TARTU ÜLIKOOL
MATEMAATIKAINFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

Neeme Loorits

Pattern Matching for Superpositional Graphs

Magistritöö (30 EAP)

Juhendaja: Mati Tombak

Autor: .... ”...” mai 2012
Juhendaja: .... ”...” mai 2012

Lubada kaitsmisele

Professor .... ”...” mai 2012

TARTU 2012

Contents

1 Introduction

The goal of the current master's thesis is to find a fast algorithm for pattern matching for superpositional graphs.

Superpositional graphs (SPG) were introduced in [3] as a skeleton of structurally synthesized binary decision diagrams, introduced by R. Ubar in [6] (see [5] for a historical overview). L. Vohandu, A. Peder and M. Tombak defined a problem of pattern matching for SPG in [7]. They found a bijection between SPG-s and separable permutations and posed a hypothesis, that the problems of pattern matching for SPG and separable permutations are equivalent. Furthermore, Mati Tombak, advisor for the current master's thesis, had proved in a yet unpublished paper, that for every solution to a pattern matching problem for SPGs there is a solution to a pattern matching problem for separable permutations and vice versa, and pattern matching for separable permutations is reducible in linear time to pattern matching for SPGs. The problem of pattern matching for permutations was posed by H. Wilf (see [1]).

Let n be a length of a text and k a length of a pattern. P. Bose, J. Buss and A.Lubiw in [1] proved, that the general decision problem of pattern matching is $NP-$complete, but the counting problem can be solved in $O(kn^6)$ time in the case, if the pattern is a separable permutation. This result was improved by L.Ibarra in [2] to $O(kn^4)$.

In the current master's thesis we build an algorithm for pattern matching for SPGs, which counts a number of matches in time $O(kn)$. We prove, that every solution to the problem for an SPG is a solution of a pattern matching problem for the corresponding separable permutation and vice versa. As a consequence, we have an algorithm for counting matches, working in time $O(kn)$ for the case, when the text and the pattern are both separable permutations.

In Chapter 2 we define superpositional graphs (SPGs) and show how an arbitrary SPG can be constructed using elementary graphs as building blocks. SPG deconstruction into elementary graphs is an important step in SPG pattern matching. In Chapter 3 we define separable permutations and present algorithms implementing bijection between SPGs and separable permutations. In Chapter 4 we present an algorithm for SPG pattern matching. We also provide a proof for algorithm correctness. In Chapter 5 we present two helper algorithms and measure the performance of given algorithms. Author's contribution to the current master's thesis are pattern matching algorithms and performance calculations (Chapters 4 and 5, in close cooperation with the advisor).

The contents of this master's thesis were submitted for publishing to a pattern matching conference DAGM-OAGM 2012 in Graz, Austria (Appendix A). We expect an answer regarding acceptance from the organizers by 18 June 2012.

2 Superpositional Graphs

Definition 1. A *binary graph* is an oriented acyclic connected graph with a root and two terminal nodes (sinks) labeled with T_0 and T_1. Every internal (i.e., not terminal) node v has two immediate successors denoted by $high(v)$ and $low(v)$. An edge $a \rightarrow b$ is called 0-*edge* (1-*edge*) if $low(a) = b$ ($high(a) = b$). [1]

A *path* from node u to node v ($u \rightsquigarrow v$) is a sequence $w_0, \ldots, w_k$ of nodes where $w_0 = u$, $w_k = v$ and for each $0 \leq i < k$, $w_{i+1} = high(w_i)$ or $w_{i+1} = low(w_i)$. A 0-*path* (1-*path*) is a path which contains only 0-edges (1-edges).

Definition 2. A binary graph G is *traceable* if there exists a directed path through all internal nodes of G (Hamiltonian path).

A binary graph is acyclic, therefore, if the Hamiltonian path exists then it is unique. The unique Hamiltonian path gives a canonical enumeration of the nodes of a traceable binary graph. Finding the Hamiltonian path of a binary graph G is a special case of the classical task of topological sorting of the nodes of a graph and can be done in time $O(n)$.

We are interested in traceable binary graphs only. Therefore we use the canonical enumeration of nodes and draw our graphs so, that the nodes are in straight line according to the canonical enumeration, 1-edges are drawn above the line and 0-edges below the line (Figure 1).

Definition 3. A binary graph G is *homogeneous* if only one type of edges (i.e. either 1-edges only or 0-edges only) enters into every node $v \in V(G)$ (Figure 2).

Definition 4. We say that a binary traceable graph is *strongly planar* if it has no crossing 0-edges and no crossing 1-edges in its stretched drawing (Figure 3).

It is obvious that if a binary graph is strongly planar then it is also planar, while the opposite does not hold in general.

In [4] there was proven, that every strongly planar traceable binary graph is homogeneous.

[1] Binary graph is a skeleton of a Binary Decision Diagram ([4]).

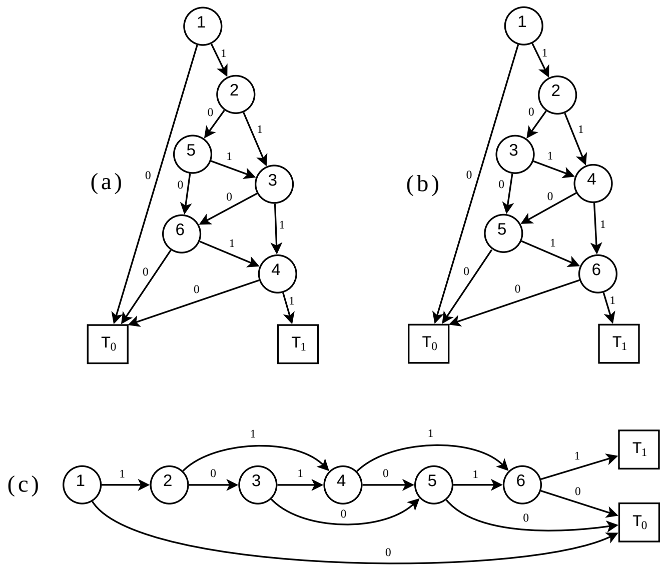

Figure 1: A traceable binary graph (a); the graph after relabeling (b); the graph after stretching (c).

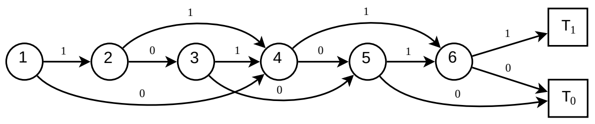

Figure 2: A inhomogeneous binary graph (node 4 violates the condition).

Definition 5. We say that a binary traceable graph is 1-*cofinal (*0-*cofinal)* if all 1-edges (0-edges) starting between the endpoints of some 0-edge (1-edge) and crossing it end in the same node.

Figure 4 illustrates the notion of 0-cofinality. For establishing 0-cofinality, one of the long edges ending at 5 and 6 must be redirected to the other vertex.

Definition 6. A binary traceable graph is *cofinal* if it is both 1-cofinal and 0-cofinal.

Definition 7. We call a binary graph, which is traceable, strongly planar and cofinal, a *superpositional graph*.

Another possibility to define superpositional graphs is using a *superposition*.

