Pattern Matching for Superpositional Graphs

Anonymous DAGM-OAGM submission

Paper ID ***

Abstract. This paper presents a pattern matching algorithm for superpositional graphs which counts a number of matches with time complexity $\mathcal{O}(kn)$, where n is a length of a text and k is a length of a pattern. As a consequence, the same time complexity is achieved for the case, when both text and pattern are separable permutations.

Keywords: pattern matching, superpositional graph, separable permutation.

1 Introduction

Superpositional graphs (SPG) were defined in [3] as a skeleton of structurally synthesized binary decision diagrams. Vohandu et al. introduced a problem of pattern matching for SPG in [5]. 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 $\mathcal{O}(kn^6)$ time in the case, if the pattern is a separable permutation. This result was improved by L. Ibarra in [2] to $\mathcal{O}(kn^4)$.

In current paper we modify slightly a definition of pattern matchig for SPG and build an algorithm for pattern matching for SPG, which counts a number of matchings in time $\mathcal{O}(kn)$. We prove, that every solution of the problem for SPG is a solution of a pattern matching problem for corresponding separable permutations and vice versa. As a consequence, we get an algorithm for counting matches, working in time $\mathcal{O}(kn)$, for the case when the text and the pattern are both separable permutations.

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 $a \to b$ 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* $u \overset{0}{\rightsquigarrow} v$ (1-*path* $u \overset{1}{\rightsquigarrow} v$) is a path which contains only 0-edges (1-edges).

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

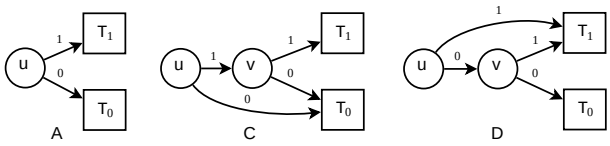

Fig. 1. Binary graphs A, C and D

Definition 2. *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. 1.

Definition 3. *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 $D = A_{v\leftarrow D} \in SPG$.

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. Vohandu et al. in [5] showed that constructors of superpositional graphs $\mathbf{C}$ and $\mathbf{D}$ are associative, so it is legal to use "long" constructors $\mathbf{C}(E_1, \ldots, E_n)$ and $\mathbf{D}(E_1, \ldots, E_n)$. The next sections of this paper need a graph-theoretical description of the superpositional graph.

Definition 4. *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. We are interested in traceable binary graphs only. Therefore we 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 5. *A binary graph G is* homogenous *if only one type of edges (i.e. either 1-edges only or 0-edges only) enters into every node $v \in V(G)$.*

Definition 6. *A binary traceable graph is* strongly planar *if it has no crossing 0-edges and no crossing 1-edges in its stretched drawing.*

Definition 7. *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.*

Definition 8. *A binary traceable graph is* cofinal *if it is both* 1*-cofinal and* 0*-cofinal.*

Theorem 1 ([4]). *A binary graph G is a superpositional graph if and only if G is traceable, homogenous, strongly planar and cofinal.*

Vohandu et al. proved in [5] a Decomposition Lemma, which is needed later.

Lemma 1 (Decomposition Lemma [5]). *If G is an SPG with nodes $1,\dots,n$ $(n>1)$ 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 $\boldsymbol{C}(G_1,\dots,G_k)$ $(k>1)$ for some superpositional graphs $G_1,\dots,G_k$. If $m<l$ then G can be uniquely represented as $\boldsymbol{D}(G_1,\dots,G_k)$ $(k>1)$ for some superpositional graphs $G_1,\dots,G_k$.*

If $G(1,\dots,n)$ can be decomposed into $\mathbf{C}(E,F)$ ($\mathbf{D}(E,F)$) then we say, that a *splitting point* of G between E and F is a last internal node of E and a type of G is C (D). If E is of type D, then the splitting point between E and F is a *leftmost splitting point* of G.

3 Pattern Matching for Superpositional Graphs

Definition 9. *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.

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 $k,k+1,\dots,l,T1,T0$ 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 1 are local variables of type *set of integer sequences*. Function *equivalent* checks if his arguments are equivalent up to the labels of internal nodes and function $split(G)$ returns a leftmost splitting point of G.

