Longest common palindromic subsequence problem

Marko Djukanovic Günther Raidl Christian Blum

June 26, 2017

Classical longest common subsequence problem-definition

- String $\tilde{s}$ is substring of string s if it can be obtain from s using operation of removing characters in s
- **Problem:** Given strings s_1 and s_2 over Σ, where Σ is an finite alphabetic set. Find the subsequence common for both strings of maximal possible length.
- Example: s_1= aababca and s_2= abcccbaaa
  s = bca is one feasible (not best) solution but s=abcba is not. Clearly, the solution is the subsequence s=abba.

Classical longest common subsequence problem-definition

- String $\tilde{s}$ is substring of string s if it can be obtain from s using operation of removing characters in s
- **Problem:** Given strings s_1 and s_2 over Σ, where Σ is an finite alphabetic set. Find the subsequence common for both strings of maximal possible length.
- Example: s_1= aababca and s_2= abcccbaaa
  s = bca is one feasible (not best) solution but s=abcba is not. Clearly, the solution is the subsequence s=abba.

Solution approaches:

- First approach-Dynamic programming in complexity $\mathcal{O}(n^2)$
- IP approaches

Notations

- $|s|$- the length of the string s
- $s \cdot a$ - string given by appending letter $a \in \Sigma$ to string s
- $s[i]$ - i-th character in s (with starting index $i = 1$)
- $|s|_a, a \in \Sigma$ is the number of occurrences of letter a in s
- $s[i, j] = s[i] \cdot s[i + 1] \cdots s[j]$
- $S = \{s_1, \ldots, s_n\}$ input set

Generalization of LCS in higher dimensions

- How to compare 2D, even 3D objects in similarly manner as LCS does for strings (1D)?
- Need for mathematical (combinatorial) definition of LCS-based on *keeping* the order of the letters in the input strings from the left to the right.

Definition (Amir et al., 2008)

LCS problem for $n = 2$ (classical LCS) is equivalent to the task of finding an one-to-one function $f : \{1, \ldots, |s_1|\} \rightarrow \{1, ..., |s_2|\}$ of maximal domain such that

- $s_1[i] = s_2[f(i)]$, $i \in \mathrm{Dom}(f)$ (matching)
- For $i, j \in Dom(f)$, $i < j \leftrightarrow f(i) < f(j)$ (keeping the order)

2D-LCS

- Given two matrices A and B dimensions n and m respectively.
- **Question:** How to compare these two matrices?
- Idea: generalize combinatorial definition of LCS
- Find one-to-one function $f : \{1, \ldots, n\}^2 \to \{1, \ldots, m\}^2$ of maximum domain size which:
  - For each $(i, j) \in \mathrm{Dom}(f)$: $A[(i, j)] = B[f(i, j)]$
  - Keeps the order in both directions: left$\to$ right and top$\to$bottom
- Problem is NP hard (can be reduced to the clique problem)

Other generalizations of LCS

- Find the LCS for an arbitrarily large set S of input strings
- Problem is NP hard
- Exact approaches:
  - IP approaches and DP

  are not successful (exponential number of variables in IP model)
- Heuristic approaches:
  - Beam Search approach (Blum and Blesa, 2009)
  - Variable neighborhood search approach (Blum and Lozano, 2015)
  - Hybrid approaches BS+ACO (Blum et al., 2016)

Beam search approach for general LCS

- Construction phase of the Beam search procedure: Best-Next heuristic (BNH)
- Each step is based on extending a partial solution s by one letter (which has been conducted to the end of s).

Beam search approach for general LCS

- Construction phase of the Beam search procedure: Best-Next heuristic (BNH)
- Each step is based on extending a partial solution s by one letter (which has been conducted to the end of s).

- Let $s_i = x_i \cdot y_i$ the string of minimal length that contains s and p_i^a the index of first occurrences of the letter a in y_i.
- The letter a is dominated in respect to the partial solution s iff there exists $b \in \Sigma$ such that $p_i^b < p_i^a$, $i = 1, \ldots n$.
- Let Σ_s^{nd} be the set of non-dominated letter in respect to s.

