Theoretical Computer Science 639 (2016) 91–109

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Algebraic dynamic programming for multiple context-free grammars

Maik Riechert [a,b], Christian Höner zu Siederdissen [b,c,d,*], Peter F. Stadler [b,c,d,e,f,g,h,*]

[a] Fakultät Informatik, Mathematik und Naturwissenschaften, Hochschule für Technik, Wirtschaft und Kultur Leipzig, Gustav-Freytag-Straße 42a, D-04277 Leipzig, Germany
[b] Bioinformatics Group, Department of Computer Science University of Leipzig, Härtelstraße 16-18, D-04107 Leipzig, Germany
[c] Institute for Theoretical Chemistry, University of Vienna, Währingerstraße 17, A-1090 Wien, Austria
[d] Interdisciplinary Center for Bioinformatics, University of Leipzig, Härtelstraße 16-18, D-04107 Leipzig, Germany
[e] Max Planck Institute for Mathematics in the Sciences, Inselstraße 22, D-04103 Leipzig, Germany
[f] Fraunhofer Institut für Zelltherapie und Immunologie, Perlickstraße 1, D-04103 Leipzig, Germany
[g] Center for Non-Coding RNA in Technology and Health, University of Copenhagen, Grønnegårdsvej 3, DK-1870 Frederiksberg C, Denmark
[h] Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM 87501, United States

ARTICLE INFO

Article history:
Received 15 July 2015
Received in revised form 20 April 2016
Accepted 20 May 2016
Available online 26 May 2016
Communicated by D. Perrin

Keywords:
Multiple context-free grammars
Dynamic programming
Algebraic dynamic programming
RNA secondary structure prediction
Pseudoknots

ABSTRACT

We present theoretical foundations, and a practical implementation, that makes the method of Algebraic Dynamic Programming available for Multiple Context-Free Grammars. This allows to formulate optimization problems, where the search space can be described by such grammars, in a concise manner and solutions may be obtained efficiently. This improves on the previous state of the art which required complex code based on hand-written dynamic programming recursions. We apply our method to the RNA pseudoknotted secondary structure prediction problem from computational biology.
Appendix and supporting files available at: http://www.bioinf.uni-leipzig.de/Software/gADP/

1. Introduction

Dynamic programming (DP) is a general algorithmic paradigm that leverages the fact that many complex problems of practical importance can be solved by recursively solving smaller, overlapping, subproblems [21]. In practice, the efficiency of DP algorithms is derived from "memoizing" and combining the solutions of subproblems of a restricted set of subproblems. DP algorithms are particularly prevalent in discrete optimization [19, Chp. 15], with many key applications in bioinformatics.

DP algorithms are usually specified in terms of recursion relations that iteratively fill a multitude of memo-tables that are indexed by sometimes quite complex objects. This makes the implementation of DP recursions and the maintenance of the code a tedious and error prone task [28].

The theory of Algebraic Dynamic Programming (ADP) [29] circumvents these practical difficulties for a restricted class of DP algorithms, namely those that take strings or trees as input. It is based on the insight that for a very large class of problems the structure of recursion, i.e., the construction of the state space, the evaluation of sub-solutions, and the

* Corresponding authors.
E-mail addresses: choener@bioinf.uni-leipzig.de (C. Höner zu Siederdissen), studla@bioinf.uni-leipzig.de (P.F. Stadler).

http://dx.doi.org/10.1016/j.tcs.2016.05.032
0304-3975/

selection of sub-solutions based on their value can be strictly separated. In ADP, a DP algorithm is completely described by a context free grammar (CFG), an evaluation algebra, and a choice function. This separation confers two key advantages to the practice of programming: (1) The CFG specifies the state space and thus the structure of the recursion without any explicit use of indices. (2) The evaluation algebra can easily be replaced by another one. The possibility to combine evaluation algebras with each other [90] provides extensive flexibility for algorithm design. The same grammar thus can be used to minimize scores, compute partition functions, density of states, and enumerate a fixed number of sub-optimal solutions. Given the set of S of feasible solutions and the cost function $f : S \to \mathbb{R}$, the partition function is defined as the sum of "Boltzmann factors" $Z(\beta) = \sum_{s \in S} \exp(-\beta f(s))$. The density of states is the number of solutions with a given value of the cost function $n_f(u) = \left|\{s \in S | f(s) = u\}\right|$. They are related by $Z(\beta) = \sum_u n_f(u) \exp(-\beta f(s))$. Both quantities play a key role in statistical physics [7]. More generally, they provide a link to the probabilistic interpretation by virtue of the relation $Prob(s) = \exp(-\beta f(s))/Z$.

