. At each state s, trying to match character α, we find the smallest state $s' > s$ such that s' has an incoming
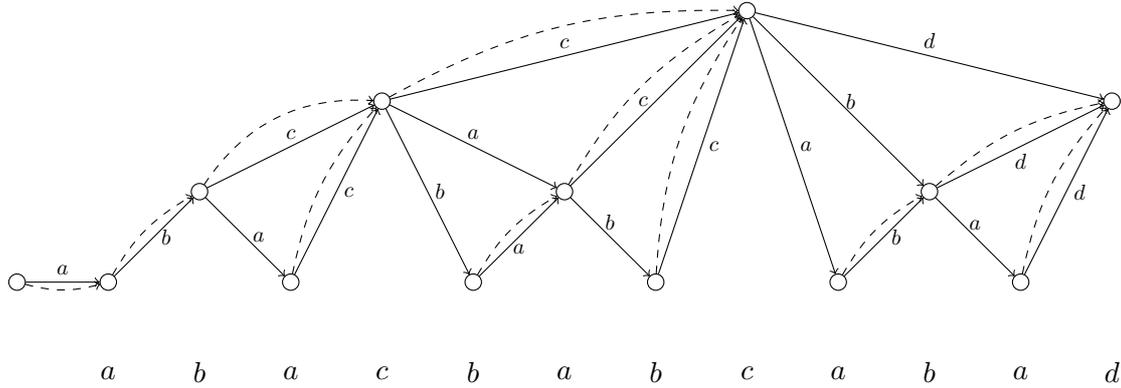
Figure 2: The level automaton constructed from the string *abacbabcabad*.

transition with label α: By the construction, either s has an outgoing transition leading directly to s' *or* we follow default transitions until reaching the first state with a transition to s'. This means that the states visited with standard transitions in the level automaton are the same states that are visited in the SA. Since the SA accepts all subsequences of S this must also hold for the level automaton.

3.1.1 Analysis

The following shows that the number of outgoing transitions increase with a factor two when the level increase by one. For all $s > 0$, we have the following property of $\overline{s}$ and level(s):

$$\overline{s} - s = 2^{\text{level}(s)} \tag{1}$$

By definition, $2^{\text{level}(s)}$ divides s. This means that we can write s as $c \cdot 2^{\text{level}(s)}$, where c is a uneven positive integer. We know that c is uneven because $2^{\text{level}(s)}$ is the largest power of two that divides s. The next integer, larger than s, that $2^{\text{level}(s)}$ divides is $s' = s + 2^{\text{level}(s)}$. This means that $\overline{s} \geq s'$. We can rewrite s' as follows: $s' = s + 2^{\text{level}(s)} = c \cdot 2^{\text{level}(s)} + 2^{\text{level}(s)} = (c + 1) \cdot 2^{\text{level}(s)}$. Since c is uneven we know that $c + 1$ is even so we can rewrite s' further: $s' = \frac{(c+1)}{2} \cdot 2^{\text{level}(s)+1}$. This shows that $2^{\text{level}(s)+1}$ divides s' which means that $s' = \overline{s}$ and we conclude that $\overline{s} - s = 2^{\text{level}(s)}$.

Since the maximal level of any state is $\log_2 n$ and the level increase every time we follow a default transition, the delay of the level automaton is $O(\log n)$.

At each level l we have $O(n/2^{l+1})$ states, since every 2^lth state is divided by 2^l, and 2^l is the largest divisor in every second of these cases. Since $\overline{s} - s = 2^{\text{level}(s)}$ each state at level l has at most $2^l + 1$ outgoing transitions. Therefore, each level contribute with size at most $n/2^{l+1} \cdot (2^l + 1) = O(n)$. Since we have at most $O(\log n)$ levels, the total size becomes $O(n \log n)$.

In summary, we have shown the following result.

