Theoretical Computer Science 660 (2017) 57–74

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the parameterized complexity of associative and commutative unification [☆]

Tatsuya Akutsu [a], Jesper Jansson [a], Atsuhiro Takasu [b], Takeyuki Tamura [a,*]

[a] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, 611-0011, Japan
[b] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

A R T I C L E   I N F O

Article history:
Received 1 March 2016
Received in revised form 13 September 2016
Accepted 22 November 2016
Available online 28 November 2016
Communicated by A. Kucera

Keywords:
Unification
Parameterized algorithm
Dynamic programming
Tree edit distance

A B S T R A C T

This article studies the parameterized complexity of the unification problem with associative, commutative, or associative-commutative functions with respect to the parameter "number of variables". It is shown that if every variable occurs only once then both of the associative and associative-commutative unification problems can be solved in polynomial time, but that in the general case, both problems are $W[1]$-hard even when one of the two input terms is variable-free. For commutative unification, an algorithm whose time complexity depends exponentially on the number of variables is presented; moreover, if a certain conjecture is true then the special case where one input term is variable-free belongs to FPT. Some related results are also derived for a natural generalization of the classic string and tree edit distance problems that allows variables.

1. Introduction

Unification is a useful concept in many areas of computer science such as automated theorem proving, program verification, natural language processing, logic programming, and database query systems [5,17,19,20]. In its fundamental form, the unification problem is to find a substitution for all variables in two given *terms* that makes the terms identical, where terms are constructed from function symbols, variables, and constants [20]. As an example, the two terms $f(x, y)$ and $f(g(a), f(b, x))$, where f and g are functions, x and y are variables, and a and b are constants, become identical by substituting x by $g(a)$ and y by $f(b, g(a))$.

Unification has a long history beginning with the seminal work of Herbrand in 1930 (see, e.g., [20]). It is becoming an active research area again because of *math search*, an information retrieval (IR) task where the objective is to find all documents containing a specified mathematical formula and/or all formulas similar to a query formula [18,23,24]; for example, math search has been adopted as a pilot task in the IR evaluation conference NTCIR [24]. Also, math search systems such as Wolfram Formula Search and Wikipedia Formula Search have been developed. Since mathematical formulas are typically represented by rooted trees, it may seem reasonable to measure the similarity between formulas simply by measuring the

☆ A preliminary version of this article appeared in *Proceedings of the 9th International Symposium on Parameterized and Exact Computation* (IPEC 2014), volume 8894 of *Lecture Notes in Computer Science*, pp. 15–27, Springer International Publishing Switzerland, 2014.

* Corresponding author.

E-mail addresses: takutsu@kuicr.kyoto-u.ac.jp (T. Akutsu), jj@kuicr.kyoto-u.ac.jp (J. Jansson), takasu@nii.ac.jp (A. Takasu), tamura@kuicr.kyoto-u.ac.jp (T. Tamura).

URLs: http://www.bic.kyoto-u.ac.jp/takutsu/members/takutsu/index.html (T. Akutsu), http://sunflower.kuicr.kyoto-u.ac.jp/~jj/ (J. Jansson), http://www.nii.ac.jp/en/faculty/digital_content/takasu_atsuhiro/ (A. Takasu), http://sunflower.kuicr.kyoto-u.ac.jp/~tamura/index.html.en (T. Tamura).

http://dx.doi.org/10.1016/j.tcs.2016.11.026

structural similarity of their trees. However, methods directly based on approximate tree matching such as the *tree edit distance* (see, e.g., the survey in [7]) alone are not sufficient if every label is treated as a constant. For example, under unit cost edit operations, the query $x^2 + x$ has the same tree edit distance to each of the formulas $y^2 + z$ and $y^2 + y$, although $y^2 + y$ is mathematically the same as $x^2 + x$ while $y^2 + z$ is not.

