RESEARCH

Algebraic Dynamic Programming over General Data Structures

Christian Höner zu Siederdissen[1,2,4*], Sonja J Prohaska[3,4] and Peter F Stadler[1,2,4,5,6,7,8]

Abstract

Background: Dynamic programming algorithms provide exact solutions to many problems in computational biology, such as sequence alignment, RNA folding, hidden Markov models (HMMs), and scoring of phylogenetic trees. Structurally analogous algorithms compute optimal solutions, evaluate score distributions, and perform stochastic sampling. This is explained in the theory of Algebraic Dynamic Programming (ADP) by a strict separation of state space traversal (usually represented by a context free grammar), scoring (encoded as an algebra), and choice rule. A key ingredient in this theory is the use of yield parsers that operate on the ordered input data structure, usually strings or ordered trees. The computation of ensemble properties, such as *a posteriori* probabilities of HMMs or partition functions in RNA folding, requires the combination of two distinct, but intimately related algorithms, known as the inside and the outside recursion. Only the inside recursions are covered by the classical ADP theory.

Results: The ideas of ADP are generalized to a much wider scope of data structures by relaxing the concept of parsing. This allows us to formalize the conceptual complementarity of inside and outside variables in a natural way. We demonstrate that outside recursions are generically derivable from inside decomposition schemes. In addition to rephrasing the well-known algorithms for HMMs, pairwise sequence alignment, and RNA folding we show how the TSP and the shortest Hamiltonian path problem can be implemented efficiently in the extended ADP framework. As a showcase application we investigate the ancient evolution of HOX gene clusters in terms of shortest Hamiltonian paths.

Conclusions: The generalized ADP framework presented here greatly facilitates the development and implementation of dynamic programming algorithms for a wide spectrum of applications.

Keywords: formal grammar; dynamic programming; gene duplications

1 Background

Dynamic Programming (DP) over rich index sets provides solutions of a surprising number of combinatorial optimization problems. Even for NP-hard problems such as the Travelling Salesman Problem (TSP) exact solutions can be obtained for moderate size problems of practical interest. The corresponding algorithms, however, are usually specialized and use specific properties of the problem in an *ad hoc* manner that does not generalize particularly well.

Algebraic dynamic programming (ADP) [1] defines a high-level descriptive domain-specific language for dynamic programs over sequence data. The ADP framework allows extremely fast development even of quite complex algorithms by rigorously separating the traversal of the state space (by means of context free grammars, CFGs), scoring (in terms of suitable algebras), and selection of desired solutions. The use of CFGs to specify the state space is a particular strength of ADP since it allows the user to avoid indices and control structures altogether, thereby bypassing many of the pitfalls (and bugs) of usual implementations. Newer dialects of ADP [2, 3] provide implementations with a running time performance close to what can be achieved by extensively hand-optimized versions, while still preserving most of the succinctness and high-level benefits of the original ADP language.

Sequence data is not the only type of data for which grammar-like dynamic programs are of interest. Inverse coupled rewrite systems (ICOREs) [4] allow the user to develop algorithms over both, sequence and tree-like data. While no implementation for these rewrite systems is available yet, they already simplify the initial development of algorithms. This is

[*]Correspondence: choener@bioinf.uni-leipzig.de
[1]Bioinformatics Group, Department of Computer Science, Universität Leipzig, Härtelstraße 16–18, D-04107 Leipzig, Germany
Full list of author information is available at the end of the article

important in particular for tree-like data. Their non-sequential nature considerably complicates these algorithms. The grammar underlying the alignment of ncRNA family models with `CMCompare` [5], which simultaneously recurses over two trees, may serve as an example for the practical complications. There are compelling reasons to use DP approaches in particular when more information than just a single optimal solution is of interest. DP over sequences and trees readily allows the enumeration of all optimal solutions, and it offers generic ways to systematically investigate suboptimal solutions and to compute the probabilities of certain sub-solutions. Classified dynamic programming [6], furthermore, enables the simultaneous calculation of solutions with different class features via the evaluation algebra instead of constructing different grammars for each class.

An important research goal in the area of dynamic programming algorithms is the development of a framework that makes it easy to implement complex dynamic programs by combining small, simple, and reusable components. A first step in this direction was the introduction of grammar products [7], which greatly simplifies the specification of algorithms for sequence alignments and related dynamic programming tasks that take multiple strings as input. Several straightforward questions, however, still remain unanswered.

