A Heuristic Approach for Solving the Longest Common Square Subsequence Problem*

Marko Djukanovic[1], Günther R. Raidl[1], and Christian Blum[2]

[1]Institute of Logic and Computation, TU Wien, Vienna, Austria,
[2] Artificial Intelligence Research Institute (IIIA-CSIC),
Campus UAB, Bellaterra, Spain
{djukanovic,raidl}@ac.tuwien.ac.at,
christian.blum@iiia.csic.es

In the field of bioinformatics, strings are used to represent DNA and protein sequences. Given a string s, any string which can be obtained from s by deleting zero or more characters is called a *subsequence* of s. The relatedness of molecules can be characterized by finding a longest subsequence that all respective strings have in common. The length of this subsequence is a well-known string comparison measure [4] which implies to solve the *Longest Common Subsequence* (LCS) problem in general on an arbitrary set of strings $S = \{s_1, \ldots, s_m\}, m \in \mathbb{N}$.

Inoue et al. [3] recently proposed a variant of it called the *Longest Common Square Subsequence* (LCSqS) problem, which asks for the longest common subsequence that is a square. A string s is called a *square* if it consists of two identical concatenated parts, i.e., $s = s' \cdot s' = s'^2$ for some string s', where the "$\cdot$" operator indicates the concatenation. The problem is motivated by measuring the similarity of DNA molecules under the consideration of similar disjunctive parts in each molecule. The LCSqS can give more insight on the similarities of molecules than taking the traditional LCS as similarities between internal parts of molecules also play a role, and more can be learned about the structure of compared molecules.

Inoue et al. [3] proposed two approaches for solving LCSqS, but just for two input strings. The first one is a basic *Dynamic Programming* (DP) approach running in $O(n^6)$ time, where n is the length of the largest string while the second one is a sparse DP approach which makes use of a special geometric data structure. The authors proved that the LCSqS with an arbitrary set of m input strings S is NP-hard. To the best of our knowledge, no algorithm for solving the LCSqS for $m > 2$ has yet been proposed.

We suggest a heuristic approach to tackle the general LCSqS problem. First, note that an LCSqS instance can be transformed into a series of standard LCS instances by the following mapping. For each *partition* vector $p = (p_1, \ldots, p_m)$ with $p_i \in \{1, \ldots, |s_i|\}$ for $i = 1, \ldots, m$, let us consider splitting each string s_i into the two substrings s'_i and s''_i with $s_i = s'_i \cdot s''_i$, s'_i ending with position p_i of s_i and s''_i starting with position $p_i + 1$ of s_i. Hence, each vector p maps an input set S with $m = |S|$ into a set S^p of $2m$ strings. Solving the LCS for S^p yields

* This project is partially funded by the Doctoral Program Vienna Graduate School on Computational Optimization, Austrian Science Foundation Project No. W1260-N35.

an LCS s^p, and the corresponding string $s^p \cdot s^p$ is a feasible common square subsequence for S. Taking a longest $s^p \cdot s^p$ over all possible partition vectors p would yield the LCSqS. Unfortunately, already solving the classical LCS problem is NP-hard and challenging in practice, and thus a naive enumeration over all partition vectors and determining the corresponding LCSs is out of question, as the number of possible partition vectors is exponential in the problem size.

However, we make use of the underlying idea of this decomposition as follows. A Variable Neighborhood Search (VNS) algorithm [5] is used as framework for deriving promising partition vectors p. Each candidate partition vector is at the first place rather crudely evaluated by a fast estimation of the length of the LCS for S^p. Only more promising candidate partition vectors are then also more precisely considered by solving the LCS problem for set S^p with a Beam Search (BS) heuristic [7]. This BS is inspired by the work of Mousavi and Tabataba [6] and incorporates a novel heuristic guidance that approximates the expected length of LCS subproblems.

As a fast alternative, we consider a simpler strategy for choosing the partition vector and use the well-known best-next heuristic for the LCS from [2] to obtain a solution for the LCSqS problem. We further randomize and iterate this fast heuristic in order to obtain an iterated greedy approach, which we compare to the above VNS & BS hybrid.

Experiments are performed using the LCS benchmark instances from [1] and detailed results will be presented at the conference. They illustrate the clear advantages of the proposed VNS & BS hybrid over the iterated greedy method in terms of the quality of obtained solutions.

References

1. M. Djukanovic, G. Raidl, and C. Blum. Exact and heuristic approaches for the longest common palindromic subsequence problem. In *Proceedings of LION12 – 12th International Conference on Learning and Intelligent Optimization*, Lecture Notes in Computer Science. Springer, 2018. In press.
2. K. Huang, C.-B. Yang, K.-T. Tseng, et al. Fast algorithms for finding the common subsequence of multiple sequences. In *Proceedings of the International Computer Symposium*, pages 1006–1011. IEEE press, 2004.
3. T. Inoue, S. Inenaga, H. Hyyrö, H. Bannai, and M. Takeda. Computing longest common square subsequences. In G. Navarro et al., editors, *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics*, pages 15:1–15:13, Dagstuhl, Germany, 2018.
4. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM.*, 25(2):322–336, 1978.
5. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations research*, 24(11):1097–1100, 1997.
6. S. R. Mousavi and F. Tabataba. An improved algorithm for the longest common subsequence problem. *Computers & Operations Research*, 39(3):512–520, 2012.
7. P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.