IA167: String Algorithms **Spring 2014**

Lecture 7: April 7

Lecturer: Alex Popa *Scribes: Dominik Szalai*

7.1 Brief history

With arrival of computing mankind needed some way of formal description of systems, algorithms and problems. At this time researchers started to realize that several problems are algorithmically unsolvable. First foundations were set by Alonzo Church and Alan Turing. In 1936 independently from each other they have shown in their works answer for Hillbert's so called Entscheidungsproblem. They have shown that testing whether an assertion has a proof is algorithmically unsolvable. [EPW,SM92] Later on the concept of $\mathcal{NP}$-completness was developed in the early 1970 in parallel by researchers in the US and the USSR. In 1971 Stephen Cook published paper "The complexity of theorem proving procedures" which included so called Cook-Levin theorem, stating that Boolean satisfiability[1] problem is $\mathcal{NP}$-complete. [CLTW]

7.2 NP-Completness

In these scribes we will refer to three main classes of problems: $\mathcal{P}$, $\mathcal{NP}$ and $\mathcal{NP} - \mathcal{C}$. The class $\mathcal{P}$ consists of problems that are solvable (by deterministic Turing machine) in polynomial time $O(n^k)$ for constant k and n which denotes the input size of the problem. Formally the definition of $\mathcal{P}$ class is following:

Definition 7.1 *A language $\mathcal{L}$ is in $\mathcal{P}$ if and only if there exists a deterministic Turing machine $\mathcal{M}$, such that*

- *$\mathcal{M}$ runs for polynomial time on all inputs*
- *$\forall x \in \mathcal{L}$, $\mathcal{M}$ outputs 1*
- *$\forall x \notin \mathcal{L}$, $\mathcal{M}$ outputs 0*

The class $\mathcal{NP}$[2] consists of problems, that are verifiable in polynomial time. By verification we mean we have a proof of solution stating that the problem can be solved in polynomial time. The formal definition of class $\mathcal{NP}$ is following:

Definition 7.2 *A language $\mathcal{L}$ is in $\mathcal{NP}$ if and only if there exist polynomials p and q, and a deterministic Turing machine $\mathcal{M}$, such that*

- *$\forall (x \wedge y) \in \mathcal{L}$, the machine $\mathcal{M}$ runs in time $p(|x|)$ on input (x, y)*
- *$\forall x \in \mathcal{L}$, there exists a string y of length $q(|x|)$ such that $\mathcal{M}(x, y) = 1$*
- *$\forall x \notin \mathcal{L}$ and all strings y of length $q(|x|)$, $\mathcal{M}(x, y) = 0$*

[1] Also known as SAT problem

[2] from word non-detereministic

The last class, that is important for further reading consists of $\mathcal{NP}$-complete problems.

Definition 7.3 *Problem $\mathcal{A}$ is $\mathcal{NP}$-complete if two following conditions are satisfied:*

1. *$\mathcal{A}$ is in $\mathcal{NP}$*
2. *every problem in $\mathcal{NP}$ is reducible to $\mathcal{A}$ in polynomial time.*[3]

We can describe reduction in definition 7.3 above in following way: Lets suppose that we have a procedure that transforms problem $\mathcal{A}$ into problem $\mathcal{B}$ with two conditions. The first one is that the transformation takes polynomial time, and the other one states that the answers for both problems are same. That is, the answer for $\mathcal{A}$ is yes, if and only if the answer for $\mathcal{B}$ is also yes. [CT09] Some of the well known $\mathcal{NP}$-complete problems with brief description taken from related Wikipedia articles are:

Boolean satisfiability problem Given boolean formula consisting of variables, conjunctions, disjunction negation and parentheses, written in conjunctive normal form we can say wheter there exists an interpretation that makes formula true.

Knapsack problem Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Hamiltonian path problem Problem determining whether a Hamiltonian path, a path that visits each vertex exactly once, exists in the given graph.

Travelling salesman problem Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Subgraph isomorphism problem Given two graphs G and H on input, decide whether G contains a subgraph that is isomorphic to H.

Subset sum problem Given a set (or multiset) of integers, is there a non-empty subset whose sum is zero?

Clique problem There are more variation of this problem e.g. maximum clique. In maximum clique we have to find clique of the largest possible size in given graph.

Vertex cover One of the classical NP-complete problem. The problem is to find in given graph set of vertices such that each edge of the graph is incident to at least one vertex of the set.