An important example is the relationship of Forward/Backward (in the context of linear grammars) [8] and Inside/Outside (in the context of CFGs) [9] algorithms. So far, the two variants need to be developed and implemented independently of each other. The close structural relationship of the two types of recursion has of course been noticed and used explicitly to facilitate algorithm design. The idea of "reverting" the inside production rules was used explicitly to explain backtracing and outside recursions in [10, 11] for the RNA-RNA interaction problem and in [12] for RNA folding with pseudoknots, albeit without providing a general operational framework. In classical ADP the Inside algorithms are phrased as parsing an input string w.r.t. a given context free grammar. This is not possible in general for the Outside recursion because these operate, conceptually, on the complement of a substring. In some situations it is possible to rescue the ADP-style approach. For RNA folding, for example, Janssen [13] proposed to concatenate the suffix and the prefix in this order. The Outside recursion is then rephrased as a CFG on this modified string.

A second unsolved issue is that not all dynamic programming algorithms can be translated into the ADP framework in a straightforward manner. A classical example is the Travelling Salesman Problem (TSP). It is easily stated as follows: given a set X of cities and a matrix $d : X \times X \to \mathbb{R}_+$ of (not necessarily symmetric) distances between them, one looks for the tour (permutation) π on X that minimizes the tour length $f(\pi) := d_{\pi(n),\pi(1)} + \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}$. W.l.o.g., we may set $X = \{1, \ldots, n\}$ and anchor the starting point of a tour at $\pi(1) = 1$. The well-known (exponential-time) DP solution for the TSP [14, 15] operates on "sets with an interface" $[A, i]$ representing the set of all tours starting in $1 \in A$, then visiting all other cities in A exactly once and ending in $i \in A$. The length of the shortest path of this type is denoted by $f([A, i])$. For an optimal tour we have $f([X, i]) + f(\langle i, 1\rangle) \to \min$, where $f(\langle i, 1\rangle) = d_{1,i}$ is the length of the edge from i to 1. The values $f([A, i])$ satisfy the recursions

$$f([A, i]) = \min_{j \in A} f([A \setminus \{i\}, j]) + f(\langle j, i\rangle) \qquad (1)$$

since the shortest path through A to i must consist of a shortest path through A ending in some $j \in A$ and a final step from j to i.

A classical ADP formulation is impossible because the set A does not admit a string representation so that its subsets could be generated by a fixed set of productions. To split off a particular element $\{i\}$ from A, for example, one requires a specific production rule of the form $A \to (A \setminus \{i\}) \cup \{i\}$. This cannot be captured by a fixed CFG since the number of productions grows with the size of A.

Instead of relaxing the constraints on the number of productions we argue here that the solution to this conundrum can be resolved by a redefinition of the concept of parsing so that we can meaningfully write $A \to Ax$ for the decomposition of a (nonempty) set into a subset with cardinality one less and the excluded single element. This restores one of the main advantages of ADP, namely the possibility to describe the state space traversal without explicit representation of indices. At the same time we will see below that the same formalism also yields a completely mechanical way to construct Outside recursions from the Inside algorithm. To this end we first consider the conceptually simple case of 1-dimensional and 2-dimensional linear grammars on strings using HMMs and pairwise sequence alignments as example. We then proceed to RNA folding as an example of a non-trivial CFG. The final step is to introduce an ADP-style formalism for non-trivial set-like data structures. Up to this point we keep our discussion informal and ignore several technical details. In section 3 we will then follow up with a much more abstract and precise account. In section 4 we finally consider the probabilistic version of the shortest Hamiltonian path problem in the context of the early evolution of HOX gene clusters as a real-life application of our framework.