Upper Bound

- The letter for extension of partial solution s is chosen in the term of maximizing the greedy function g given as

$$g(a|s) = \left(\sum_{i=1}^{n} \frac{p_i^a - p_i}{|s_i|} \right)^{-1}, \tag{1}$$

  where $a \in \Sigma_s^{\mathrm{nd}}$ (Blum and Blesa, 2009).
- The evaluation of partial solution s is done by

$$UB(s) = |s| + \sum_{a \in \Sigma} \min\{|s_i|_a |\, i = 1, ..., n\}$$

Longest common palindromic subsequence problem (n=1)

Definition

String s is called palindromic if the reverse of the string is the same as string s.

- **Problem:** For string s, find a maximal subsequence which is *palindromic*.
- Example: For the string s_1 = bananas the solution is s=anana.
- Algorithms:
  - Brute Force ($\mathcal{O}(n^3)$)
  - Dynamic programming ($\mathcal{O}(n^2)$)
  - In-place algorithm, Manacher algorithm (Jeuring, 1994) ($\mathcal{O}(n)$)
- **Harder** problem: Given two strings s_1 and s_2. Find a subsequence of both string which is palindrome of maximal possible length (classical LCPS).

LCPS and biology - motivation

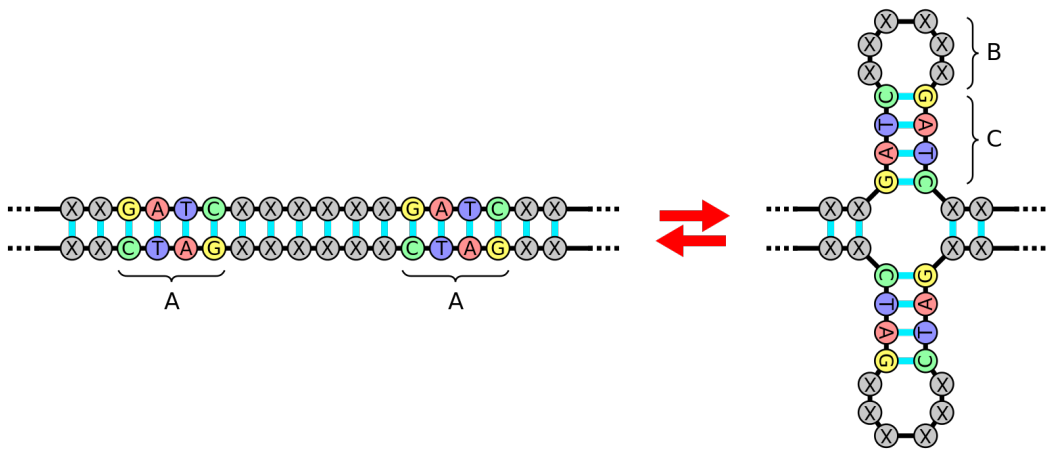
One could asks **why** solving LCPS!

Palindromes appear widespread in the human genetics structure as

- *DNA*, proteins
- cancer cells and genomes.

- Palindromes in biological context exert the regulation of gene activity.
- Instability in the gene functioning can be in direct correspondence with gene palindromic sequences.

Figure: Double helix - example of palindromic subsequence structure in the nature.



Solution approaches

- Dynamic programming (Chowdhury, 2014)($\mathcal{O}(n^4)$)
- Using automaton approach (M.M. Hasan et al., 2017) ($\mathcal{O}(n^4)$)
- Reduction to the computational geometry problem (Inenaga and Hyyrö, 2016); ($\mathcal{O}(\sigma R^2 + n)$) time complexity where R denotes the number of pairs at which two strings are match and σ is the number of distinct characters in both string.

Generalized LCPS

- Extend the problem to an arbitrary input of n strings
- NP hard problem
- To the best of our knowledge, no algorithm has been published yet for solving the LCPS with an arbitrary number of input strings
- Based on a construction heuristic: Palindromic Best-Next Heuristic (PBNH)

PBNH

Main idea (briefly):

- Starts from the empty partial solution $s = \varepsilon$ which pointer variables are $p_i^{\mathrm{L}} = 1$ and $p_i^{\mathrm{U}} = |s_i|$, $i = 1, \ldots n$
- In each iteration: possibly identify a letter a^* using **greedy** selection criteria so that (current) partial solution s can be feasibly extended. Extend s with a^*.
- Update the pointer variables of s such that $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{U}}]$ denotes the maximal possible remaining substring of s_i used for choosing letters in the next extension of s.
- Do it until no further extension of s is possible
- Reconstruction of a partial solution to a feasible solution

PBNH - important questions

Denote the set $\Sigma_s \subseteq \Sigma$ to be the set of all letters appear in each $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$.