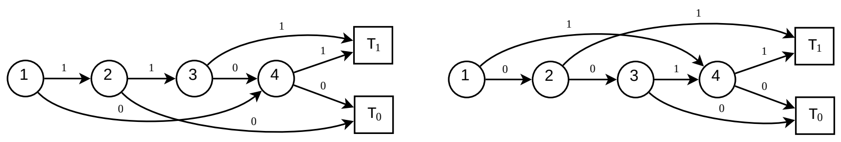

Figure 3: Two binary graphs, which are not strongly planar.

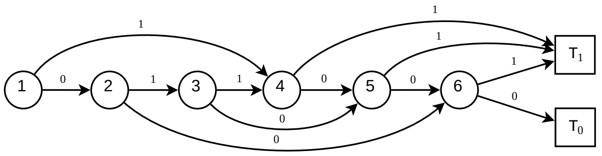

Figure 4: Situation forbidden by 0-cofinality.

Definition 8. Let G and E be two binary graphs. A *superposition* of E into G in place of internal node v ($G_{v\leftarrow E}$) is a graph, which we obtain by deleting v from G and redirecting all edges, pointing to v, to the root of E, all edges of E pointing to terminal node T_1 to the node $high(v)$ and all edges pointing to the terminal node T_0 to the node $low(v)$.

Let A, C and D be binary graphs, whose descriptions are shown in Fig. 5.

Definition 9. A *class of superpositional graphs* (SPG) is defined inductively as follows:

1° Graph $A \in SPG$.

2° If $G \in SPG$ and v is an internal node of G, then $G_{v\leftarrow C} \in SPG$ and $G_{v\leftarrow D} \in SPG$.

Note that $C = A_{v\leftarrow C} \in SPG$ and $D = A_{v\leftarrow D} \in SPG$.

In [4] there was proven, that definitions 7 and 9 are equivalent.

Theorem 1 ([4]). *If $G,H \in SPG$ and v is an internal node of G, then $G_{v\leftarrow H} \in SPG$ (the class of superpositional graphs is closed under superposition).*

Elementary graphs C and D can be considered as constructors of superpositional graphs (we use bold $\mathbf{C},\mathbf{D}$ to emphasize their role as constructors): if E and F are SPG with different sets of nodes, then $\mathbf{C}(E,F) = (\mathbf{C}[u \leftarrow E])[v \leftarrow F]$ and $\mathbf{D}(E,F) = (\mathbf{D}[u \leftarrow E])[v \leftarrow F]$ are SPGs. There was shown in [7] that constructors of superpositional graphs $\mathbf{C}$ and $\mathbf{D}$ are associative, so it is legal to use "long"

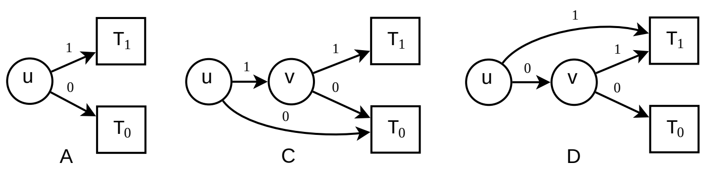

Figure 5: Binary graphs A, C and D.

constructors $\mathbf{C}(E_1,\ldots,E_n)$ and $\mathbf{D}(E_1,\ldots,E_n)$. Next lemma was proved in [7] for the purposes of transforming an SPG into a separable permutation. We reproduce it here with a proof, because all algorithms for a pattern matching are using the proof.

Lemma 1 (Decomposition Lemma [7])**.** *If G is an SPG with nodes $1,\ldots,n$ $(n>1)$ in canonical order, m is a least node such that $m \xrightarrow{1} T_1$ and l is a least node such that $l \xrightarrow{0} T_0$. If $l<m$ then G can be uniquely represented as $\mathbf{C}(G_1,\ldots,G_k)$ $(k>1)$ for some superpositional graphs $G_1,\ldots,G_k$. If $m<l$ then G can be uniquely represented as $\mathbf{D}(G_1,\ldots,G_k)$ $(k>1)$ for some superpositional graphs $G_1,\ldots,G_k$.*

Proof. Suppose $G=\mathbf{C}(E,F)$ with internal nodes $1,\ldots,n$ in canonical order. Let $i|i+1$ be the splitting line between E and F , i. e. internal nodes of E are $1,\ldots,i$ and internal nodes of F are $i+1,\ldots,n$ and $i \xrightarrow{1} i+1$. By definition of superposition, all edges $j \xrightarrow{1} T_1$ in E were redirected to node $i+1$ in G. It means, that only edges of the form $j \xrightarrow{0} T_0$ $(1 \le j < i)$ can overcome node $i+1$ in G. Similarly, if $G=\mathbf{D}(E,F)$, only edges $j \xrightarrow{1} T_1$ $(1 \le j < i)$ can overcome node $i+1$ in G.

Let G be a superpositional graph with internal nodes $1,\ldots,n$ in canonical order. Let l be the starting point of the leftmost 0-edge, pointing to T_0 and m be the starting point of the leftmost 1-edge, pointing to T_1.

1. Suppose $l<m$. Let $i: 1 \le i < m$ be the rightmost starting point (left from m) of the 0-edge, pointing to T_0. We claim, that $i|i+1$ is the rightmost splitting line of $G=\mathbf{C}(G_1,\ldots,G_k)$ between G_{k-1} and G_k. (Figure 6)

The edge $i \longrightarrow i+1$ must be a 1-edge because G is a binary graph (0-edge $i \xrightarrow{0} T_0$, starting from i already exists). There are some 0-edges $j \xrightarrow{0} T_0$ overcoming $i+1$ (at least one – $i \xrightarrow{0} T_0$) and no 1-edges $j \xrightarrow{1} T_1$, because leftmost such an edge is $m \xrightarrow{1} T_1$. We have to show, that there are no edges $r \longrightarrow s$, where $1 \le r < i$ and $i+1 < s \le n$. Suppose it is a 0-edge $r \xrightarrow{0} s$. Then it must cross

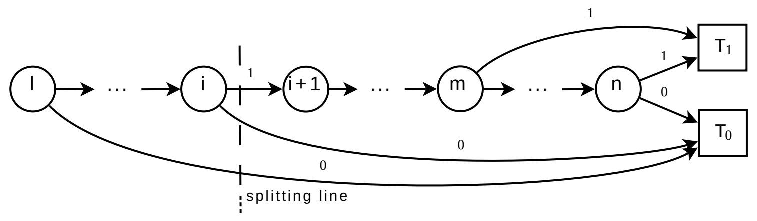

Figure 6: A rightmost splitting line.

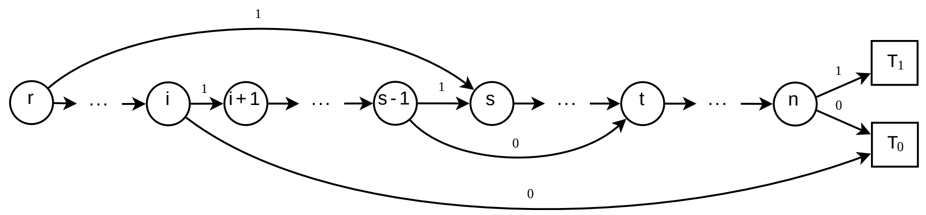

Figure 7: An 1-edge, overcoming $i+1$ yields non-cofinality.

the 0-edge $i \xrightarrow{0} T_0$, which violates the property of strong planarity. Suppose it is a 1-edge $r \xrightarrow{1} s$. Then $s \leq m$, otherwise it crosses $m \xrightarrow{1} T_1$. Due to homogeneity only 1-edges should enter s. Consequently $s-1 \xrightarrow{1} s$. We are dealing with binary graphs, therefore some 0-edge must start from $s-1$. It can not point to the terminal T_0, because $i \xrightarrow{0} T_0$ is the rightmost such an edge. If it is $s-1 \xrightarrow{0} t$, where $s < t \leq n$, then the 0-edges $i \xrightarrow{0} T_0$ and $s-1 \xrightarrow{0} t$ are starting between the endpoints of a 1-edge $r \xrightarrow{1} s$, which violates the property of cofinality (Figure 7). Consequently, G can be split into $G = \mathbf{C}(E,F)$ at the point $i|i+1$.