2 Case Studies

2.1 HMMs and the Forward/Backward Algorithms

A simple Hidden Markov Model (HMM) for detecting CpG islands in genomic DNA can specified as follows: (1) Each nucleotide position is contained either in a CpG island (state "+") or not (state "−"). (2) The probability that a nucleotide p follows q is given as a^{σ}_{pq} and differs between the two states $\sigma \in \{+,-\}$. Furthermore we require a probability to switch from + to − of $q^{\pm}$ and $q^{\mp}$, respectively. This yields transition probabilities $t^{++}_{i,i-1} = (1-q^{\pm})a^{+}_{x_i,x_{i-1}}$ and $t^{--}_{i,i-1} = (1-q^{\mp})a^{-}_{x_i,x_{i-1}}$ for the cases where the state remains unchanged + or − and $t^{+-}_{i,i-1} = q^{\mp}a^{-}_{x_i,x_{i-1}}$ and $t^{-+}_{i,i-1} = q^{\pm}a^{+}_{x_i,x_{i-1}}$ for the two possible state changes. Note that this formulation is much simpler than the usual HMM formalism since the emission probabilities are trivial here.

$$\begin{aligned} f^+[i] &= f^+[i-1]t^{++}_{i,i-1} + f^-[i-1]t^{+-}_{i,i-1} \\ f^-[i] &= f^-[i-1]t^{--}_{i,i-1} + f^+[i-1]t^{-+}_{i,i-1} \end{aligned} \tag{2}$$

The corresponding backward probabilities are

$$\begin{aligned} b^+[i] &= b^+[i+1]t^{++}_{i,i+1} + b^-[i+1]t^{+-}_{i,i+1} \\ b^-[i] &= b^-[i+1]t^{--}_{i,i+1} + b^+[i+1]t^{-+}_{i,i+1} \end{aligned} \tag{3}$$

This allows to compute $\mathbb{P}(i \in +) = f_+[i]b_+[i]$ and $\mathbb{P}(i \in -) = f_-[i]b_-[i]$.

In an ADP-style framework the forward recursion corresponds to the grammar with the productions

$$\begin{array}{rclclcl} S & \to & P & \mid & M & & \\ P & \to & Pc & \mid & Mc & \mid & \varepsilon \\ M & \to & Pc & \mid & Mc & \mid & \varepsilon \end{array} \tag{4}$$

Apart from the formal start symbol S, it describes the two states as P and M and the possible transitions. The latter are both associated with prefixes $[1..i]$ of the input strings up to some position i. The non-terminal P signifies that i has state +, while non-terminal M corresponds to the − state. The scoring of the productions $P \to Pc$, $P \to Mc$, etc., is relegated to a scoring algebra that encodes the multiplicativity of probabilities. Translated to recursion form, with indices $0 < i \leq n$ referring to positions in the input string and n denoting the length of the input, the forward recursions take on their usual form, see e.g. [16, p. 51ff]:

$$\begin{aligned} &S_n = P_n + M_n \\ &P_i = P_{i-1} \times c_i^{P\to Pc} + M_{i-1} \times c_i^{P\to Mc} + 0 \\ &M_i = P_{i-1} \times c_i^{M\to Pc} + M_{i-1} \times c_i^{M\to Mc} + 0 \\ &P_0 = 1 \qquad M_0 = 1 \end{aligned} \tag{5}$$

where $c_i^{P\to Pc} := (1-q^{\pm})a^{+}_{x_i,x_{i-1}}$, etc., are the tabulated parameters of the HMM. The initialization $P_0 = M_0 = 1$ follows as the conditional probability of ending in the ϵ state after having read all input. Note that the strucuture of the recursion (5) is completely determined by the productions in equ.(4).

The backward recursion corresponds to a traversal of the input by means of suffixes. To each forward prefix $[1..i]$ we have a matching suffix $[i..n]$, where n is the length of the input. This overlap of corresponding prefix and suffix is just one indication that we might want to modify how we interpret the grammar (4). The fact that the scoring function explicitly refers to transitions, i.e., pairs of consecutive positions gives another hint. In this alternative picture we think of P and M as prefixes in which the last position takes on the role of a boundary ∂M and ∂P. Now we can think of the corresponding suffixes as the complements w.r.t. the input, i.e., to "forward objects" P and M we associate "backward objects" P^* and M^* so that, in terms of the index sets to which they refer, we have $P \cup P^* = S = \{1 \dots n\}$, $M \cup M^* = S$, $P \cap P^* = \partial P$, and $M \cap M^* = \partial M$. Correspondingly, the terminals can be thought of as pairs of consecutive positions. This provides us with a mechanical way of scoring c as $c_i^{\pm}$ in the forward recursion and as $c_{i+1}^{\pm}$ in the backward recursion, since the terminal c defined for positions $(i-1, i)$ overlaps with the boundary of the forward objects P and M at $i-1$, while the one defined on $(i, i+1)$ overlaps on $i+1$ with the boundary of the backward objects P^* and M^*.