Lemma 1. *Let S be a string of n characters. We can construct a subsequence automaton with default transitions of size $O(n \log n)$ and delay $O(\log n)$.*
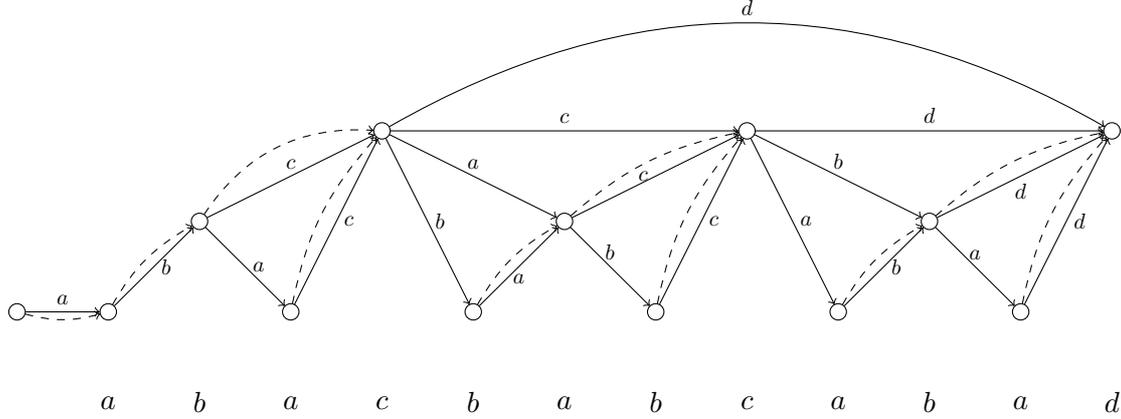
5

Figure 3: The alphabet level automaton constructed from the string *abacbabcabad*.

3.2 Alphabet-aware level automaton

We introduce the *Alphabet-aware level automaton*. When the level automaton reaches a state s where $\overline{s} - s \geq \sigma$, then s can have up to σ outgoing transitions without violating the space analysis above. The level automaton only has a transition for each character in $\Sigma(S[s + 1, \min(\overline{s}, n)])$. Hence, for all states s in the alphabet-aware level automaton where $\overline{s} - s \geq \sigma$, we let s have a transition for each symbol α in Σ, to the smallest state $s' > s$ such that $S[s'] = \alpha$. No matching path can take a default transition from a state with σ outgoing transitions. Hence, states with σ outgoing transitions do not need default transitions.

We change the level function to reflect this. For each state $1 \leq i \leq n$ we have that:

$$\text{level}(i) = \min(\lceil \log_2 \sigma \rceil, \max(\{x \mid i \bmod 2^x = 0\}))$$

The transitions in the alphabet-aware level automaton is as follows: From state 0 we have a default transition to state 1 and a regular transition to state 1 with label $S[1]$. For every other state s, $1 \leq s \leq n$, we have the following transitions:

- A default transition to state $\overline{s}$. If no such state exist, the state s does not have a default transition.

- If $\overline{s} - s < \sigma$ then for each character α in $\Sigma(S[s + 1, \min(\overline{s}, n)])$, there is a transition labeled α to the smallest state $s' > s$ such that $S[s'] = \alpha$.

- If $\overline{s} - s \geq \sigma$ then for each character α in $\Sigma(S[s + 1, n])$, there is a transition labeled α to the smallest state $s' > s$ such that $S[s'] = \alpha$.

An example of the alphabet-aware level automaton constructed from the string *abacbabcabad* and alphabet $\{a, b, c, d\}$ is given in Figure 3. The level automaton in Figure 2 is constructed from the same string and the same alphabet. For comparison, state 4 in Figure 3 now has outdegree σ and has transitions to the first succeeding occurrence of any unique character and state 8 has been constrained to level $\lceil \log_2 \sigma \rceil$.

The alphabet-aware level automaton is a SAD by the same arguments that led to Lemma 1.

The delay is now bounded by $O(\log \sigma)$ since no state is assigned a level higher than $\lceil \log_2 \sigma \rceil$. The size of each level is still $O(n)$. Hence, the total size becomes $O(n \log \sigma)$

In summary, we have shown the following result.