Superpositional graph F can not be split further using constructor $\mathbf{C}$, because inside F either $m = l$ (then F is a 1-node SPG A) or $m > l$, which is the case 2 of current proof.

We get the graph E, removing F from $\mathbf{C}(E,F)$ and redirecting edges, pointing to $i+1$ in $\mathbf{C}(E,F)$ into T_1. It means, that the starting point of the leftmost edge, pointing to T_1 in E, is the leftmost node k, for which $k \xrightarrow{1} i+1$ in G. Let node h in E be the rightmost node such that $h < k$ and $h \xrightarrow{0} T_0$, then $h < h+1$ is the

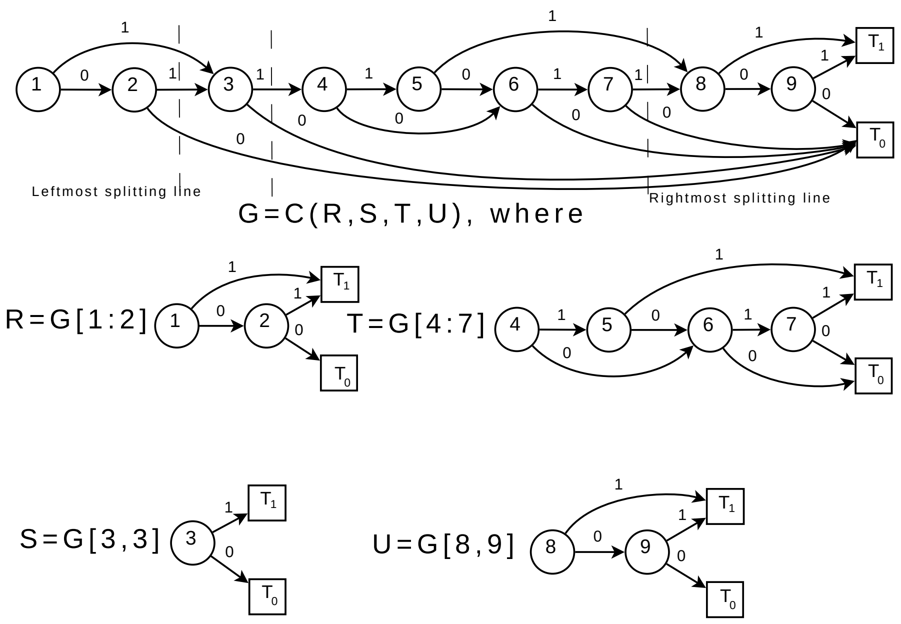

Figure 8: Decomposition of G into **C**(R,S,T,U).

rightmost splitting line for constructor **C** in E. Proceeding recursively until no such h exists, we receive a full decomposition of G by constructor **C**.

2. $m < l$. Dual to the previous case, gives a decomposition of G by constructor **D**.

□

If $G(1,\dots,n)$ can be decomposed into $\mathbf{C}(E,F)$ ($\mathbf{D}(E,F)$) using splitting line $i|i+1$, then we say that $type(G)$ is C (D), splitting point is i and designate the components E,F in terms of G by $E = G[1:i]$ and $F = G[i+1:n]$. See Figure 8 for an example of an decomposition of some SPG G of type C.

3 Pattern Matching for Separable Permutations

Definition 10. Let $[n] = \{1, \ldots, n\}$. A *permutation* p on the set $[n]$ is a bijection $p : [n] \to [n]$. We use a traditional notation: $p = p_1 \ \ldots \ p_n$, where $p_i = p(i)$. Let S_n be a set of all permutations on $[n]$.

An inverse of p is given by an equation $p^{-1}(p) = 12\ldots n$. The *pattern matching problem for permutations* is the following: Let $t \in S_n$ (the *text* and $p \in S_k,\ k \leq n$ (the *pattern*. The text t contains a pattern p or p matches into t, if there is a subsequence of t, say $t' = t_{i_1}, \ldots, t_{i_k}$, with $i_1 < i_2 < \ldots < i_k$, such that the elements of t' are ordered according to the permutation p – i.e. $t_{i_r} < t_{i_s}$ iff $p_r < p_s$.

If t does not contain such a subsequence, we will say that t is avoiding pattern p. Let $S_n(p)$ be the set of all n-permutations, avoiding p.

Definition 11. A *separable* n-permutation is a permutation, avoiding patterns 2413 and 3142, i.e. the class of permutations $S_n(2413, 3142)$.

Theorem 2 ([7])**.** *There is a bijection between a set of separable n-permutations and a set of superpositional graphs with n internal nodes.*

A proof of Theorem 2 was given in [7]. We will reproduce it here along with a couple of definitions, another theorem and an algorithm.

Definition 12. Let L be a set of labels and $l : [n] \to L$ a *labeling function*. A *labeled superpositional graph* is a pair $< G, f >$, where G is a SPG with n internal nodes and f is a labeling function. $f(i)$ assigns a label to node i in the canonical enumeration of internal nodes of G.

Definition 13. Let $\mathbf{C}$, $\mathbf{D}$ be two different labels. An *alternating tree* $T(\mathbf{C}, \mathbf{D})$ is an ordered tree with leaves $1\ldots, n$ (from left to right) whose leaves are labeled by a labeling function $l : [n] \to L$, where L is a set of labels, different from $\mathbf{C}$, $\mathbf{D}$. Internal nodes are labeled by labels from set $\{\mathbf{C}, \mathbf{D}\}$ so that the labels of the internal nodes are alternating in every path from root to leaf.

Theorem 3. *There is a bijection between alternating trees and labeled superpositional graphs.*

Proof. $\Longrightarrow$. Let T be an alternating tree in prefix form whose labeling function is l. To get a labeled SPG G_T, perform all superpositions, determined by constructors $\mathbf{C}$ and $\mathbf{D}$.
$\Longleftarrow$. Let G, l be a labeled SPG. Decompose G into $\mathbf{C}(G_1, \ldots, G_k)$ or $\mathbf{D}(G_1, \ldots, G_k)$

using the proof of Decomposition Lemma. Proceed recursively for $G_1, \ldots, G_k$ until single internal nodes.

It is easy to see, that both transformations are injections and reversing each other Therefore we have a needed bijection. An enumeration of leaves of T transforms into a canonical enumeration of internal nodes of G and vice versa. Therefore the labeling function remains the same.