Graph coloring Assigning labels (colors) to elements of graphs (vertices, edges) with certain constraint. For example in vertex coloring, assign color to vertices such that no two adjacent vertices have the same color.

7.3 Approximation Algorithms

Lot of practical problems that computer science deals with are $\mathcal{NP}$-complete. We don't know how to find algorithm that has optimal polynomial running time. There are several ways how to deal with this problem. First one might be, if the input is raletively small, we don't have worry with performance, thanks to the speed of nowadays computers. It makes really small difference if algorithm runs in $O(n^2)$ or $O(2^n)$ for small input n e.g $n \leq 5$. The other option is to reduce our entire problem, or at least parts of it into something already known like Satisfiability problem mentioned above. Third option is to find near-optimal solution in polynomial time. These algorithms are called approximation algorithms. [CT09]

[3] Also known as $\mathcal{NP}$-hardness

Definition 7.4 *Let $\mathcal{X}$ be a minimization (respectively, maximization) problem. Let $\varepsilon > 0$, and set $c = 1 + \varepsilon$ (reespectively, $c = 1 - \varepsilon$). An algorithm $\mathcal{A}$ is called a c-approximation algorithm for problem $\mathcal{X}$, if for all instances I of X it delivers a feasible solution with objective value $\mathcal{A}(I)$ such that*

$$|\mathcal{A}(I) - OPT(I)| \leq \varepsilon \cdot OPT(I). \tag{7.1}$$

In this case, the value c is called the performance guarantee or the worst case ratio of the approximation algorithm $\mathcal{A}$.

The value c can be viewed as the quality measure of the approximation algorithm. The closer c is to 1, the better the algorithm is.

Definition 7.5 *Let X be a minimization (respectively, maximization) problem.*

APX *An approximation scheme for problem X is a family of $(1+\varepsilon)$-approximation algorithms $\mathcal{A}_\varepsilon$ (respectively, $(1-\varepsilon)$-approximation algorithms $\mathcal{A}_\varepsilon$) for problem $\mathcal{X}$ over all $0 < \varepsilon < 1$.*

PTAS *A polynomial time approximation scheme for problem $\mathcal{X}$ is an approximation scheme whose time complexity is polynomial in the input size. For example if if algorithm runs in $O(n^{\frac{1}{\varepsilon}})$ then with increasing of ε the power of n decreases, which bring us decreased running time.*

FPTAS *A fully polynomial time approximation scheme for problem $\mathcal{X}$ is an approximation scheme whose time complexity is polynomial in the input size and also polynomial in $\frac{1}{\varepsilon}$. For example running time might be in $O((\frac{1}{\varepsilon})^3 n^2)$.*

Previous two definitions were taken from [SP07]. To show real values of c-approximation we may choose for example problems from section 7.2 such as Vertex-cover which is polynomial time 2-approximation, Travelling salesman problem with triangle inequalities which is also a polynomial time 2-approximation. For Set cover we have polynomial time $(\ln|\mathcal{X}| + 1)$-approximation.

7.4 Shortest common superstring

Problem 7.6 *Given a finite alphabet Σ, and a set of n strings, $S = \{s_1, \cdots, s_n\} \subseteq \Sigma^+$, find a shortest possible string s that contains each s_i as a substring.*

Example 7.7 *Given set $S = \{s_1 = abaab, s_2 = baba, s_3 = aabbb, s_4 = bbab\}$ find the shortest common superstring s.*

Solution

If we merge strings as is shown below we will get the result $s = bbabaabbb$

$s =$	b	b	a	b	a	a	b	b	b
$s_4 =$	b	b	a	b					
$s_2 =$		b	a	b	a				
$s_1 =$			a	b	a	a	b		
$s_3 =$					a	a	b	b	b

Shortest common superstring problem belongs to $\mathcal{NP}$-complete class of problems. The exact algorithm for finding shortest common superstring is in $O(2^n)$ class as has been shown by Held & Karp in [HM61]. In following few sections we will show how several algorithms for obtaining Shortest common superstring.