6

Lemma 2. *Let S be a string of n characters. We can construct a SAD of S with size $O(n \log \sigma)$ and delay $O(\log \sigma)$.*

3.3 Full trade-off

We can generalize the construction above by introducing parameter k, $1 < k \leq \sigma$, which is the base of the exponential increase in outdegree of states on every path that only uses default transitions. Now, when we follow a default transition from s to $\overline{s}$, the number of outgoing transitions increase with a factor k instead of a factor 2. This gives a trade-off between size and delay in the SAD determined by k. Increasing k gives a shorter delay of the SAD but increases the size and vice versa.

Each state, except state 0, is still associated with a level, but we need to generalize the level function to account for the parameter k. For every k and i we have that:

$$\text{level}(i, k) = \min(\lceil \log_k \sigma \rceil, \max(\{x \mid i \bmod k^x = 0\}))$$

Now, the level function gives the largest power of k that divides i.

The transitions in the generalized alphabet-aware level automaton is as follows: From state 0 we have a default transition to state 1 and a regular transition to state 1 with label $S[1]$. For every other state s, $1 \leq s \leq n$, we have the following transitions:

- A default transition to state $\overline{s}$. If no such state exist, the state s does not have a default transition.

- If $\overline{s} - s < \sigma$ then for each character α in $\Sigma(S[s+1, \min(\overline{s}, n)])$, there is a transition labeled α to the smallest state $s' > s$ such that $S[s'] = \alpha$.

- If $\overline{s} - s \geq \sigma$ then for each character α in $\Sigma(S[s+1, n])$, there is a transition labeled α to the smallest state $s' > s$ such that $S[s'] = \alpha$.

We can show that the generalized alphabet-aware level automaton is a SAD by the same arguments that led to Lemma 2.

3.3.1 Analysis

The delay is bounded by $O(\log_k \sigma)$ because no state is assigned a level higher than $\lceil \log_k \sigma \rceil$.

With the new definition of the level function we have that

$$\overline{s} - s \leq k^{\text{level}(s,k)+1}$$

for all $s > 0$. This expression bounds the number of outgoing transitions from state s.

At level l we have $O(n(k-1)/(k^{l+1}))$ states each with $O(k^{l+1})$ outgoing transitions such that each level has size $O(nk)$. The size of the automaton becomes $O(nk \log_k \sigma)$ because we have $O(\log_k \sigma)$ levels.

In summary, we have shown Theorem 1.

4 Subsequence automata for multiple strings

Troníček et al. [3] generalizes the simple subsequence automaton to multiple strings. Given a set of strings $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$ of length $n1, n2, \ldots, n_N$, two types of automata are presented: The subsequence automaton accepts a pattern P iff P is a subsequence of *some* string in $\mathcal{S}$ and the *common* subsequence automaton accepts P iff P is a subsequence of *every* string in $\mathcal{S}$. When $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ and $n_1 = n_2 = \ldots = n_N = n$, a $\Omega(n^N/(N+1)^N N!))$ lower bound on the number of states is shown for the subsequence automaton [2]. For both automata, the number of states is trivially upper bounded by $O(n_1 \cdot n_2 \cdot \ldots \cdot n_N)$ such that the total size becomes $O(\sigma \cdot n_1 \cdot n_2 \cdot \ldots \cdot n_N)$. We can reduce the size of these automata by augmenting them with default transitions. This generalization is in the same spirit as the naive generalization of the single string automaton in section 1: We introduce default transitions and save a factor σ in the space but also introduce a delay. Consider the naive common subsequence automaton with default transitions: For two strings S_1, S_2 we have $n_1 \cdot n_2 + 1$ states that we identify with the points $\{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \cup \{(0,0)\}$. For each state (s_1, s_2) we have the following transitions:

- A default transition to state $(s_1 + 1, s_2 + 1)$. If no such state exist, the state (s_1, s_2) does not have a default transition.

- If character $S_1[s_1 + 1]$ is in $\Sigma(S_2[s_2 + 1, n_2])$, there is a transition labeled $S_1[s_1 + 1]$ to the state $(s_1 + 1, s_2')$ such that $s_2' > s_2$ is the minimal index where $S_2[s_2'] = S_1[s_1 + 1]$.