The vectors $p_{i,a}^{\mathrm{L}}$ and $p_{i,a}^{\mathrm{R}}$ denote position at which letter a appears first and last in $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$ respectively.

PBNH - important questions

Denote the set $\Sigma_s \subseteq \Sigma$ to be the set of all letters appear in each $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$.
The vectors $p_{i,a}^{\mathrm{L}}$ and $p_{i,a}^{\mathrm{R}}$ denote position at which letter a appears first and last in $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$ respectively.

- 1. Which set of letters to choose for further extension?
  - A letter $a \in \Sigma_s$, a is *dominated* if there exists a letter $b \in \Sigma_s$, $b \neq a$, such that $p_{i,b}^{\mathrm{L}} < p_{i,a}^{\mathrm{L}} \wedge p_{i,b}^{\mathrm{R}} > p_{i,a}^{\mathrm{R}}$ for $i = 1, \ldots, n$.
  - Clearly, better to select **non-dominated** letters $\sum_s^{\mathrm{nd}}$ for the extension of s first.

PBNH - important questions

Denote the set $\Sigma_s \subseteq \Sigma$ to be the set of all letters appear in each $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$.
The vectors $p_{i,a}^{\mathrm{L}}$ and $p_{i,a}^{\mathrm{R}}$ denote position at which letter a appears first and last in $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$ respectively.

- 1. Which set of letters to choose for further extension?
  - A letter $a \in \Sigma_s$, a is *dominated* if there exists a letter $b \in \Sigma_s$, $b \neq a$, such that $p_{i,b}^{\mathrm{L}} < p_{i,a}^{\mathrm{L}} \wedge p_{i,b}^{\mathrm{R}} > p_{i,a}^{\mathrm{R}}$ for $i = 1, \ldots, n$.
  - Clearly, better to select **non-dominated** letters $\sum_s^{\mathrm{nd}}$ for the extension of s first.
- 2. Which greedy criteria to use for choosing the most promising letter in $\sum_s^{\mathrm{nd}}$ used for extension of partial sol. s?
  - Greedy function

$$g(a|s) = \sum_{i=1}^{n} \frac{p_{i,a}^{\mathrm{L}} - p_i^{\mathrm{L}} + p_i^{\mathrm{R}} - p_{i,a}^{\mathrm{R}}}{|s_i|}. \tag{2}$$

Ties are broken randomly.

Pseudo-code for PBNH

input: Set of n strings S

$s \leftarrow \varepsilon$

$(p_i^{\mathrm{L}}, p_i^{\mathrm{R}})_{i=1}^{n} \leftarrow (1, |s_i|)_{i=1}^{n}$

$\Sigma_s^{\mathrm{nd}} \leftarrow SetNDLetter(s, p_i^{\mathrm{L}}, p_i^{\mathrm{R}})$

while$(\Sigma_s^{\mathrm{nd}} \neq \emptyset)$ **do**

$a^* \leftarrow \mathrm{Choose\_From}(\Sigma_s^{\mathrm{nd}})$ %*using greedy function* $g(a|s)$

if $p_{i,a^*}^{\mathrm{L}} = p_{i,a^*}^{\mathrm{R}}$ for some i **then**

return $s \cdot a^* \cdot s^{\mathrm{rev}}$

end if

$s \leftarrow s \cdot a^*$

$(p_i^{\mathrm{L}}, p_i^{\mathrm{R}})_{i=1}^{n} \leftarrow (p_{i,a*}^{\mathrm{L}} + 1, p_{i,a^*}^{\mathrm{R}} - 1)_{i=1}^{n}$

if $p_i^{\mathrm{L}} > p_i^{\mathrm{R}}$ for some i **then**

return $s \cdot s^{\mathrm{rev}}$

end if

$\Sigma_s^{\mathrm{nd}} \leftarrow SetNDLetter(s, p_i^{\mathrm{L}}, p_i^{\mathrm{R}})$

end while

return $s \cdot s^{rev}$

Example - PBNH

- Given 3 strings
  $s_1 = abcbbcab$, $s_2 = accbca$ and $s_3 = abcbcaa$.
  Assume that $s = a$ is a partial solution.