Algorithm 1. 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 1 we need two lemmas first. We omit the proofs of Lemmas 2, 3 and Theorem 3 due to the lack of space. All these claims can be proven by quite straightforward, but tedious case analysis, using the properties of SPG from Theorem 1 and Decomposition Lemma.

Lemma 2. *Let* G *be an SPG,* l *be a splitting point of* G *and* $1 \leq u < l < v \leq n$. *If* G *is of type D 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 C 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 $l+1$.

Lemma 3. *1. If pattern P is of type C and text $T = \mathbf{D}(T_1, \ldots, T_h)$, where $T_1, \ldots, T_h$ are of type C then every match of P into T lies entirely inside some T_i $(1 \leq i \leq h)$.*

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

Theorem 2. *Algorithm 1 is correct.*

Proof. Let $T(1, \ldots, n)$ (text) and $P(1, \ldots, k)$ (pattern) be two superpositional graphs. We have to show, that there exists a sequence of integers $(i_1, \ldots, i_k)$ (where $1 \leq i_1 < \ldots < i_k \leq n$), such that conditions of Definition 9 are fulfilled if and only if $(i_1, \ldots, i_k) \in match(T[1:n], P[1:k])$.

1. ($\Rightarrow$) Let $(i_1, \ldots, i_k)$ fulfill the conditions of Definition 9.

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

The case $k = 1$ is obvious. By Lemma 3 we can assume, that $(i_1, \ldots, i_p)$ 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 $i_1, \ldots, i_m$ are nodes of $T[r:l]$ and nodes $i_{m+1}, \ldots, i_p$ are nodes of $T[l+1:s]$. By induction hypothesis $(i_1, \ldots, i_m) \in match(T[r:l], P[1:m])$ and $(i_{m+1}, \ldots, i_k) \in match(T[l+1:s], P[m+1:k])$. 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) An edge $m \xrightarrow{0} T0$ is the single edge overcoming a node $m+1$. Both conditions can be checked, using Lemma 1 and Theorem 1. According to Algorithm 1 $(i_1, \ldots, i_k) \in match(T[1:l], P[1:m]) \times match(T[l+1:n], P[m+1:k]) \subseteq match(T[1:n], P[1:k])$.

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

We have to show, that the conditions of Definition 9 are fulfilled. Let $m \in \{1, \ldots, p\}$ 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 $(i_1, \ldots, i_p) \in match(T[1:n], P[1:m])$ in two cases.

a) There are subgraphs $T[r:s]$ and $P[u:v]$ which are equivalent and $m, m+1 \in P[u:v]$. 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 splitpoint is m, $T[r:s]$ is a subgraph of $T[1:n]$, whose leftmost splitpoint is l and $(i_i, \ldots, i_k) \in match(T[r:l], P[u:m]) \times match(T[l+1:s], P[m+1:v])$. 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 $i_m \overset{1}{\rightsquigarrow} i_{m+1}$ and the first part of condition 3 of the Definition 9 is fulfilled. If $i_{m+1} > l+1$, then we have $i_m \overset{1}{\rightsquigarrow} l+1$ and $l \xrightarrow{0} T0$ (again because $T[r:s] = \mathbf{C}(T[r:l], T[l+1:s])$ and l is a leftmost splitting point) and the second part of condition 3 is fulfilled. □

An obvious modification of Algorithm 1 counts the number of matches. Variables X, Y, Z, V, W are of type *integer*.

Algorithm 2. **#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

4 Performance

Obviously, the performance of Algorithm 1 equals to the performance of Algorithm 2 plus the length of matches. Therefore we concentrate on estimating of the performance of Algorithm 2. We rewrite this algorithm, trying to economize the performance. We represent a superpositional graph $G(1,\ldots,n)$ by a two-dimensional array $AG[0:1, 1:n]$, where $AG[0,i] = j$ iff there is a 0-edge $i \xrightarrow{0} j$ and $AG[1,i] = j$ 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 AG just by indexes r, s. So, there is no need to duplicate subgraphs for recursive calls.

