Computationally Hard Problems

Group: Hotsauce
Troels Thomsen (s152165)
Anders Rydbirk (s113725)

November 2016

1 Introduction

We will be using the following notation in this report. Let R denote the sets of subsets $R_j, 1 \leq j \leq m$, and let T denote the set $t_i, 1 \leq i \leq k$.

2 Decision on test01.SWE

There is no expansion $e(A)$ that makes all k strings in T valid substrings of s, and therefore the answer is *NO*.

3 Proving NP

- We choose the random string S as a sequence of uniformly distributed integers of length m.
- The i'th integer in S corresponds to the index in R_i and the expansion of R_i is given by $e(\gamma_i) = R_{i,S_i}$.
- The decision algorithm verifies S by replacing all letters from Γ appearing in T with the expansion of $e(\gamma_i) = R_{i,S_i}, 1 \leq i \leq m$.
- If all of the resulting k strings are subsets of s the algorithm answers YES, otherwise NO.
- Assume that the conditions of the decision algorithm are met:
    1. Then there is a sequence of words $r_1 \in R_1, r_2 \in R_2, ..., r_m \in R_m$ such that for $1 \leq i \leq k$ the so-called expansion $e(t_i)$ is a substring of s.
    2. Let L be the set of indexes of $r_1, ..., r_m$ in their corresponding sets.

3. Construct a integer string S^* containing all elements in L.
4. The decision algorithm uses S^* to replace letters in T, verify that the resulting k strings are subsets of s and output *YES*.
5. There is a string of length at most m that will output *YES*. The probability of randomly choosing it is positive since it is drawn uniformly at random from a finite set.

- Assume that the conditions of the decision algorithm are not met:
  1. Then there is no sequence of words that satisfies the decision algorithm.
  2. The decision algorithm uses S^* to replace letters in t and checks that the resulting k strings are subsets of s.
  3. As there is no set sequence of words $r_1 \in R_1, r_2 \in R_2, ..., r_m \in R_m$ such that for $1 \leq i \leq k$ the expansion $e(t_i)$ is a substring of s, the algorithm outputs *NO*.
- The running time of the decision algorithm is given by:
  1. We assume we can replace a single letter in $t_i \in T$ in constant time.
  2. Let the length of s be n. Then the length of each string in T is bound by n.
  3. We replace each letter $\gamma_j \in \Gamma, 1 \leq j \leq m$ in T with its corresponding expansion r_j given by S. This uses $O\left(k \cdot n\right)$ time.
  4. We verify that each new string $t'_1, \ldots, t'_k$ is a substring of s. This uses $O(k \cdot n)$ time.
- The worst case running time is $O(2(k \cdot n)) = O(k \cdot n)$ and is polynomial.

4 Proving NP-Completeness

Theorem: SuperStringWithExpansion is NP-complete.

Given an instance of for LongestCommonSubsequence (LCS), we will transform it into an instance for SuperStringWithExpansion (SSWE) such that the original instance is satisfied iff the transformed one is. The input to the transformation is an instance of LCS, i.e. a set of strings $w_1, w_2, ..., w_n$ over an alphabet σ and a natural number k. Let X be the input to LCS and $T(X)$ the transformed input to SSWE.

4.1 Transformation

The transformation proceeds as follows:

- Let σ and $\Gamma = \{\gamma_1, ..., \gamma_n\}$ be disjoint alphabets. Let λ be a special character which does not occur in LCS and let $\Sigma = \sigma \cup \lambda$.

- For each string in $w_1, ..., w_n \in \sigma*$, let $R_i, 1 \leq i \leq n$ consist of all substrings of length k in w_i, and let $e(\gamma_i) = r_i \in R_i$.
- Let t be a string consisting of the concatenation of $\{\gamma_1, \gamma_2, ..., \gamma_n, \lambda\}$.
- Select an arbitrary $i, 1 \leq i \leq n$ and let $W = w_i$. Let S be the set of all substrings of length k in W. Then let s be a string consisting of each substring in S repeated n times separated by λ.

Transforming X to $T(X)$ is polynomial bounded in the number of words n, k and the size of each string. This is given that generating all substrings of length k of a string w uses $O(|w| - k)$ time, and generating s uses $O\left(n(|W| - k)\right)$ time.

4.2 Example

Given the following LCS instance X:

- $w_1 = abc$
- $w_2 = cbc$
- $w_3 = abcc$
- $k = 2$

We transform into the following instance of SSWE T(X) selecting $W = w_1$:

- $s = ababab\lambda bcbcbc\lambda$
- $t = ABC\lambda$
- $R_1 = A : \{ab, bc\}$
- $R_2 = B : \{cb, bc\}$
- $R_3 = C : \{ab, bc, cc\}$
- $k = 2$

The solution to $LCS(X)$ is bc. The solution to $SSWE(T(X))$ is $A = bc$, $B = bc$, $C = bc$.