Example - PBNH

- Given 3 strings
  $s_1 = abcbbcab$, $s_2 = accbca$ and $s_3 = abcbcaa$.
  Assume that $s = a$ is a partial solution.
- Then, the remaining relevant substrings for further expansion of the partial solution s are between the underlined letters:
  $s_1 = \underline{a}bcbbc\underline{a}b$ $s_2 = \underline{a}ccbc\underline{a}$ $s_3 = \underline{a}bcbca\underline{a}$
- $\Sigma_s^{\mathrm{nd}} = \{c\}$ (**Why?**)

Example - PBNH

- Given 3 strings
  $s_1 = abcbbcab$, $s_2 = accbca$ and $s_3 = abcbcaa$.
  Assume that $s = a$ is a partial solution.
- Then, the remaining relevant substrings for further expansion of the partial solution s are between the underlined letters:
  $s_1 = \underline{a}bcbbc\underline{a}b$ $s_2 = \underline{a}ccbc\underline{a}$ $s_3 = \underline{a}bcbca\underline{a}$
- $\Sigma_s^{\mathrm{nd}} = \{c\}$ (**Why?**)
- $s_1 = ab\ \underset{p_{1,c}^{\mathrm{L}}}{\underline{c}}\ bb\ \underset{p_{1,c}^{\mathrm{R}}}{\underline{c}}\ ab$
  $s_2 = a\ \underset{p_{2,c}^{\mathrm{L}}}{\underline{c}}\ cb\ \underset{p_{2,c}^{\mathrm{R}}}{\underline{c}}\ a$
  $s_3 = ab\ \underset{p_{3,c}^{\mathrm{L}}}{\underline{c}}\ b\ \underset{p_{3,c}^{\mathrm{R}}}{\underline{c}}\ aa$
- $\rightarrow s = s \cdot c = \mathrm{ac}$; $\Sigma_s^{\mathrm{nd}} = \{b\}$
- $\rightarrow s = s \cdot b = \mathrm{acb}$; $p_{3,c}^{\mathrm{L}} = p_{3,c}^{\mathrm{R}} = 4 \rightarrow s = acb(ac)^{rev} =$ **acbca** is the solution for LCPS problem constructed by PBNH.

Upper bound function UB

- Measures the best possible length of complete solution generated from current partial solution s.
- The candidates for UB:
  - $UB(s) = |s| + \sum_{\{a \in \Sigma_s\}} \min \left\{ \left\lceil \frac{|s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]|_a}{2} \right\rceil \mid i = 1, \ldots, n \right\}$

Upper bound function UB

- Measures the best possible length of complete solution generated from current partial solution s.
- The candidates for UB:
  - $UB(s) = |s| + \sum_{\{a \in \Sigma_s\}} \min \left\{ \left\lceil \frac{|s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]|_a}{2} \right\rceil | i = 1, \ldots, n \right\}$
  - Only one letter among all letters whose number of occurrences in each s_i is exactly one, could increase UB value for maximum 1, i.e., the UB function from above can be tighter:

Upper bound function UB

- Measures the best possible length of complete solution generated from current partial solution s.
- The candidates for UB:
  - $UB(s) = |s| + \sum_{\{a \in \Sigma_s\}} \min \left\{ \left\lceil \frac{|s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]|_a}{2} \right\rceil | i = 1, \ldots, n \right\}$
  - Only one letter among all letters whose number of occurrences in each s_i is exactly one, could increase UB value for maximum 1, i.e., the UB function from above can be tighter:

$$UB(s) = |s| + 1 + \sum_{a \in \Sigma_s^+} \min \left\{ \left\lceil \frac{|s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]|_a}{2} \right\rceil | i = 1, \ldots, n \right\}, \tag{3}$$

where Σ_s^+ denotes the set of all letters appear in each $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$ **at least** two times.