7.4.1 Greedy algorithm

`Greedy` algorithm repeatedly merges two strings from input set S with maximum overlap until one merged string remains which is the result shortest common superstring. By overlap we mean function $ov(s_i, s_j)$ where two strings are written as $s_i = u.v$ and $s_j = v.w$, then $ov(s_i, s_j) = |v|$. The algorithm with defined steps is following:

Input: Set of input string $S = \{s_1, \cdots, s_n\}$
Output: Shortest common superstring s
  function GREEDY(S)
    while $|S| \neq 1$ **do**
      $s_{ij} = s_i \cdot s_j$ such that $\forall i, j > 0 \wedge i \neq j : ov(s_j, s_i)$ is maximal.
      $S = S \cup s_{ij}$
      $S = S \setminus \{s_i \cup s_j\}$
    end while
    return $s \in S$
  end function

When the algorithm has been defined we may apply it on following example.

Example 7.8 *Given set of strings* $S = \{s_1 = alf, s_2 = ate, s_3 = half, s_4 = lethal, s_5 = alpha, s_6 = alfalfa\}$ *find the shortest common superstring* s.

Solution

Because s_1 is already substring to s_4 and s_6 we can remove it from original set leaving us modified one $S = \{s_2 = ate, s_3 = half, s_4 = lethal, s_5 = alpha, s_6 = alfalfa\}$. The biggest overlap from strings $s_1, \cdots, s_6$ is betweeen $s_3 = half$ and $s_4 = lethal$, with length of overlaping $ov(lethal, half) = 3$. Merging these two strings results in new string $s_{43} = lethalf$. We repeat the step and find out which of strings s_2, s_{43}, s_5, s_6 gives the biggest overlap. After computing all combination we will get to the result that $ov(s_{43}, s_6) = 3$. Merging these two strings results in $s_{436} = lethalfalfa$. Our set now looks as $S = \{s_{436} = lethalfalfa, s_2 = ate, s_5 = alpha\}$. Another repetition of merging step gives `Greedy` algorithm two choices because $ov(s_{436}, s_2) = ov(s_436, s_5) = 1$. With human intuition we would rather merge s_{436} with s_5 because of later possible overlap between s_5 and s_2, resulting in $s_{4365} = lethalfalfalpha$. Final merge of s_{4365} with s_2 results in $s_{43652} = lethalfalfalphate$ which is the shortest common superstring. If we would merge s_{436} with s_2 the result in final step would be then $s_{43625} = lethalfalfatealpha$. The major problem is that the length difference of $|s_{43625}| - |s_{43652}| = 1$ thus the second possible result is one character longer.

Another example where `Greedy` algorithm may fail is set of strings $S = \{c(ab)^k, (ba)^k, (ab)^k c\}$, with result of string twice as long as the optimal one.

7.4.2 4-approxmiation aglorithm

Graph $G_S = (V, E, d)$ has m vertices $V = \{1, \cdots, n\}$, and n^2 edges $E = \{(i, j) : 1 \leq i, j \leq n\}$. String s_i is then associated with vertex i and edge (i, j) is then prefix between strings s_i and s_j. The d is the weight function defined as distance between to vertices. Distance is calculated as prefix between strings $s_i = u \cdot v$ and $s_j = v \cdot w$ defined $pref(s_i, s_j) = |u| = d(s_i, s_j)$. Constructing such graph where vertices are strings

from input set and edges are lengths of prefixes we see similarity between travelling salesman problem where we have to visit all vertices at lowest cost. By [VV001] and [BA91] the key idea is to lower-bound optimal version of algorithm to cycle cover of the prefix graph.

Definition 7.9 *Algorithm Concat-Cycles*

1. *On input S, create graph G_s and find a minimum weight assignment C on G_s. Let C be the collection of cycles $\{c_1, \cdots, c_p\}$.*
2. *for each cycle $c_i = i_1 \to \cdots i_r \to i_1$, let $\tilde{s}_i = \langle s_{i_1}, \cdots, s_{i_r} \rangle$ be the string obtained by opening c_i, where i_1 is arbitrarily chosen. The string $\tilde{s}_i$ has length at most $d(c_i) + |s_{i_1}|$.*
3. *Concatenate together the strings $\tilde{s}_i$ and produce the resulting string $\tilde{s}$ as output.*

For detailed proof with additional theorems and lemmas see page 61 in [VV001].

7.4.3 3-approxmiation aglorithm

3-approximation algorithm uses same steps as 4-approximation algorithm. The only difference is that in step instead of concatenation of strings $\tilde{s}_1, \cdots, \tilde{s}_n$ `greedy` algorithm from subsection 7.4.1 is used.