□

Separable permutations can be sorted by Algorithm 1.

```
Algorithm 1. separatingsort (tree)
begin Traverse the tree in postorder.
    for every internal node i
    do if label(i)="-"
        then reverse the order of subtrees of node i
        fi
    od
end
```

After applying Algorithm 1 to the separating tree of permutation p we get the tree, whose labels of leaves are ordered and numbers of leaves (from left to right) are in order of p^{-1} (the revers of p). So, if we exchange the labels of the leaves and the numbers of the leaves in sorted tree, we get a separating tree for the permutation p^{-1}. If we apply the sorting algorithm to the sorted tree, we get back the original tree of the separable permutation.

On Figure 9 are depicted a separating tree of the permutation 85673412 and a sorted tree.

Now we are ready to show the proof of Theorem 2.

Proof. Every separable permutation $p = p_1, \ldots, p_n$ has an unique contracted separating tree $T_p(+,-)$ with labeling function $p(i) = p_i$. By Theorem 3 (taking $\mathbf{C} = +$, $\mathbf{D} = -$ and p as a labeling function) we have a bijection between sets $T_p(+,-)$ and labeled superpositional graphs. All we have to show is, that the information about the labeling function p is represented by the structure of SPG. Let G be an SPG. We add to G an identity labeling function $id(i) = i$ and build a contracted separating tree. Then we sort it using Algorithm 1, renumber leaves in ascending order and apply sorting algorithm once more. After exchanging labels and numbers of leaves, we get contracted separating tree for permutation p.

□

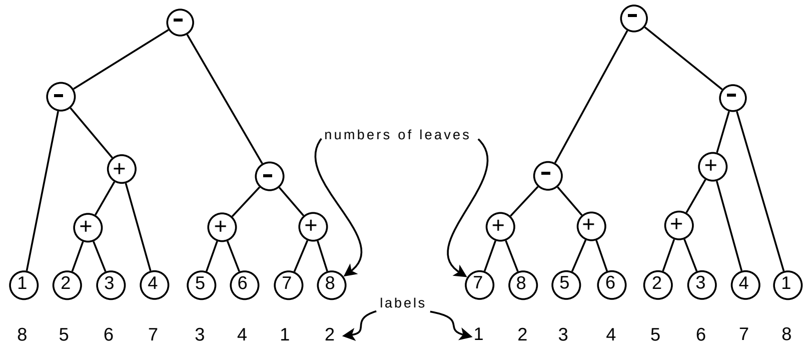

Figure 9: A separating tree (left) and the tree after sorting (right).

We present here the algorithms, implementing the bijection. Let G_p denote a superpositional graph, corresponding to a permutation p and p_G be a permutation, corresponding to a superpositional graph G.

```
Algorithm 2. sepperm2SPG(separable permutation $p = p_1 \dots p_n$)
//returns a superpositional graph $G_p$.
begin
    Augment the permutation to indices $T_0$ and $T_1$
    taking $p(T_0) = 0,\ p(T_1) = n+1$.
    Start with $n+2$ isolated nodes $1, \dots, n, T_1, T_0$;
    for $i := 1$ step 1 until $n-1$
    do if $p(i) < p(i+1)$
        then set $i \xrightarrow{1} i+1$; set $i \xrightarrow{0} j$, where $j \in \{i+2, \dots, n, T_0\}$
            is a least index for which $p(j) < p(i)$.
        else set $i \xrightarrow{0} i+1$; set $i \xrightarrow{1} j$, where $j \in \{i+2, \dots, n, T_1\}$
            is a least index for which $p(j) > p(i)$.
        fi
    od
    set $n \xrightarrow{1} T_1$; set $n \xrightarrow{0} T_0$
end
```

Let G be a superpositional graph with internal nodes $1, \dots, n$ in canonical order; let k, l be indexes: $1 \le k \le l \le n$. $\alpha || \beta$ denotes a concatenation of sequences of integers α and β.

Algorithm 3. **seq** (*superpositional graph* $G[k:l]$)

```
// returns a sequence of integers $i_1,\dots,i_l$
begin
    if $l = k$ then return $k$ fi;
    if $type(G) = C$
    then return $seq(G[k:split(G[k:l])])||seq(G[split(G[k:l])+1,l])$;
    else return $seq(G[split(G[k:l])+1,l])||seq(G[k:split(G[k:l])])$;
    fi;
end
```

Algorithm 4. **SPG2sepperm**(*superpositional graph* $G[1:n]$)

```
// returns a permutation $p_G$
begin
    return $inverse(seq(G[1:n]))$;
end
```

A function $inverse(p)$ calculates an inverse of the permutation p.

Definition 14. The *pattern matching problem for superpositional graphs* is the following: Let T (text) and P (pattern) be superpositional graphs with internal nodes $1,\dots,n$ and $1,\dots,k$ ($k \le n$). We say, that P matches into T if there exists a sequence of integers $i_1,\dots,i_k$ such that:

1. For every arrow $l \xrightarrow{1} T1$ in P there exists a 1-path $i_l \rightsquigarrow T1$ in T, which consists of nodes from the set $\{i_l, i_l+1,\dots,i_{l+1}-1\}$.

2. For every arrow $l \xrightarrow{0} T0$ in P there exists a 0-path $i_l \rightsquigarrow T0$ in T, which consists of nodes from the set $\{i_l, i_l+1\dots,i_{l+1}-1\}$.

3. For every arrow $l \xrightarrow{1} m$ ($m \le k$) in P there exists a 1-path $i_l \rightsquigarrow i_m$ or there are indexes $r,s:\ r < i_m < s$ such that there exists a 1-path $i_l \rightsquigarrow r$ and $r-1 \xrightarrow{0} s$ in T.

4. For every arrow $l \xrightarrow{0} m$ ($m \le k$) in P there exists a 0-path $i_l \rightsquigarrow i_m$ or there are indexes $r,s:\ r < i_m < s$ such that there exists a 0-path $i_l \rightsquigarrow r$ and $r-1 \xrightarrow{1} s$ in T.

Lemma 2. *Let $t = t_1 \dots t_n$ be a separable permutation and G_t its SPG, built using Algorithm 2. Let $1 \le l < m \le n$. Then:*

1. $t(l) < t(m)$ iff there is a 1-path $l \rightsquigarrow m$ in G_t or there exist r,s: $l < r < m < s$ such that there is a 1-path $l \rightsquigarrow r$ and $r-1 \xrightarrow{0} s$ in G_t.

2. $t(l) > t(m)$ iff there is a 0-path $l \rightsquigarrow m$ in G_t or there exist r,s: $l < r < m < s$ such that there is a 0-path $l \rightsquigarrow r$ and $r-1 \xrightarrow{1} s$ in G_t.

Proof. We prove the first assertion, the proof of the second assertion is dual.

1a. ($\Leftarrow$). If there is a 1-path $l \rightsquigarrow m$ in G_t, then by Algorithm 2 $t(l) < t(m)$. Let $l \rightsquigarrow r$ be a 1-path and $r-1 \xrightarrow{0} s$ in G_t, where $r < m < s$. We show, that if under these conditions $t(l) > t(m)$ then t is not a separable permutation. If $t(l) > t(m)$, then $low(l) < r$, otherwise $low(r-1) = s \leq low(l)$ (strong planarity of G_t), $m < low(l)$ and by Algorithm 2 $t(l) < t(m)$. We have $t(m) > t(r-1)$, otherwise $low(r-1) = m$ instead of $low(r-1) = s$. Also we know, that $t(l) < t(r)$. If $t(m) < t(l)$, then we have four indices $l < r-1 < r < m$ and $t(r-1) < t(m) < t(l) < t(r)$, which is a forbidden subsequence for a separable permutation.