Upper bound function UB

- Measures the best possible length of complete solution generated from current partial solution s.
- The candidates for UB:
  - $UB(s) = |s| + \sum_{\{a \in \Sigma_s\}} \min \left\{ \left\lceil \frac{|s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]|_a}{2} \right\rceil | i = 1, \ldots, n \right\}$
  - Only one letter among all letters whose number of occurrences in each s_i is exactly one, could increase UB value for maximum 1, i.e., the UB function from above can be tighter:

$$UB(s) = |s| + 1 + \sum_{a \in \Sigma_s^+} \min \left\{ \left\lceil \frac{|s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]|_a}{2} \right\rceil | i = 1, \ldots, n \right\}, \tag{3}$$

    where Σ_s^+ denotes the set of all letters appear in each $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$ **at least** two times.
  - For each $s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{R}}]$ the maximal length of palindromic subsequence (P_i) can be computed efficiently (linear time) $\rightarrow$

$$UB(s) = |s| + \min \left\{ \left\lceil \frac{P_i}{2} \right\rceil | i = 1, \ldots, n \right\}. \tag{4}$$

Greedy functions - further development

- Consider the remaining relevant length of each string, the equation (2) becomes:

$$g(a|s) = \sum_{i=1}^{n} \frac{p_{i,a}^{\mathrm{L}} - p_i^{\mathrm{L}} + p_i^{\mathrm{R}} - p_{i,a}^{\mathrm{R}}}{p_i^{\mathrm{R}} - p_i^{\mathrm{L}} + 1}. \tag{5}$$

- As only the letters appear in Σ_s are relevant for the extension, the function (5) can be changed for

$$g(a|s) = \sum_{i=1}^{n} \frac{p_{i,a}^{\mathrm{L}} - p_i^{\mathrm{L}} + p_i^{\mathrm{R}} - p_{i,a}^{\mathrm{R}}}{\sum_{a \in \Sigma_s} |s_i[p_i^{\mathrm{L}}, p_i^{\mathrm{U}}]|_a}. \tag{6}$$

General search strategy

- Let $G = G(N, A)$ be graph whose nodes represent partial solutions and arcs represent the extensions of partial solutions by single letter.
- A node is a triple $(p^{\mathrm{L}}, p^{\mathrm{R}}, |s|)$ where p^{L} and p^{R} are the left and right position vector referencing still relevant substring of all strings in S.
- The initially generated node is $((1, \ldots, 1), (|s_1|, \ldots, |s_n|), 0)$ $\leftrightarrow$ empty partial solution.
- It can happen that several partial solutions with the same length having the same p^{L} and p^{R} $\rightarrow$ want to cover these cases with only one node.
- General search: consists of generated nodes (nodes in G) and open (not yet expanded) nodes Q.

Algorithmic steps for general search strategy

1. Do further steps until Q is not empty
2. Select a node $(p^{\mathrm{L}}, p^{\mathrm{R}}, |s|)$ from Q according to the specific search strategy.
3. Remove the node from Q.
4. Determine the set Σ_s^{nd} following the PBNP.
5. If $\Sigma_s^{\mathrm{nd}} = \emptyset$ then a respective complete solution is reached. Possibly update the best found complete solution so far.
6. For each letter $a \in \Sigma_s^{\mathrm{nd}}$:

   7.1 Extend the current partial solution s by a, update position vectors p^{L} and p^{R}.

   7.2 If a node with the position vectors p^{L} and p^{R} and some $|s'|$ already exists:

      7.2.1 If $|s| + 1 > |s'|$, update s' to $s + 1$ in this existing node and insert the node into Q if it is not already there. Otherwise the current partial solution is dominated and can be pruned.

   Else create a new node $(p^{\mathrm{L}}, p^{\mathrm{R}}, |s| + 1)$, insert it into Q.