- If character $S_2[s_2 + 1]$ is in $\Sigma(S_1[s_1 + 1, n_1])$, there is a transition labeled $S_2[s_2 + 1]$ to the state $(s_1', s_2 + 1)$ such that $s_1' > s_1$ is the minimal index where $S_1[s_1'] = S_2[s_2 + 1]$.

The states of the automaton represents the progression in S_1 and S_2, such that state (s_1, s_2) represents that subsequences of the prefixes $S_1[1, s_1]$ and $S_2[1, s_2]$ have been used to match a prefix of P. Each state (s_1, s_2) considers the symbols $S_1[s_1 + 1]$ and $S_2[s_2 + 1]$ for matching with the next symbol in P. If this is not possible, a default transition is followed to state $(s_1 + 1, s_2 + 1)$.

For this automaton the size is $O(n_1 \cdot n_2)$ and the delay is $O(\min(n_1, n_2))$. Hence, we save a σ factor in the size, but introduce a significant delay. As we did for the subsequence automaton for a single string, we introduce a level automaton that associates a level with each state. In this way we are able to reduce the delay significantly with only a small increase in size. For simplicity we only present our construction for the common subsequence automaton, but it follows immediately that the construction also applies to the subsequence automaton.

4.1 The alphabet-aware level automaton for two strings

The alphabet-aware level automaton for two strings S_1, S_2 have $n_1 \cdot n_2 + 1$ states that we identify with the points $\{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \cup \{(0,0)\}$. We define the *diagonal* of a state (i, j), as the set of states $\{(i+k, j+k) \mid 0 < i+k \leq n_1 \text{ and } 0 < j+k \leq n_2\}$. We say that states belong to the same diagonal if the diagonals of the states defines identical sets of states. For states $(s_1, s_2), (s_1', s_2')$ in the same diagonal, $(s_1, s_2) < (s_1', s_2')$ if $s_1 < s_1'$. For each state (s_1, s_2) we associates the integer $\min(s_1, s_2)$, which is also its position in the diagonal, such that $(s_1, s_2) - (s_1', s_2') = \min(s_1, s_2) - \min(s_1', s_2')$ and $(s_1, s_2) \bmod x = \min(s_1, s_2) \bmod x$. With each diagonal of states, we associate a level structure identical to the one used in the alphabet-aware level automaton for a single string. Now, when following a default transition from state

(s_1, s_2) to (s_1+k, s_2+k), $k > 0$, every unique symbol in $\Sigma(S_1[s_1+1, s_1+k]) \cup \Sigma(S_2[s_2+1, s_2+k])$ contributes with a transition. For each state (i, j), $1 \le i \le n_1, 1 \le j \le n_2$, we again associate a level:

$$\text{level}((i, j)) = \min(\lceil \log_2 \sigma \rceil, \max(\{x \mid (i, j) \bmod 2^x = 0\}))$$

For a pair of positive integers (i, j), we define $\overline{(i, j)} > (i, j)$ to be the smallest pair of integers in the same diagonal such that $\text{level}((i, j)) < \text{level}(\overline{(i, j)})$.

The alphabet-aware level automaton for two strings S_1, S_2 has the following transitions: State $(0, 0)$ has a transition labeled $S_1[1]$ to state $(1, s_2)$ where s_2 is the minimal index such that $S_2[s_2] = S_1[1]$, a transition labeled $S_2[1]$ to state $(s_1, 1)$ where s_1 is the minimal index such that $S_1[s_1] = S_2[1]$ and a default transition to state $(1, 1)$. These transitions only exists if the indices exists. Every other state state (s_1, s_2), where $(\overline{s_1}, \overline{s_2}) = \overline{(s_1, s_2)}$, have the following transitions:

- A default transition to state $\overline{(s_1, s_2)}$. If no such state exist, the state (s_1, s_2) does not have a default transition.