1b. ($\Rightarrow$). Let $t(l) < t(m)$ for some l,m: $1 \leq l < m \leq n$. We show, that every attempt to find m, which does not satisfy the conditions ends up with the subsequence of indexes, matching forbidden pattern 2413, i.e. t is not a separable permutation. Let $r < m$ be greatest index such that there is a 1-path $l \rightsquigarrow r$. $t(r) > t(m)$, otherwise $r \xrightarrow{1} m$ and, consequently, we have a 1-path $l \rightsquigarrow m$. r must be greater than $low(l)$, otherwise we have a forbidden subsequence $t(l), t(r), t(low(l)), t(m)$. Let $h \leq r$ be a maximal index, such that $p(h) > p(l)$ and $p(h-1) < p(l)$ (such node h always exists, because $low(l) < r$, $p(low(l)) < p(l)$ and $p(r) > p(l)$). There exists a 1-path $l \rightsquigarrow p(h)$, otherwise there must be a node $k:\ l < k < h-1$ such that $p(k) > p(h)$ and $p(l), p(k), p(h-1), p(h)$ is a forbidden subsequence. Let $h-1 \xrightarrow{0} s$. Due to the construction we have $s > r$. If $s < m$, then $p(l), p(r), p(s), p(m)$ is a forbidden pattern. If $s > m$, then there is a 1-path $l \rightsquigarrow h$ and $h-1 \xrightarrow{0} s$, which means that the conditions of the lemma are fulfilled for m. □

Theorem 4. *A separable permutation p matches into a separable permutation t iff G_p matches into G_t.*

Proof. $\Longrightarrow$. Let p matches into t. Then there exists a sequence $i_1, \ldots, i_k$ such that $t_{i_l} < t_{i_m}$ iff $p(l) < p(m)$. Match of G_p into G_t is a subgraph of G_t, induced by nodes $i_1, \ldots, i_k$ (according to the Definition 14).

$\Longleftarrow$. Let G_p matches into G_t, i. e there exists a subsequence of nodes $i_1, \ldots, i_k$ in G_t, which determines a match. The same subsequence is a match of p into t.

The conditions of matching are fulfilled in both directions due to Lemma 2.

□

Pattern matching for separable permutations can thus be reduced to SPG pattern matching in linear time.

4 Pattern Matching for Superpositional Graphs

We need some preliminary denotations for presenting an algorithm for pattern matching. We denote by $G[k:l]$ a subgraph of G, induced by nodes $k, k+1, \ldots, l, T1, T0$ in which every edge $i \xrightarrow{1} m$ ($i \xrightarrow{0} m$) for $m > l$ is redirected to $T1$ ($T0$) If A and B are sets of sequences of integers, then $A \cup B$ denotes a union and $A \times B$ a Cartesian product of A and B. Note, that $A \times \emptyset = \emptyset \times A = \emptyset$; $\{r\ r+1\ \ldots\ s\}$ is a set which consists of a single sequence $r\ r+1\ \ldots\ s$ and $\{r, r+1, \ldots, s\}$ consists of $s-r+1$ sequences, each of length 1. Variables X, Y, Z, V, W in Algorithm 5 are local variables of type *set of integer sequences*. A function *equivalent* checks if its arguments are equivalent up to the labels of internal nodes and function $split(G)$ returns a leftmost splitting point of G.

```
**Algorithm 5.** **match**$(T[r:s], P[u:v])$
//returns a set of integer sequences, which are matches of SPG $P[u:v]$ into SPG $T[r:s]$.
begin
    if $s-r < v-u$ then return $\emptyset$ fi;
    if $s-r = v-u$
    then if $equivalent(T[r:s], P[u:v])$
        then return $\{r\ r+1\ \ldots\ s\}$
        else return $\emptyset$
        fi
    fi;
    if $u = v$ then return $\{r, r+1, \ldots, s\}$ fi;
    $dt = split(T[r:s])$; $dp = split(P[u:v])$;
    $X := match(T[r:dt], P[u:v])$; //all matches of $P[u:v]$ in the left part.
    $Y := match(T[dt+1, s], P[u:v])$; //all matches of $P[u:v]$ in the right part.
    $Z := X \cup Y$
    while $type(T[r:s]) = type(P[u:v])$ & $dt - r \geq dp - u$
    //a cycle over splitting points of $P[u:v]$.
    do
      if $s - dt \geq u - dp$ then
        $V := match(T[r:dt], P[u:dp)$;
```

```
            $W := match(T[dt+1,s], P[dp+1:v])$;
            $Z := Z \cup (V \times W)$;
        fi;
        $dp := split(P[dp+1:v])$ :
    od
    return $Z$;
end
```

To prove the correctness of Algorithm 5 we need two lemmas first.

Lemma 3. *Let G be an SPG, l be a splitting point of G and $1 \le u < l < v \le n$. If G is of type C then:*

(a) There does not exist a 1-path $u \overset{1}{\rightsquigarrow} v$;

(b) Every 0-path $u \overset{0}{\rightsquigarrow} v$ contains a node $l+1$.

If G is of type D then:

(c) There does not exist a 0-path $u \overset{0}{\rightsquigarrow} v$;

(d) Every 1-path $u \overset{1}{\rightsquigarrow} v$ contains a node $l+1$.

Proof. Follows immediately from Decomposition Lemma. □

Lemma 4. *1. If pattern P is of type C and text $T = \mathbf{D}(T_1,\dots,T_h)$, where $T_1,\dots,T_h$ are of type C then every match of P into T lies entirely inside some T_i $(1 \le i \le h)$.*

2. If pattern P is of type D and text $T = \mathbf{C}(T_1,\dots,T_h)$, where $T_1,\dots,T_h$ are of type D then every match of P into T lies entirely inside some T_i $(1 \le i \le h)$.

Proof. We prove 1., the proof of 2. is dual.

Let $(i_1,\dots,i_k)$ be a match of P into T. Let us assume on the contrary, that there is m $(1 \le m < k)$ such that i_m is a node of T_r and i_{m+1} is a node of $\mathbf{D}(T_{r+1},\dots,T_h)$. Due to Decomposition Lemma there must be a splitting point l of type D between nodes i_m and i_{m+1} that splits T into $\mathbf{D}(T_1,\dots,T_r)$ and $\mathbf{D}(T_{r+1},\dots,T_h)$. By Decomposition Lemma we have $l \xrightarrow{0} l+1$, $l \overset{1}{\rightsquigarrow} T1$ and only edges $j \xrightarrow{1} T1$ can overcome node $l+1$. Suppose $m \xrightarrow{1} m+1$ in P. By the definition of matching $i_m \overset{1}{\rightsquigarrow} i_{m+1}$, which violates (a) of Lemma 3, or $i_m \overset{1}{\rightsquigarrow} u$ and $u \xrightarrow{1} v$, where $u < i_{m+1} < v$. Node u must be left from $l+1$, otherwise we had a 1-path overcoming $l+1$. Then an edge $u \xrightarrow{1} v$ overcomes a splitting point. which is also impossible. Suppose $m \xrightarrow{0} m+1$. As P is a binary graph, there must be a 1-edge

$m \xrightarrow{1} v$, where $m+1 < v$. If $v \leq n$, then the existence of a path $i_m \overset{1}{\rightsquigarrow} v$ violates (a) of Lemma 3. If $v = T1$, then there must exist a node $u < m$ such that $u \xrightarrow{0} T0$, otherwise P is not of type C. By definition of pattern matching there must be a path $i_u \xrightarrow{0} T0$ which does not contain any node from match. As $m \xrightarrow{0} m+1$, there must be also a path $i_m \overset{0}{\rightsquigarrow} i_{m+1}$ By (b) of Lemma 3, both paths have a common node $l+1$ and, therefore all nodes from $l+1$ to i_{m+1} are common. It means, that the path $i_u \xrightarrow{0} T0$ contains a node $i_{m+1}$from the pattern, which violates the definition. This is a contradiction.