Now, the corresponding backward grammar is, and should be compared closely to its progenitor (equ. 4):

$$\begin{array}{rclclcl} \varepsilon^* & \to & P^* & \mid & M^* & & \\ P^* & \to & P^*c & \mid & M^*c & \mid & S^* \\ M^* & \to & P^*c & \mid & M^*c & \mid & S^* \end{array} \tag{6}$$

At first glance this notation looks awkward. One might have expected something like $P^* \to cP^*$. However, we will see below that for general CFGs the backward or outside objects P^* refer to the index set not covered by P. The notation P^*c can be interpreted as the insertion of c at the right hand end of the "hole", i.e., as a left extension of the suffix. For completeness we translate eq.(6) into the corresponding recursions

$$\begin{aligned} &\varepsilon^*_0 = P^*_0 + M^*_0 \qquad \texttt{(outside final result)} \\ &P^*_i = P^*_{i+1} \times c_{i+1}^{P^*\to P^*c} + M^*_{i+1} \times c_{i+1}^{P^*\to M^*c} + 0 \\ &M^*_i = P^*_{i+1} \times c_{i+1}^{M^*\to P^*c} + M^*_{i+1} \times c_{i+1}^{M^*\to M^*c} + 0 \\ &P^*_n = 1 \qquad M^*_n = 1 \end{aligned} \tag{7}$$

The *a posteriori* probability that sequence position i is in the + state is given by $P_i P^*_i$. In our framework, we obtain the complete list of these proabilities by using the probability scoring algebra and the formal production rule $S \to P^*P$ or, equivalently, $S \to PP^*$. Writing $S \to P^*P$ as a production rule from the start symbol ties P and P^* together to be complementary, rather than independent non-terminals following the forward and backward grammar. Note that symbolically $S \to P^*P$ is no longer a linear production. It becomes useful, however, in our formalism to use the notation of production rules to specify any kind of decomposition of a data object. In this setting $S \to P^*P$ does makes sense: it defines the list of all complementary pairs of inside and outside objects, i.e., it serves as implicit specification of the outside object P^* to P. We will formally define this construct in Sect. 3.

2.2 Prefix and Suffix Style Pairwise Sequence Alignments

An analogous construction pertains to multiple sequence alignment. The only difference there is that now we operate simultaneously on multiple input tapes. For simplicity of exposition we consider only the pairwise alignment problem. Let us start with the well-known Needleman-Wunsch (NW) algorithm [17]. Starting from an empty alignment, we can think of it as extending an alignment A by either a (mis)match, an insertion, or a deletion. In grammar form this can be written as

$$S \to A \qquad A \to A\binom{u}{v} \mid A\binom{u}{-} \mid A\binom{-}{v} \mid \binom{\varepsilon}{\varepsilon} \tag{8}$$

In contrast to the HMM example above, it is convenient here to interpret the string pairs A with an empty boundary: if A refers to the pair $[1..i, 1..j]$ then A^* refers to $[i+1..n, j+1..m]$ where n and m, resp. are the lengths of the input strings. The formal outside derivation, in terms of suffixes, of the NW algorithm is:

$$\begin{aligned} &\binom{\varepsilon}{\varepsilon}^* \to A^* \\ &A^* \to A^*\binom{u}{v} \mid A^*\binom{u}{-} \mid A^*\binom{-}{v} \mid S^* \end{aligned} \tag{9}$$

As for the HMM we think of A^* as a representation of the hole that is left over by A, and we read $A^*\binom{u}{-}$ as "fill $\binom{u}{-}$ into the hole A^* at its r.h.s. end. Clearly $S \to AA^*$ and $S \to A^*A$ refer to the complete global alignments with all possible "splitting constraints".