The strict separation of state space construction and evaluation is given up e.g. in the context of sparsification [62,44], where the search space is pruned by means of rules that depend explicitly on intermediate evaluations. Similarly, shortest path algorithms such as Dijsktra's [22] construct the state space in a cost-dependent manner. At least some of these approaches can still be captured with a suitably extended ADP formalism [62]. A class of DP algorithms to which the current framework of ADP is not applicable are those that iterate over values of the cost function, as in the standard DP approach to the knapsack problem [3].

Alternative abstract formalisms for dynamic programming have been explored. The `tornado` software [77] uses a "super-grammar" that can be specialized to specific RNA folding models. Much more generally, *forward-hypergraphs* were introduced in [70] as an alternative to grammars to describe dependencies between partial solutions. *Inverse coupled rewrite systems* (ICORES) "describe the solutions of combinatorial optimization problems as the inverse image of a term rewrite relation that reduces problem solutions to problem inputs" [31]. So far, there it has remained unclear, however, if and how this paradigm can be implemented in an efficient manner.

As it stands, the ADP framework is essentially restricted to decompositions of the state space that can be captured by CFGs. This is not sufficient, however, to capture several difficult problems in computational biology. We will use here the prediction of pseudoknotted RNA structures as the paradigmatic example. Other important examples that highlight the complicated recursions in practical examples include the simultaneous alignment and folding of RNA (a.k.a. Sankoff's algorithm [79]), implemented e.g. in `foldalign` [33] and `dynalign` [59], and the RNA-RNA interaction problem (RIP [2]). For the latter, equivalent implementations using slightly different recursions have become available [17,42,43], each using dozens of 4-dimensional tables to memoize intermediate results. The implementation and testing of such complicated multi-dimensional recursions is a tedious and error-prone process that hampers the systematic exploration of variations of scoring models and search space specifications. The three-dimensional structure of an RNA molecule is determined by topological constraints that are determined by the mutual arrangements of the base paired helices, i.e., by its secondary structure [6]. Although most RNAs have simple structures that do not involve crossing base pairs, pseudoknots that violate this simplifying condition are not uncommon [91]. In several cases, pseudoknots are functionally important features that cannot be neglected in a meaningful way, see e.g. [23,64,27]. In its most general form, RNA folding with stacking-based energy functions is NP-complete [1,58]. The commonly used RNA folding tools (`mfold` [101] and the `Vienna RNA Package` [56]), on the other hand, exclude pseudoknots altogether.

Polynomial-time dynamic programming algorithms can be devised for a wide variety of restricted classes of pseudoknots. However, most approaches are computationally very demanding, and the design of pseudoknot folding algorithms has been guided decisively by the desire to limit computational cost and to achieve a manageable complexity of the recursions [75]. Consequently, a plethora of different classes of pseudoknotted structures have been considered, see e.g. [18,78,15,74], the references therein, and the book [73]. Since the corresponding folding algorithms have been implemented at different times using different parametrizations of the energy functions it is hard to directly compare them and their performance. On the other hand, a more systematic investigation of alternative structure definitions would require implementations of the corresponding folding algorithms. Due to the complicated structure of the standard energy model this would entail major programming efforts, thus severely limiting such efforts. Already for non-pseudoknotted RNAs that can be modeled by simple context-free grammars, the effort to unify a number of approaches into a common framework required a major effort [77].

Multiple context-free grammars (MCFG) [81] have turned out to be a very natural framework for the RNA folding problem with pseudoknots. In fact, most of the pseudoknot classes can readily be translated to multiple context-free grammars (MCFG), see Fig. 1(1) for a simple example. In contrast to CFGs, the non-terminal symbols of MCFGs may consist of multiple components that must be expanded in parallel. In this way, it becomes possible to couple separated parts of a derivation and thus to model the crossings inherent in pseudoknotted structures. Reidys et al. [74], for instance, makes explicit use of MCFGs to derive the DP recursions. Stochastic MCFGs were used for RNA already in [50], and a unified view of many pseudoknot classes recently has been established in terms of MCFGs [65], introducing a direct connection between MCFGs and generating functions.

In computational linguistics, many distinct "mildly context-sensitive grammar formalisms" have been introduced to capture the syntactic structure of natural language. Dutch and Swiss–German, for example, have non-context-free structures [86,88], see Fig. 1(2). Given the constraints of natural languages, the complexity of their non-context free structure is quite limited in practice. MCFGs have been used explicitly to model grammars for natural languages e.g. in [88].