□

Theorem 5. *Algorithm 5 is correct.*

Proof. Let $T(1,\dots,n)$ (text) and $P(1,\dots,k)$ (pattern) be two superpositional graphs. We have to show, that there exists a sequence of integers $i_1, i_2, \dots, i_k$ $(1 \leq i_1 < i_2 < \dots < i_k \leq n)$ such that conditions of Definition 14 are fulfilled if and only if $(i_1, i_2, \dots, i_k) \in match(T[1:n], P[1:k])$.

1. $(\Rightarrow)$. Let $(i_1, i_2, \dots, i_k)$ fulfill the conditions of Definition 14.

We prove by induction on k, that then $(i_1, i_2, \dots, i_k) \in match(T[1:n], P[1:k])$.

The case $k = 1$ is obvious. By Lemma 4 we can assume, that $(i_1, i_2, \dots, i_p)$ lies entirely in some (minimal) subgraph $T[r,s]$. Let the type of $T[r,s]$ and $P[1:k]$ be C (case D is dual) and $split(T[r,s] = l$. Let m $(1 \leq m < k)$ be an index such that nodes $i_1, \dots, i_m$ are nodes of $T[r,l]$ and nodes $i_{m+1}, \dots, i_p$ are nodes of $T[l+1,s]$. By induction hypothesis $(i_1, \dots, i_m) \in match(T[r:l], P[1:m])$ and $(i_{m+1}, \dots, i_k) \in match(T[l+1:s], P[m+1:k])$. We have to show, that m is some splitting point of P. The conditions for m to be an splitting point of type C are: (a) $m \xrightarrow{1} m+1$; (b) $m \xrightarrow{0} T0$; (c) There does not exist an 1-edge $u \xrightarrow{1} v$, where $u < m < m+1 < v$. All the conditions can be checked by considerations, similar to the proof of Lemma 4. According to Algorithm 5 $(i_1, i_2, \dots, i_k) \in match(T[1:l], P[1:m]) \times match(T[l+1,n], P[m+1,k]) \subseteq match(T[1:n], P[1:k])$.

2. $(\Leftarrow)$. Let $(i_1, i_2, \dots, i_p) \in match(T[1:n], P[1:m])$.

We have to show, that the conditions of Definition 14 are fulfilled. Let $m \in \{1,\dots,p\}$ and $m \xrightarrow{1} m+1$ in P (case $m \xrightarrow{0} m+1$ is dual. Indexes i_m and i_{m+1} can be adjacent in a sequence $(i_1, i_2, \dots, i_p) \in match(T[1:n], P[1:m])$ in two cases.

a) There are subgraphs $T[r:s]$ and $P[u:v]$ which are equivalent. In this case $i_m \xrightarrow{1} i_{m+1}$ follows immediately.

b) $P[u:v]$ is a maximal subgraph of $P[1:k]$ whose split point is m, $T[r:s]$ is a subgraph of $T[1:n]$, whose leftmost split point is l and $(i_i, \dots, i_k) \in match(T[r:$

$l], P[u:m]) \times match(T[l+1:s], P[m+1:v])$. By Theorem 3 in [4] there exists a 1-path from every internal node of an SPG into $T1$. Applying this result to $T[r:l]$ we can claim, that there exists a 1-path $i_m \overset{1}{\rightsquigarrow} T1$. According to a definition of superposition this path transforms into $i_m \overset{1}{\rightsquigarrow} l+1$ in $T[r:s] = \mathbf{C}(T[r:l], T[l+1,s]$. IF $i_{m+1} = l+1$ then the first part of condition 3 of the Definition 14 is fulfilled. If $i_{m+1} < l+1$, then we have $i_m \overset{1}{\rightsquigarrow} l+1$ and $l \overset{0}{\rightsquigarrow} T0$ (again because $T[r:s] = \mathbf{C}(T[r:l], T[l+1,s])$ and the second part of condition 3 is fulfilled.

□

An obvious modification of Algorithm 5 counts the number of matches. Variables X, Y, Z, V, W are of type *integer*.

Algorithm 6. **#match**$(T[r:s], P[u:v])$

```
//returns a number of matches of SPG $P[u:v]$ into SPG $T[r:s]$.
begin
    if $s-r < v-u$ then return 0 fi;
    if $s-r = v-u$
    then if $equivalent(T[r:s], P[u:v])$
        then return 1
        else return 0
        fi;
    fi;
    if $u = v$ then return $s-r+1$ fi;
    $dt = split(T[r:s])$; $dp = split(P[u:v])$;
    $X := \#match(T[r:dt], P[u:v])$;
    $Y := \#match(T[dt+1,s], P[u:v])$;
    $Z := X + Y$
    while $type(T[r:s]) = type(P[u:v])$ & $dt - r \geq dp - u$
    do
      if $s - dt \geq u - dp$ then
        $V := \#match(T[r:dt], P[u:dp)$;
        $W := \#match(T[dt+1,s], P[dp+1:v])$;
        $Z := Z + (V \cdot W)$;
      fi;
      $dp := split(P[dp+1:v]) :$
    od
    return $Z$;
```

```
end
```

5 Performance

Obviously, the performance of Algorithm 5 equals to the performance of Algorithm 6 plus the number of matches. Therefore we concentrate on estimating of the performance of Algorithm 6.

We represent a superpositional graph $G(1,\dots,n)$ by a two-dimensional array $G[0:1,1:n]$, where $G[0,i]=j$ iff there is a 0-edge $i \xrightarrow{0} j$ and $G[1,i]=j$ iff there is a 1-edge $i \xrightarrow{1} j$ (both terminal nodes are designated by $n+1$). If we are dealing with a subgraph $G[r:s]$, then we can extract it from an array G just by indexes r,s. So, there is no need to duplicate subgraphs for recursive calls.

Function $\mathit{equivalent}(T[r:s],P[u:v])$, where $T[r:s]$ and $P[u:v]$ are SPG-s of equal length is obviously linear in the length. Function split can be in advance calculated for every subgraph, which occurs in recursive decomposition of the SPG and the results can be stored in a two-dimensional array $S[1:n,1:n]$. As our algorithm only splits graph into subgraphs and does not contain "joins", every line $i|i+1$ $(1 \le i < n)$ is used exactly once. Therefore we need to use only $n-1$ elements of an array $S[1:n,1:n]$. Algorithm 7 calculates splitting points for all subgraphs $G[k,l]$, needed for the full decomposition of G.

```
Algorithm 7. read(k,m)
//uses global arrays $G[0:1,1:n]$, representing an SPG and $S[1:n,1:n]$ for storing the splitting points.
begin
    i:=k; //current node.
    t:=0; //0, if current long edge is 0-edge, 1 otherwise.
    if $G[1,k] > k+1$ then $t := 1$ fi; //if first 1-edge is longer.
    while $i < m$
    do $r := \min\{G[t,i]-1,m\}$;//r is an endpoint of a subgraph
        $S[k,r] := i$;
        if $r-i > 1$ then $read(i+1,r)$ fi; //read nodes under the current edge.
        $i := r$;
        $t := XOR(t,1)$; //switches between 1 and 0.
    od
```

end

Before we prove the correctness of algorithm 7, we need to define split points for superpositional graph G.