- If $\overline{(s_1, s_2)} - (s_1, s_2) < \sigma$ then for each character α in $\Sigma(S_1[s_1+1, \min(\overline{s_1}, n_1)]) \cup \Sigma(S_2[s_2+1, \min(\overline{s_2}, n_2)])$, there is a transition labeled α to the state (s_1', s_2'), where $s_1' > s_1$ and $s_2' > s_2$ are the minimal indices such that $S_1[s_1'] = S_2[s_2'] = \alpha$.

- If $\overline{(s_1, s_2)} - (s_1, s_2) \ge \sigma$ then for each character α in $\Sigma(S_1[s_1+1, n_1]) \cup \Sigma(S_2[s_2+1, n_2])$, there is a transition labeled α to the state (s_1', s_2'), where $s_1' > s_1$ and $s_2' > s_2$ are the minimal indices such that $S_1[s_1'] = S_2[s_2'] = \alpha$.

An example of an incomplete common subsequence automaton for two strings is given in Figure 4.

4.1.1 Analysis

For every pair of positive integers (i, j), we have the following property:

$$\overline{(i, j)} - (i, j) = 2^{\text{level}((i,j))}$$

This property follows from the same argument that led to equation (1). The number of transitions out of every state s, is now bounded by $2 \cdot 2^{\text{level}(s)}$ since both S_1 and S_2 can contribute with up to $2^{\text{level}(s)}$ transitions.

We can calculate the size of the alphabet-aware level automaton for two strings by summing up the space contribution from each diagonal of states. Let d be a diagonal consisting of $|d|$ states. Then the size of d is $O(|d| \log \sigma)$, since each diagonal has the size of an alphabet-aware level automaton for one string. If D is the set of all diagonals, then the total size of the automaton becomes

$$\sum_{d \in D} O(|d| \log \sigma) = \log \sigma \cdot \sum_{d \in D} O(|d|) = O(\log \sigma \cdot n_1 \cdot n_2)$$

The last step is possible since the sum over the states in all diagonals is exactly the number of states in the automaton. In summary we have shown the following result:

Lemma 3. *Let S_1, S_2 be strings of length n_1 and n_2 over an alphabet of size σ. We can construct a subsequence automaton and a common subsequence automaton with default transitions of size $O(n_1 n_2 \log \sigma)$ and delay $O(\log \sigma)$.*
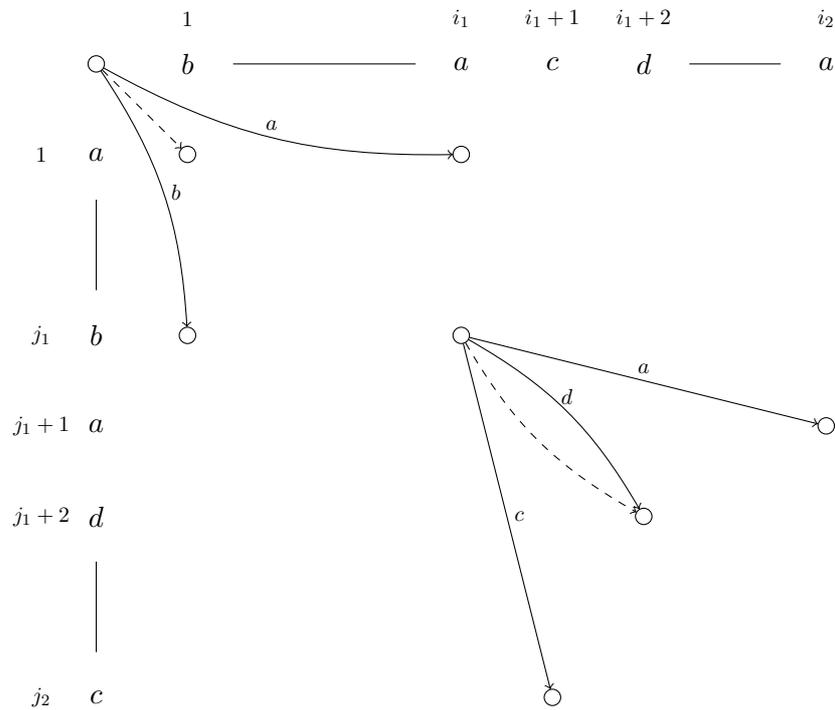
9