Notes examples and definintions from previous sections were taken from [BA91]. For proofs and more detailed explanation how `Greedy`, 3 or 4-approximation algorithms work refer to [BA91] aswell.

7.4.4 3.5-approxmiation aglorithm

The bound of 3.5 is made by Kaplan with improvement in `greedy` algorithm shown by Blum in [BA91]. The improvement is done by defining culprits, the special intervals of cycles. For more details with proofs refer to [KH05].

7.4.5 2.59-approxmiation aglorithm

The bound $2\frac{25}{42} \approx 2.569$ is made by constructing a superstring by computing optimal cycle covers on the distance graph. For more details refer to [BD97].

References

[BA91] BLUM, Avrim, Tao JIANG, Ming LI, John TROMP a Mihalis YANNAKAKIS. Linear approximation of shortest superstrings. *Journal of the ACM* [online]. vol. 41, issue 4, pp. 630-647 [cit. 2014-05-04]. DOI: 10.1145/179812.179818. Available at: `http://portal.acm.org/citation.cfm?doid=179812.179818`

[BD97] BRESLAUER, Dany, Tao JIANG a Zhigen JIANG. Rotations of Periodic Strings and Short Superstrings. *Journal of Algorithms* [online]. 1997, vol. 24, issue 2, pp. 340-353 [cit. 2014-05-04]. DOI: 10.1006/jagm.1997.0861. Available at: `http://linkinghub.elsevier.com/retrieve/pii/S0196677497908610`

[CT009] CORMEN, Thomas H. *Introduction to algorithms.* 3rd ed. Cambridge: MIT Press, c2009, xix, 1292 pp. ISBN 978-0-262-03384-8.

[KH005] KAPLAN, Haim a Nira SHAFRIR. The greedy algorithm for shortest superstrings. *Information Processing Letters* [online]. 2005, vol. 93, issue 1, pp. 13-17 [cit. 2014-05-04]. DOI: http://dx.doi.org/10.1016/j.ipl.2004.09.012. Available at: `http://www.cs.tau.ac.il/~haimk/papers/greedy3.5.2.ps`

[HM61] HELD, Michael a Richard M. KARP. A dynamic programming approach to sequencing problems. *Proceedings of the 1961 16th ACM national meeting on* [online]. New York, New York, USA: ACM Press, 1961, 71.201-71.204 [cit. 2014-05-03]. DOI: 10.1145/800029.808532. Available at: `http://portal.acm.org/citation.cfm?doid=800029.808532`

[VV001] VAZIRANI, Vijay V. *Approximation Algorithms* [online]. Berlin: Springer, 2001, 378 s. [cit. 2014-05-05]. ISBN 35-406-5367-8. Available at: `www.cc.gatech.edu/fac/Vijay.Vazirani/book.pdf`

[SM92] SIPSER, Michael. The history and status of the P versus NP question. *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing - STOC '92* [online]. New York, New York, USA: ACM Press, 1992, pp. 603-618 [cit. 2014-05-03]. DOI: 10.1145/129712.129771. Available at: `http://portal.acm.org/citation.cfm?doid=129712.129771`

[SP007] SCHUURMAN, Petra a Gerhard J. WOEGINGER. *Approximation Schemes: A Tutorial* [online]. 2007 [cit. 2014-05-03]. Available at: `http://www.win.tue.nl/~gwoegi/papers/ptas.pdf`

[SZ000] SWEEDYK, Z. A $\mathbf{2\frac{1}{2}}$-Approximation Algorithm for Shortest Superstring. *SIAM Journal on Computing* [online]. 2000, vol. 29, issue 3, pp. 954-986 [cit. 2014-05-04]. DOI: 10.1137/S0097539796324661. Available at: `http://epubs.siam.org/doi/abs/10.1137/S0097539796324661`

[WCLT] CookLevin theorem. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-05-03]. Available at: `http://en.wikipedia.org/wiki/Cook%27s_theorem`

[WEP] Entscheidungsproblem. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-05-03]. Available at: `http://en.wikipedia.org/wiki/Entscheidungsproblem`

[WNP] NP (complexity). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-05-03]. Available at: `http://en.wikipedia.org/wiki/NP_(complexity)`

[WP] P (complexity). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-05-03]. Available at: `http://en.wikipedia.org/wiki/P_(complexity)`