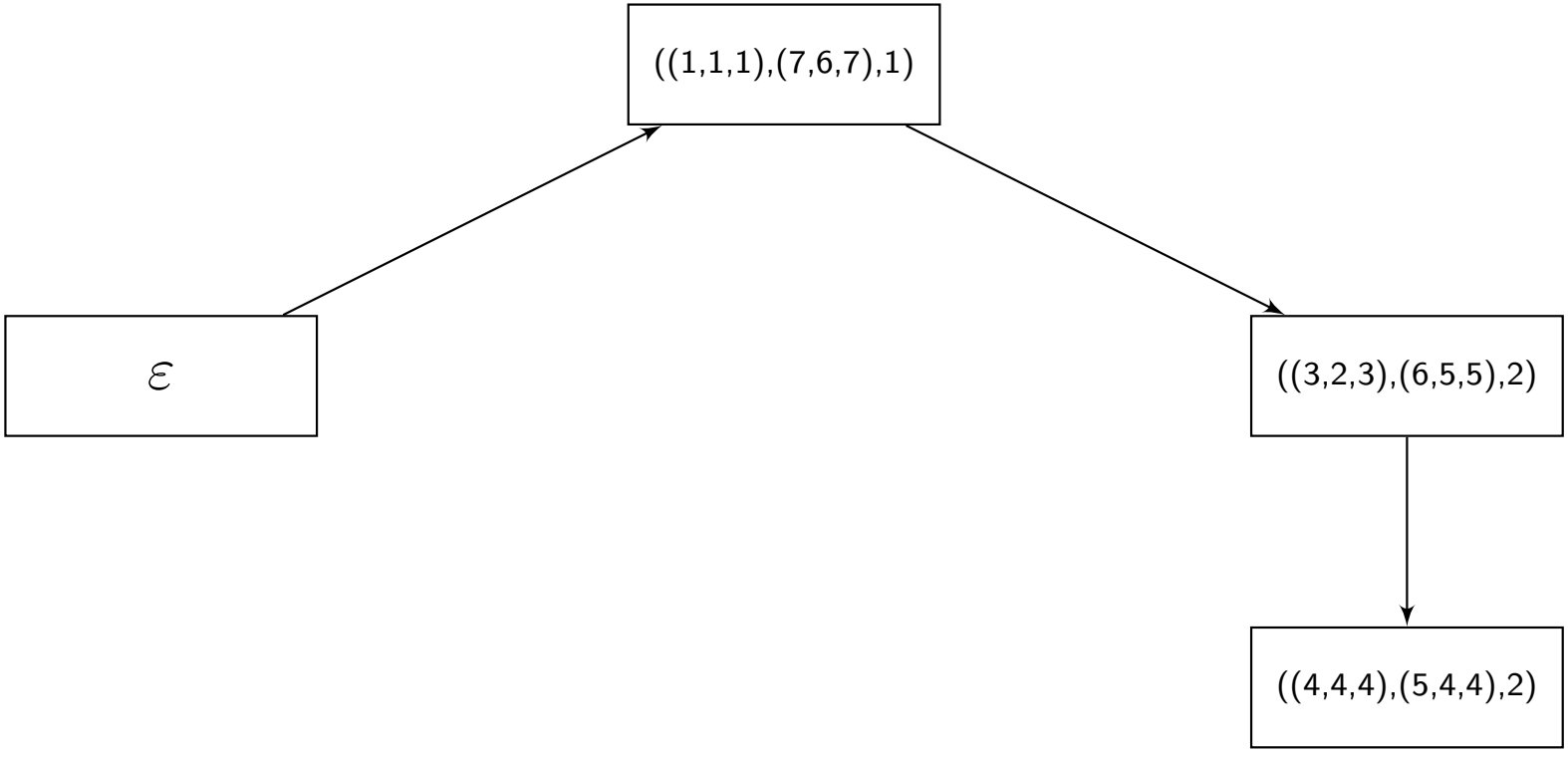
   7.3 Restore the original partial solution s and corresp. p^{L} and p^{R}.
8. Return the best found complete solution

A^* and Beam Search

- A^* search: The set of open nodes Q realized by priority queue in which the entries are sorted using the upper bound function $\mathrm{UB}(s)$.
- Beam Search:
  - The set of most promising search nodes is called *beam* (queue Q in our case)
  - In each step if a new extended node is not in Q or is not complete solution, add it into Q.
  - Non-promising nodes are pruned at any step according to the upper bound value
  - At the end of each step, the reduction of Q is made
  - Two parameter are used: k_{bw} and k_{ext}