Figure 4: An incomplete common subsequence automaton for two strings S_1, S_2, laid out in a two-dimensional grid. State $(0,0)$ has a transition labeled $a = S_2[1]$ to state $(i_1, 1)$ and a transition labeled $b = S_1[1]$ to state $(0, j_1)$. State (i_1, j_1) is at level 1 and has a transition for each unique character in $\Sigma(S_1[i_1 + 1, i_1 + 2]) \cup \Sigma(S_2[j_1 + 1, j_1 + 2]) = \{a, c, d\}$. Transitions out of the remaining states are missing from the illustration.

4.2 The alphabet-aware level automaton for multiple strings

The alphabet-aware level automaton for the set of strings $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$ have $1 + \prod_{i=1}^{N} S_i$ states that we identify with the set of integer points $\{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \times \ldots \times \{1, \ldots, n_N\} \cup \{(0, 0, \ldots, 0)\}$. Hence, a state in the automaton corresponds to a tuple with N elements $(s_1, s_2, \ldots, s_N)$. We generalize the definition of diagonals for dimension N as follows. The diagonal of a state $(s_1, s_2, \ldots, s_N)$ is the set of states:

$$\{(s_1 + k, s_2 + k, \ldots, s_N + k) \mid \bigwedge_{i=1}^{N} 0 < s_i + k \le n_i\}$$

Again, states belong to the same diagonal if the diagonal of each state defines identical sets of states, and for states $(s_1, s_2, \ldots, s_N), (s_1', s_2', \ldots, s_N')$, in the same diagonal, $(s_1, s_2, \ldots, s_N) < (s_1', s_2', \ldots, s_N')$ if $s_1 < s_1'$. For each state $(s_1, s_2, \ldots, s_N)$ we associate the integer $\min(s_1, s_2, \ldots, s_N)$ and define subtraction and modulo operations on states as in the previous section.

With each state we again associate the level:

$$\text{level}((s_1, s_2, \ldots, s_N)) = \min(\lceil \log_2 \sigma \rceil, \max(\{x \mid (s_1, s_2, \ldots, s_N) \bmod 2^x = 0\}))$$

For a tuple of positive integers $(s_1, s_2, \ldots, s_N)$, we define $\overline{(s_1, s_2, \ldots, s_N)} > (s_1, s_2, \ldots, s_N)$ to be the smallest tuple of integers in the same diagonal such that $\text{level}((s_1, s_2, \ldots, s_N)) < \text{level}(\overline{(s_1, s_2, \ldots, s_N)})$.

The alphabet-aware level automaton for multiple strings, $S_1, S_2, \ldots, S_N$, has the following transitions: State $(0, 0, \ldots, 0)$ has a transition labeled $S_i[1]$ to state $(s_1, s_2, \ldots, s_N)$ where $s_i = 1$, such that s_j is the minimal index where $S_j[s_j] = S_i[1]$, for every $i = \{1, 2, \ldots, N\}$ and $j \ne i$ and a default transition to state $(1, 1, \ldots, 1)$. Every other state $(s_1, s_2, \ldots, s_N)$, where $(\overline{s_1}, \overline{s_2}, \ldots, \overline{s_N}) = \overline{(s_1, s_2, \ldots, s_N)}$, has the following transitions:

- A default transition to state $\overline{(s_1, s_2, \ldots, s_N)}$. If no such state exist, the state $(s_1, s_2, \ldots, s_N)$ does not have a default transition.

- If $\overline{(s_1, s_2, \ldots, s_N)} - (s_1, s_2, \ldots, s_N) < \sigma$ then for each character α in $\bigcup_{i=1}^{N} \Sigma(S_i[s_i + 1, \min(\overline{s_i}, n_i)])$ there is a transition labeled α to the state $(s_1', s_2', \ldots, s_N')$, where, $s_i' > s_i$ is the minimal index such that $S_i[s_i'] = \alpha$, for all $1 \le i \le N$

- If $\overline{(s_1, s_2, \ldots, s_N)} - (s_1, s_2, \ldots, s_N) \ge \sigma$ then for each character α in $\bigcup_{i=1}^{N} \Sigma(S_i[s_i + 1, n_i])$ there is a transition labeled α to the state $(s_1', s_2', \ldots, s_N')$, where $s_i' > s_i$ is the minimal index such that $S_i[s_i'] = \alpha$, for all $1 \le i \le N$.