Because of the practical importance of the unification problem and its variants, heuristic algorithms have been proposed, some of which incorporate approximate tree matching techniques [15,17]. On the negative side, their worst-case performance may be poor. To make the study of the computational complexity of unification more formal, this article examines some natural variants of the unification problem from the viewpoint of parameterized complexity and presents several new algorithms for them. The parameterized complexity is considered with respect to the parameter "number of variables appearing in the input"; we choose this parameter because the number of variables is often much smaller than the length of the terms.

1.1. Related work

An exponential-time algorithm for the unification problem was given in [26]. In the twenty years that followed, numerous faster and more practical algorithms were published (see [20] for a comprehensive survey), and in particular, a linear-time algorithm for the problem was developed [11,25].

Various extensions of unification have also been considered [5,6,19,20]. Three such extensions are unification with associative, commutative, and associative-commutative functions (where a function f is called *associative* if $f(x, f(y,z)) = f(f(x,y),z)$ always holds, *commutative* if $f(x,y) = f(y,x)$ always holds, and *associative-commutative* if it is both associative and commutative). These are especially relevant for math search since many functions encountered in practice have one of these properties. Interestingly, when allowing such functions, there are more ways to map nodes in the two corresponding trees to each other, and as a result, the computational complexity of unification may *increase*. Indeed, each of the associative, commutative, and associative-commutative unification problems is NP-hard [6,12].

The special case of unification where one of the two input terms contains no variables is also known as *matching*. Unfortunately, all of the associative, commutative, and associative-commutative matching problems remain NP-hard [6,12], and polynomial-time algorithms are known only for very restricted cases [2,6,19]. E.g., associative-commutative matching can be done in polynomial time if every variable occurs exactly once [6].

We remark that associative unification is in PSPACE and both commutative unification and associative-commutative unification are in NP (see, e.g., the references in Section 3.4 in [5]). Although this means that all three problems can be solved in single exponential-time in the size of the input, it does not necessarily mean single exponential-time algorithms with respect to the number of variables.

For an introduction to parameterized complexity, the reader is referred to the textbook [14]. When a problem is proved to be W[1]-hard or W[2]-hard, it is strongly believed that developing an FPT algorithm is impossible. To prove W[i]-hardness, we often use reduction from Longest Common Subsequence (LCS). LCS is, given a set of strings $R = \{r_1, r_2, \ldots, r_q\}$ over an alphabet Σ_0 and an integer l, to determine whether there exists a string r of length l such that r is a subsequence of r_i for every $r_i \in R$, where r is called a *subsequence* of r' if r can be obtained by performing deletion operations on r'. LCS is $W[1]$-hard with respect to the parameter (q,l) (this problem variant is called "LCS-3" in [8]). On the other hand, LCS is $W[2]$-hard with respect to the parameter l (this problem variant is called "LCS-2" in [8]). The definitions of FPT and W[i] by [8] are given in Appendix A.

Other previous work on associative and/or commutative unification has focused on central aspects such as termination, soundness, and completeness of algorithms (see, for example, [28]) as well as implementations [27]. In computational experiments done in the 1980s, associative-commutative unification was not efficiently computable in general [9], but computable for DO-terms for very small instances [21].

1.2. Summary of new results

We present a number of new results on the parameterized complexity of associative, commutative, and associative-commutative unification with respect to the parameter "number of variables appearing in the input", denoted from here on by k. See Table 1 for an overview. Most notably:

- Both associative and associative-commutative matching are $W[1]$-hard.
- Commutative matching can be done in $O(2^k poly(m,n))$ time, where m and n are the sizes of the two input terms, if a certain conjecture is true.
- Both associative and associative-commutative unification can be done in polynomial time if every variable occurs exactly once.
- Commutative unification can be done in polynomial time if the number of variables is bounded by a constant.

In addition, we show that generalizing the classic string and tree edit distance problems [7,16] to also allow variables yields $W[1]$-hard problems.