Definition 15. Split points are points $i|i+1$ for superpositional graph G, where internal nodes $1,\ldots,n$ of G are in canonical order and G is a superposition $\mathbf{C}(E,F)$ or $\mathbf{D}(E,F)$, where internal nodes of E and F are $1,\ldots,i$ and $i+1,\ldots,n$ respectively.

Next we will state a lemma that will help us to prove the correctness of algorithm 7.

Lemma 5. *$i|i+1$ is a split point for superpositional graph G if and only if all the edges of G of one type (0- or 1-edges) crossing the split point direct to $i+1$ and all the edges of G of the opposite type crossing the split point direct to a terminal node.*

Proof. Let G be a superposition $\mathbf{C}(E,F)$ and $i|i+1$ its split point. According to the definition of superposition, all edges of E pointing to T_1 are redirected to $i+1$. Since the only 1-edges crossing the split point are the redirected edges, which now point to $i+1$, and the only 0-edges crossing it are pointing to T_0, the condition is satisfied. The case for superposition $\mathbf{D}(E,F)$ is analogous.

It is also easy to see that when the all the edges of one type crossing $i|i+1$ point to $i+1$ and all the edges of opposite type point to a terminal node, then the superpositional graph G with internal nodes $1,\ldots,n$ can be decomposed into graphs E and F with internal nodes $1,\ldots,i$ and $i+1,\ldots,n$ respectively, using either constructor $\mathbf{C}(E,F)$ or $\mathbf{D}(E,F)$.

□

Theorem 6. *Algorithm 7 finds the split points for superpositional graph G, its subgraphs E and G, and recursively for all their subgraphs down to elementary graphs A (which consist of just one node and thus cannot be decomposed further).*

Proof. The algorithm searches for the leftmost point $i|i+1$ that satisfies the conditions of lemma 5. Since there are exactly two edges leaving any internal node and they cannot point to the same internal node, the node i we are looking for must have one of its edges point to a terminal node. Algorithm starts its search from node 1 (let us call this node d_1). If high(1) or low(1) points to a terminal node, then $i = 1$ satisfies conditions of lemma 5 and obviously it is the leftmost such node.

Let us assume that $high(1)$ and $low(1)$ both point to internal nodes of G. Algorithm finds the longer of the two edges (let us denote this by $long(1)$) and records the endpoint of it. Let us call the node just left of it d_2. It is clear that none of the nodes left of d_2 can satisfy the conditions of lemma 5. However, node d_1 is the split point for a subgraph E_1 consisting of nodes $1,\dots,d_2$ for the reasons given in the previous paragraph. The split point allows us to split E_1 into its subgraphs E_0 and F_0 whereas we already know that E_0 has just one node and thus cannot be split further. If F_0 has more than one node then the algorithm continues to find its split point and subgraphs until it reaches elementary graphs.

Once the algorithm has found a splitpoint d_i for a subgraph E_i consisting of nodes $1,\dots,d_{i+1}$, the next split point is $d_{i+1} = long(d_i) - 1$, unless $long(d_i)$ is a terminal node in which case $E_i = G$.

Edges crossing the split point $d_i|d_i + 1$ direct to exactly two nodes. We can show that recursively. We have already shown that it is so for $i = 1$. Now, if this is true for d_{i-1}, then one of the nodes edges coming from left of d_{i-1} direct to is the node $d_{i-1} + 1$ and the other one is the node $d_i + 1$. All the edges of the same type as the edge long(d_{i-1}) coming from the nodes $d_{i-1} + 1,\dots,d_i$ and crossing $d_i|d_i + 1$ have to direct to $d_i + 1$ because of the strong planarity requirement. All the edges of the opposite type coming from the nodes $d_{i-1} + 1,\dots,d_i$ and crossing $d_i|d_i + 1$ have to direct to $long(d_i)$ because of the cofinality requirement.

If $long(d_i)$ is a terminal node, then we have found the split point for G, else the algorithm records d_i as a split point for a subgraph and continues.

$\square$

Since there is no overlap between subgraphs E and F, all split points are only used once by the algorithm and there are exactly n - 1 split points in total. The number of steps for Algorithm 7 is thus obviously linear in the length of an SPG. The algorithm moves from the first node of an SPG to the end using long edges, until one finds a long edge, pointing out of the limits. As shown in the proof, this is a leftmost splitting point.

The decomposition type of a subgraph $G[k,m]$ is C if a short edge from $split(G[k,m])$ to the next node is a 1-node and D otherwise.

Algorithm 6 makes multiple recursive calls with the same text and pattern in some cases.

Example. Let us look for number of matches of $P = (4,4,4)(2,3,4)$ into

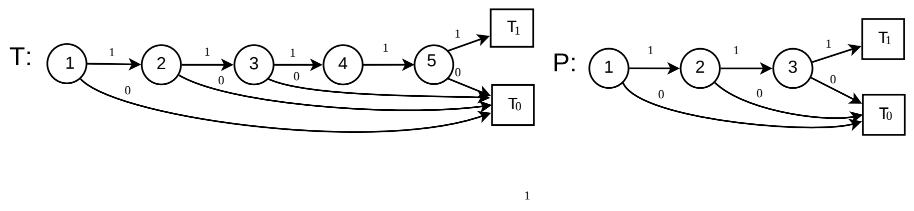

Figure 10: Every subsequence of $(1,2,3,4,5)$ of length 3 is a match of P into T, so there are $\binom{5}{3} = 10$ matches.

$T = (6,6,6,6,6)(2,3,4,5,6)$ (see Figure 10). Then algorithm **read** gives us

$$SP[1,3] = 1, SP[2,3] = 2,$$

$$ST[1,5] = 1, ST[2,5] = 2, ST[3,5] = 3, ST[4,5] = 4$$

and

$$\begin{aligned}
& \#match(T[1:5], P[1:3]) \\
= \; & \#match(T[1:1], P[1:3]) + \#match(T[2:5], P[1:3]) \\
& + \#match(T[1:1], P[1:1]) * \#match(T[2:5], P[2:3]) \\
= \; & 0 + \#match(T[2:2], P[1:3]) + \#match(T[3:5], P[1:3]) \\
& + \#match(T[2:2], P[1:1]) * \underline{\#match(T[3:5], P[2:3])} \\
& + 1 * (\#match(T[2:2], P[2:3]) + \underline{\#match(T[3:5], P[2:3])} \\
& + \#match(T[2:2], P[2:2]) * \#match(T[3:5], P[3:3])) \\
= \; & 0 + (0 + 1 + 1 * (\#match(T[3:3], P[2:3]) + \#match(T[4:5], P[2:3]) \\
& + \#match(T[3:3], P[2:2]) * \#match(T[4:5], P[3:3]))) \\
& + 1 * (0 + (\#match(T[3:3], P[2:3]) + \#match(T[4:5], P[2:3]) \\
& + \#match(T[3:3], P[2:2]) * \#match(T[4:5], P[3:3]) + 1 * 3)) \\
= \; & 0 + (0 + 1 + 1 * (0 + 1 + 1 * 2)) + 1 * (0 + (0 + 1 + 1 * 2) + 1 * 3) \\
= \; & 10
\end{aligned}$$

In this example $\#match(T[3:5], P[2:3])$ had to be calculated twice.