4.2.1 Analysis

For each state $(s_1, s_2, \ldots, s_N)$ we have that

$$\overline{(s_1, s_2, \ldots, s_N)} - (s_1, s_2, \ldots, s_N) = 2^{\text{level}((s_1, s_2, \ldots, s_N))}$$

The number of transitions out of every state s, is now bounded by $N \cdot 2^{\text{level}(s)}$ because each of the N strings can contribute with up to $2^{\text{level}(s)}$ transitions.

We can calculate the size of the alphabet-aware level automaton for N strings by summing up the space contribution from each diagonal of states. Let d be a diagonal consisting of $|d|$

states. Then the size of d is $O(N|d|\log\sigma)$. If D is the set of all diagonals, then the total size of the automaton becomes

$$O\left(\sum_{d\in D} N|d|\log\sigma\right) = O\left(N\log\sigma \cdot \sum_{d\in D}|d|\right) = O\left(N\log\sigma \cdot \prod_{i=1}^{N} n_i\right)$$

The last step is possible since the sum over the states in all diagonals is the number of states in the automaton. In summary we have shown the following result:

Theorem 2. *Let $S_1, S_2, \ldots S_N$ be a set of strings of length $n_1, n_2, \ldots, n_N$ over an alphabet of size σ. We can construct a subsequence automaton and a common subsequence automaton with default transitions of size $O(N\log\sigma \cdot \prod_{i=1}^{N} n_i)$ and delay $O(\log\sigma)$.*

References

[1] Baeza-Yates, R.A.: Searching subsequences. Theoret. Comput. Sci. **78**(2) (1991) 363–376

[2] Troníček, Z., Shinohara, A.: The size of subsequence automaton. Theoret. Comput. Sci. **341**(1) (2005) 379–384

[3] Crochemore, M., Melichar, B., Troníček, Z.: Directed acyclic subsequence graph: Overview. J. Disc. Algorithms **1**(3-4) (2003) 255–280

[4] Crochemore, M., Tronıcek, Z.: Directed acyclic subsequence graph for multiple texts. Technical Repport, Institut Gaspard-Monge (1999) 99–13

[5] Troníček, Z.: Episode matching*. In: Proc. 12th. CPM. (2001) 143–146

[6] Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S.: Online construction of subsequence automata for multiple texts. In: Proc. 7th SPIRE. (2000) 146–152

[7] Farhana, E., Ferdous, J., Moosa, T., Rahman, M.S.: Finite automata based algorithms for the generalized constrained longest common subsequence problems. In: Proc. 17th SPIRE. (2010) 243–249

[8] Bannai, H., Inenaga, S., Shinohara, A., Takeda, M.: Inferring strings from graphs and arrays. In: Proc. 28th MFCS. (2003) 208–217

[9] Tronìček, Z.: Operations on DASG. In: Proc. 4th WIA. (1999) 82–91

[10] Troníček, Z.: Searching subsequences. Ph. D. Thesis, Department of Computer Science and Engineering, FEE CTU in Prague (2001)

[11] Tronîcek, Z.: Common subsequence automaton. In: Proc. 8th CIAA. (2003) 270–275

[12] Bille, P., Farach-Colton, M.: Fast and compact regular expression matching. Theoret. Comput. Sci. **409** (2008) 486 – 496

[13] Knuth, D.E., James H. Morris, J., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(2) (1977) 323–350

[14] Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18**(6) (1975) 333–340

[15] Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., Turner, J.: Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In: Proc. 12th SIGCOMM. (2006) 339–350

[16] Hayes, C.L., Luo, Y.: Dpico: a high speed deep packet inspection engine using compact finite automata. In: Proc. 3rd ANCS. (2007) 195–203