Beam Search-example

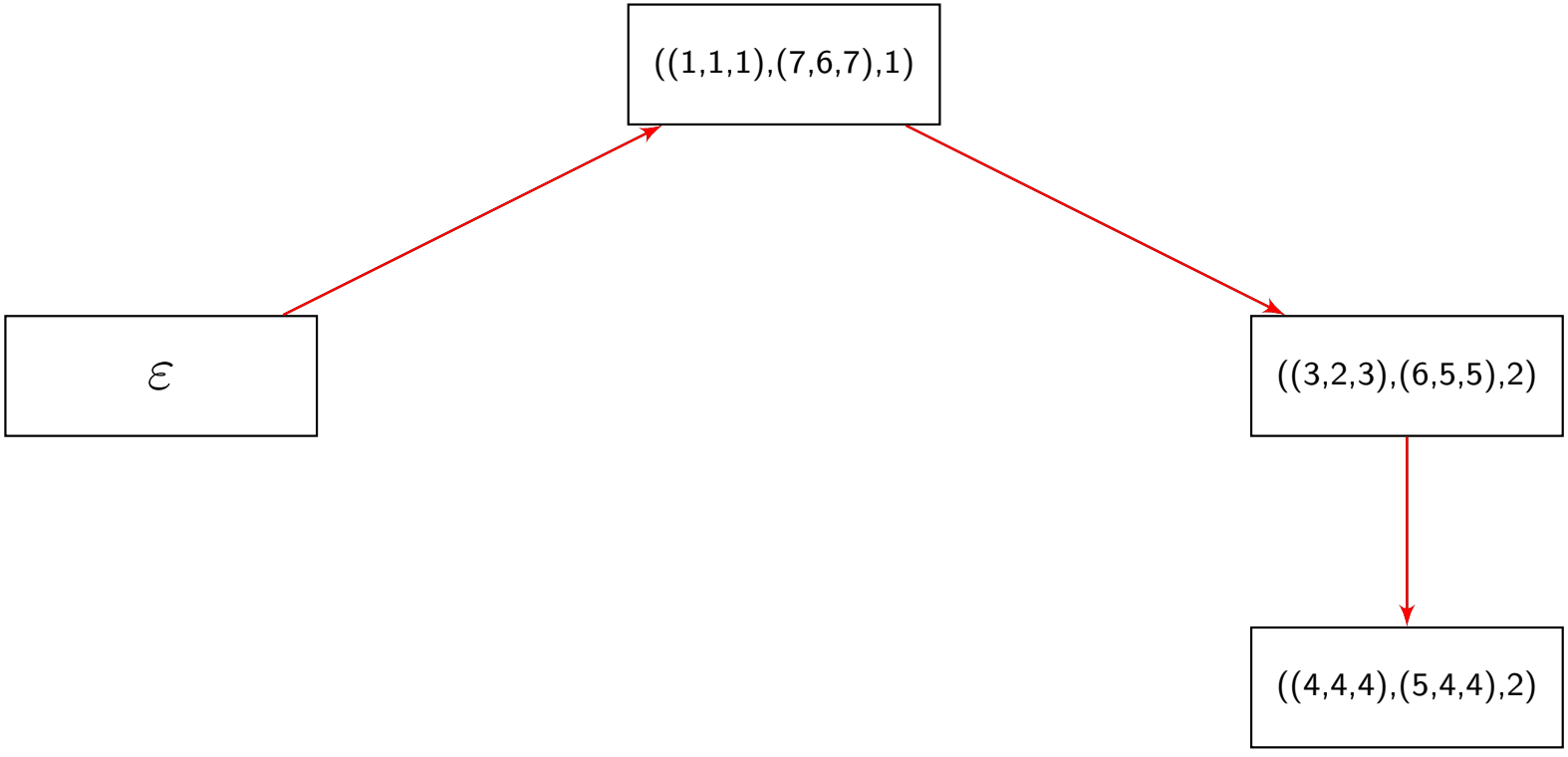
For $k_{bw} = 2 = k_{ext}$ and S=$\{s_1, s_2, s_3\}$ where
$s_1 = abcbbcab$, $s_2 = accbca$ and $s_3 = abcbcaa$ we get:



*green nodes-complete solutions
*gray nodes-final complete solutions
*red edges-reconstruction of the best solution