To avoid multiple calls we have to store the number of matches for every combination of text and pattern. There are $n-1$ splitting points in the text and $p-$

1 splitting points in the pattern, so we need a two-dimensional array $COUNT[1 : n-1, 1 : k-1]$. We assume, that we have prepared global arrays $T[0:1, 1:n]$ for a text, $ST[1:n-1, 1:n]$ for splitting points of the text, $P[0:1, 1:k]$ for a pattern, $SP[1:k-1, 1:k]$ for splitting points of the pattern and $COUNT[1:n-1, 1:k-1]$, filled in with constants -1.

Algorithm 8. count(r, s, u, v)

```
//returns a number of matches of SPG $P[u:v]$ into SPG $T[r:s]$ .
begin
        $dt = ST[r,s];\ dp = SP[u,v]);$
        if $COUNT[dt,dp] \neq -1$ then return $COUNT[dt,dp]$
        if $s-r < v-u$ then $COUNT[dt,dp] := 0$; return 0 fi;
        if $s-r = v-u$
        then if $equivalent(T[r:s], P[u:v])$
                then $COUNT[dt,dp] := 1$; return 1
                else $COUNT[dt,dp] := 0$; return 0
                fi;
        fi;
        if $u = v$ then $COUNT[dt,dp] := s-r+1$; return $s-r+1$ fi;
        $X := count(T[r:dt], P[u:v]);$
        $Y := count(T[dt+1,s], P[u:v]);$
        $Z := X+Y$
        while $type(T[r:s]) = type(P[u:v])$ & $dt-r \geq dp-u$
        do
            if $s-dt \geq u-dp$ then
                $V := count(T[r:dt], P[u:dp);$
                $W := count(T[dt+1,s], P[dp+1:v]);$
                $Z := Z + (V \cdot W);$
            fi;
            $dp := split(P[dp+1:v]):$
        od
        $COUNT[dt,dp] := Z$; return $Z$;
end
```

It is easy to see, that it takes $O(kn)$ steps to compute the number of matchings. An obvious modification of Algorithm 8 allows us to compute all matches in time $O(kn+p)$, where p is a number of matches.

6 Conclusion

We can conclude, that superpositional graphs are useful formalism for a pattern matching problem for separable permutations. For investigating more general problems – if text is a Baxter permutation or a general permutation – we need a bijection between permutations and homogeneous binary graphs, which gives strongly planar binary graph, if limited to Baxter permutations and agrees with the bijection, defined by Algorithms 2 and 4, if limited to separable permutations.

7 Kokkuvõte

7.1 Mustrite leidmine superpositsioonigraafides

Käesoleva magistritöö eesmärgiks on leida võimalikult kiire algoritm mustrite leidmiseks superpositsioonisgraafides.

Superpositsioonigraafi (SPG) mõistet kasutati esmakordselt artiklis [3] sktruktuurselt sünteesitud binaarsete otsustusdiagrammide skeleti kohta. Artiklis [7] defineeriti mustrite leidmine superpositsioonisgraafides ja leiti ka bijektsioon SPGde ja lahutatavate permutsioonide vahel. Mati Tombak, käesoleva magistritöö juhendaja, oli tõestanud veel avaldamata töös, et mustrite leidmine superpositsioonisgraafides ja lahutatavates permutsioonides on samaväärne ning mustrite leidmine lahutatavates permutsioonides on taandatav lineaarse ajaga mustrite leidmisele superpositsioonisgraafides.

Olgu n teksti pikkus ja k mustri pikkus. P. Bose, J. Buss and A. Lubiw näitasid artiklis [1], et üldine mustrite leidmise probleem on $NP-$täielik, aga loendamise probleem on lahenduv $O(kn^6)$ ajaga juhul kui muster on lahutatav permutsioon. L. Ibarra näitas artiklis [2], et see on teostatav ajaga $O(kn^4)$.

Käesolevas magistritöös leitakse algoritm kiirusega $O(kn)$ mustrite leidmiseks superpositsioonisgraafides. Me näitame ka, et iga lahendus mustri leidmiseks superpositsioonisgraafides on ka lahenduseks sellelevastava mustri leidmiseks lahutatavates permutsioonides ja vastupidi. Tulemuseks saame algoritmi, mis loendab mustri sobivusi ajaga $O(kn)$, mustrite leidmiseks juhul kui nii tekst kui ka muster on lahutatavad permutsioonid.

Peatükis 2 defineerime superpositsioonisgraafid ja näitame SPGde konstrueerimist ja lahutamist elementaargraafideks. SPGde lahutamisel on oluline osa SPGde mustrite leidmises. Peatükis 3 defineerime lahutatavad permutatsioonid ja esitame algoritmid SPGde ja lahutatavate permutatsioonide bijektsiooni jaoks. Peatükis 4 anname SPGde mustrite leidmise algoritmi ja esitame tõestuse selle õigsuse kohta. Peatükis 5 esitame paar abialgoritmi ja määrame mustrite leidmise algoritmi töökiiruse. Autori panus antud töös on SPGde mustrite leidmise algoritmid ja töökiiruse arvutused (peatükid 4 ja 5, tihedas koostöös magistritöö juhendajaga).

Käesoleva magistritöö sisu on esitatud mustrite leidmise konverentsile DAGM-OAGM 2012, Grazis, Austrias (lisa A). Vastust töö vastuvõtmise kohta ootame 18. juuniks 2012.

References

[1] Bose, P., Buss, P.J., Lubiw, A.; Pattern Matching for Permutations. Information Processing Letters, 65, 277–283 (1998).

[2] Ibarra, L.; Finding Pattern Matchings for Permutations. Information Processing Letters, 61, 293-295 (1997).

[3] Jutman, A., Peder, A., Raik, J., Tombak, M., Ubar, R.; Structurally Synthesized Binary Decision Diagrams. 6th International Workshop on Boolean Problems, Freiberg University, 271–278 (2004).

[4] Peder, A., Tombak, M.; Superpositional graphs. Acta et Commentationes Universitatis Tartuensis de Mathematica, 13, 51–64 (2009).

[5] Stankovic, R.S., Ubar, R., Astola, J.T; Decision Diagrams: From a Mathematical Notion to Engineering Applications. Facta Universitatis - series: Electronics and Energetics, Niš, 2011 24(3):281–301.

[6] Ubar, R.; Test Generation for Digital Circuits Using Alternative Graphs. (in Russian). Proc. Tallinn Technical University, 409, Tallinn, Estonia, 75–81 (1976).

[7] Vohandu, L., Peder, A., Tombak, M.; Permutations and Bijections. Information Modelling and Knowledge Bases XXIII, IOS Press, 419–437 (2012).

A Paper submitted to DAGM-OAGM 2012