Function $equivalent(T[r:s], P[u:v])$, where $T[r:s]$ and $P[u:v]$ are SPGs of equal length, performs obviously in linear time. 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 $AS[1 : n, 1 : n]$. As our algorithm only splits graph into subgraphs and does not contain "joins", every line $i|i+1$ $(1 \leq i < n)$ is used exactly once. Therefore we need to use only $n-1$ elements of an array $AS[1 : n, 1 : n]$. Algorithm 3 calculates splitting points for all subgraphs $G[k : l]$, needed for the full decomposition of G.

Algorithm 3. read *(k,m)*
//uses global arrays $AG[0 : 1, 1 : n]$, representing an SPG and $AS[1 : n, 1 : n]$ for storing the splitting points.
begin
    i:=k; //current node.
    t:=0; //0, if current long edge is 0-edge, 1 otherwise.
    if $AG[1, k] > k + 1$ **then** $t := 1$ **fi**; *//if first 1-edge is longer.*
    while $i < m$
    do $r := \min\{AG[t, i] - 1, m\}$;*//r is an endpoint of a subgraph*
        $AS[k, r] := i$;
        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

Theorem 3. *Algorithm* **read(1,n)** *calculates correctly leftmost splitting points for all subgraphs $G[k : l]$, needed in Algorithm 2, in $\mathcal{O}(n)$ time and uses $n-1$ elements of array AS.*

Algorithm 2 makes multiple recursive calls with the same text and pattern in some cases.

Example. If one calculates a number of matches of $P = (4, 4, 4)(2, 3, 4)$ into $T = (6, 6, 6, 6, 6)(2, 3, 4, 5, 6)$ using Algorithm 2, then $\#match(T[3 : 5], P[2 : 3])$ 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 $AT[0 : 1, 1 : n]$ for a text, $ST[1 : n-1, 1 : n]$ for splitting points of the text, $AP[0 : 1, 1 : k]$ 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 4. 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 ≥ dp − u
        do
           if s − dt ≥ u − dp then
              V := count(T[r : dt], P[u : dp);
              W := count(T[dt + 1 : s], P[dp + 1 : v]);
              Z := Z + (V · W);
           fi:
           dp := split(P[dp + 1 : v]) :
        od
        COUNT[dt, dp] := Z; return Z;
end
```

It is easy to see, that it takes $\mathcal{O}(kn)$ steps to compute the number of matchings. An obvious modification of Algorithm 4 allows us to compute all matches in time $\mathcal{O}(kn + kp)$, where p ia a number of matches.

5 An Application: 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 4 ([5]). *There is a bijection between a set of separable n-permutations and a set of superpositional graphs with n internal nodes.*

A proof of the Theorem 4 can be found in [5]. In the following we need from the proof a linear time algorithm, transforming a separable permutation into SPG.

Algorithm 5. 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

Lemma 4. *Let* $t = t_1 \dots t_n$ *be a separable permutation and* G_t *his SPG, built using Algorithm 5. 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 5 $t(l) < t(m)$. 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 $low(r - 1) = s \le low(l)$ (strong planarity of G_t), $m < low(l)$ and by Algorithm 5 $t(l) < t(m)$. 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: $1 \le l < m \le n$. 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, concequently, 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 subsequence. 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 5. *A separable permutation p matches into a separable permutation t iff G_p matches into G_t.*

Proof. ($\Rightarrow$) 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 9).

($\Leftarrow$) 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 4. □

To find a number of matchings for separable permutations t and p, we need to transform them into superpositional graphs using Algorithm 5. Then we can apply Algorithm 4 and achieve a time complexity $\mathcal{O}(kn)$.

6 Conclusion

We can conclude, that the notion of *superpositional graph* is an useful formalism for a pattern matching problem for separable permutations.

References

1. Bose, P., Buss, P., 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.: Strusturally synthesized binary decision diagrams. In: Proceedings of 6th International Workshop on Boolean Problems. pp. 271–278. Freiberg University of Mining and Technology (2004)
4. Peder, A., Tombak, M.: Superpositional graphs. Acta et Commentationes Universitatis Tartuensis de Mathematica 13, 51–64 (2009)
5. Vohandu, L., Peder, A., Tombak, M.: Permutations and bijections. In: Information Modelling and Knowledge Bases XXIII, Frontiers in Artificial Intelligence and Applications, vol. 237, pp. 419–437. IOS Press (2012)