Beam Search-example

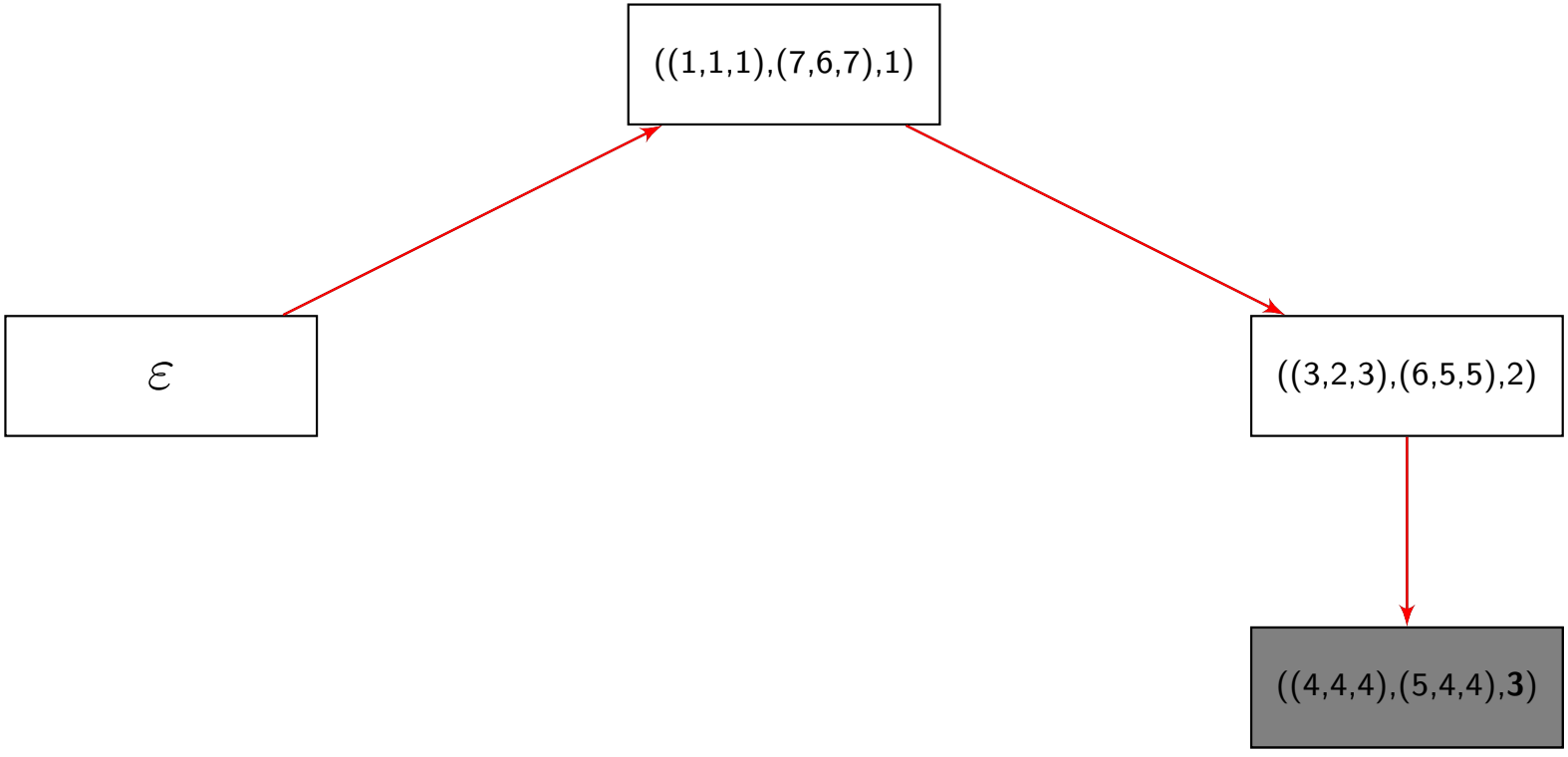
For $k_{bw} = 2 = k_{ext}$ and S=$\{s_1, s_2, s_3\}$ where
$s_1 = abcbbcab$, $s_2 = accbca$ and $s_3 = abcbcaa$ we get:



*green nodes-complete solutions
*gray nodes-final complete solutions
*red edges-reconstruction of the best solution

Beam Search-example

For $k_{bw} = 2 = k_{ext}$ and S=$\{s_1, s_2, s_3\}$ where
$s_1 = abcbbcab$, $s_2 = accbca$ and $s_3 = abcbcaa$ we get:



*green nodes-complete solutions
*gray nodes-final complete solutions
*red edges-reconstruction of the best solution

Thank you for your attention!

Questions?