4.3 Proof

- If the answer to X is *YES*, then there is a substring u of length k present in all strings $w_1, w_2, ..., w_n$. For $T(X)$ we must have $u \in R_i, 1 \leq i \leq n$ and since u is present in each w_i, s must have been constructed from a string W containing the substring u. Thus, there exist a sequence of letters in s of u repeated n times followed by λ which is equal to $e(t)$. For $T(X)$ to answer *YES*, the expansion of each letter in Γ must be $e(\gamma_i) = r_i = u$.

- If the answer to X is *NO*, then there is not a substring u of length k which is present in all $w_1, w_2, ..., w_n$. Given λ is not present in σ, s must contain λ exactly $N = |W| - k + 1$ times where W is the arbitrary string that s is constructed from. Since t ends with λ, there is exactly N substrings in s of length $n \cdot k + 1$ that ends with λ. Since each of these substrings is constructed from substrings of length k from the arbitrary string W, we cannot choose a configuration of t that is a substring of s given that there are no substring of W of length k present in all $R_i, 1 \leq i \leq n$.

5 Heuristic Algorithm

5.1 Design

We have designed a heuristic search algorithm which consists of two parts. First we preprocess the validated input, and then we iteratively search the state space using a Best First Search with a weighted A* heuristic.

5.1.1 Preprocessing

The preprocessing is done by pruning R and T. R is pruned for all letters $\in \Gamma$ not present in T. Besides, we remove all expansions in all subsets of R that is not a substring of s. T is pruned by sorting T by the length in descending order. Let T' be the pruned set of T. Then each $t \in T$ is checked if T' contains a string that are equal to t or if T' contains a string, that t is a substring of. If not, t is added to T'. In case T was not sorted in descending order of the length, the last check would not be applicable.

5.1.2 Best First Search

We have devised a heuristic that aims to limit the search space by pruning invalid expansions from R for each step taken. This is under the observation that the order of application of expansions is indifferent for a valid solution. Thus selecting the set in R that limits the search the most is an applicable heuristic. Then, when one set has been selected, all subsets in R are validated. For all considered states, apply the following for each expansion $r \in R_j, 1 \leq j \leq m$:

1. Replace γ_i with r in all strings in T. Let the replaced set be T'.

2. Let $R' = R$.

3. Validate if each string $t \in T'$ is a valid substring of s. If not, remove r from R'_j.

4. Let h be the size of the set R'_j with the least number of expansions, g be the number of steps applied so far and W the weight applied by the weighted A*. The heuristic for the given state is then given by $W \cdot h + g$.

All processed states are stored in a minimum heap sorted by the heuristic by design of Best First Search. When selecting the most promising state to process, the next state is removed from the heap and all possible actions, a, from the given state are applied. In this case, the actions are all the possible expansions (and only these) of the set in R' with the least number of expansions. This yields $|a|$ new states to consider. All these new states are processed as listed above and then added to the minimum heap.

The search terminates and yields *YES* if it reaches a state where all $\gamma_i, 1 \leq i \leq m$ present in T are replaced by an expansion and all $e(t_j), 1 \leq j \leq k$ are substrings of s. It yields *NO* if it does not manage to find a valid solution.

5.2 Time analysis

5.2.1 Preprocessing

Let α be the total number of expansions in R, and let β be the total length of the strings in T. R is pruned for all γ not present in s which uses $O(m \cdot s)$ time. Pruning of T uses $O(log(k))$ time for sorting, $O(k^2 + \beta)$ time for removing duplicates. Pruning T uses $O(k^2 + \beta)$ time since β can be much larger than k. The total preprocessing time is $O(m \cdot s + k^2 + \beta)$.

5.2.2 Search

The branching factor is given by the average number of expansions in each subset of R, $\frac{\alpha}{m}$. The depth of the search is given by the number of subsets in R, m. Inserting a state into the min heap uses $O\left(\log(\frac{\alpha}{m}^m\right) = O\left(m \log(\frac{\alpha}{m})\right)$ time. Then, the upper time bound for weighted A* itself is $O\left(\frac{\alpha}{m}^m \cdot m \log(\frac{\alpha}{m})\right)$.

The reduction of the search space by removing impossible expansions is applied for each state. Validating each state scans through all expansions in R and each expansion is replaced in T and validated if all T are valid substrings of s. Thus, the worst case running time of this is $O\left(\alpha \cdot k \cdot |s| \cdot \frac{\beta}{k}\right) = O\left(\alpha \cdot |s| \cdot \beta\right)$.

The total worst case running time of the algorithm is $O\left(\alpha \cdot |s| \cdot \beta \cdot \left(\frac{\alpha}{m}\right)^m \cdot m \log\left(\frac{\alpha}{m}\right)\right)$. We have omitted the preprocessing since it is strictly smaller than $O\left(\left(\frac{\alpha}{m}\right)^m\right)$.

6 Implementation

In case the presented file is invalid, a description of the error is printed to the console before exit.

Running our program requires the following arguments:

```
java -jar hotsauce.jar <filename>
```