The outside productions (9) *look like* the suffix version of the NW algorithm. Writing the decomposition with full index information, however, shows that there is a suble difference: $A_{ij} \mapsto A_{i-1j-1}\binom{i}{j}$ transforms to $A^*_{ij} \mapsto A^*_{i+1,j+1}\binom{i+1}{j+1}$, etc. This highlights the interpretation that A^* refers to the "hole" extending to the right from the positions *after* i and j, i.e., not including i and j itself. While this make little formal difference for the NW algorithm, it does have an important impact in the more complex case of Gotoh's algorithm for affine gap costs [18].

The different scoring of gap "opening" and "extension" implies that the gap-status at the end of a partial alignment must be known. To this end the CFG uses three non-terminals M, D, and I depending on whether the r.h.s. end of the alignment is a match state, a gap in the second sequence, or a gap in the first sequence. We ignore the issues of start (S) and stop ($\binom{\varepsilon}{\varepsilon}$) symbols for the moment and return to them in Sec. 3 below in a more systematic manner. The productions of the "body" of the recursions are of the form

$$\begin{array}{lcllll} S & \to & M & \mid D & \mid I & \\ M & \to & M\binom{u}{v} & \mid D\binom{u}{v} & \mid I\binom{u}{v} & \mid \binom{\varepsilon}{\varepsilon} \\ D & \to & M\binom{u}{-} & \mid D\binom{u}{.} & \mid I\binom{u}{-} & \mid \binom{\varepsilon}{\varepsilon} \\ I & \to & M\binom{-}{v} & \mid D\binom{-}{v} & \mid I\binom{.}{v} & \mid \binom{\varepsilon}{\varepsilon} \end{array} \tag{10}$$

where u and v denote terminal symbols. '$-$' corresponds to gap opening, while '.' denotes the (differently scored) gap extension.

The interpretation of the non-terminals M, D, and I is determined by the last column of the prefix alignment: it ends in a (mis)match, a deletion, or an insertion, respectively. In contrast to the HMM example of the previous section we do not score transitions here. Thus we interpret the non-terminals as boundary-free. Hence M^* becomes a suffix object complementing a prefix alignment that ends in a (mis)match. Note that this does *not* mean that M^* itself ends in a (mis)match. Because M, and thus M^* are boundary-free, the corresponding alignments do not overlap. As a consequence, their scores can be added. This property is required for the evaluation algebras to behave properly.

Transforming the linear grammar eq.(10) into its outside recursions yields

$$\begin{array}{lcllll} \binom{\varepsilon}{\varepsilon}^* & \to & M^* & \mid D^* & \mid I^* & \\ M^* & \to & M^*\binom{u}{v} & \mid D^*\binom{u}{-} & \mid I^*\binom{-}{v} & \mid S^* \\ D^* & \to & M^*\binom{u}{v} & \mid D^*\binom{u}{.} & \mid I^*\binom{-}{v} & \mid S^* \\ I^* & \to & M^*\binom{u}{v} & \mid D^*\binom{u}{-} & \mid I^*\binom{.}{v} & \mid S^* \end{array} \tag{11}$$

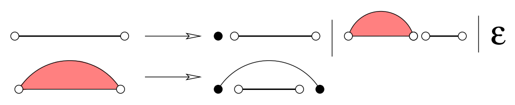
A first glance, this grammar looks odd. It is not the grammar for the suffix-version of Gotoh's algorithm. Instead, it refers to a rather unusual way of solving the affine gap cost problem. Here the distinction is not made between opening or extending a gap, but rather between closing or extending it. The nonterminals on the r.h.s. of the rule thus refer to the type of alignment that is reached *after* extending the one on the l.h.s. of the rule by the terminal symbol appearing on the r.h.s. Since our forward recursion (10) is set up to separately score gap opening, i.e., the left-most gapped position in the alignment, the same must be true for the backward recursion. Since it proceeds from right to left on the input string, we naturally arrive at the algorithmic variant that scores gap closing separately. The corresponding non-terminals therefore depend on how the alignment is continued in the subsequent step.

On a more technical note, the conversion of (10) to (11) does *not* break the signature type isomorphism between inside and outside variant. Where previously the rule $D \to I(\genfrac{}{}{0pt}{}{u}{-})$ makes use of an attribute function with type $\Gamma \times \binom{Char}{-} \to \Gamma$ for evaluation, this type now corresponds to the rule $I^* \to D^*(\genfrac{}{}{0pt}{}{u}{-})$. Here Γ is the type of the evaluated parse, e.g. a probability for SCFGs or an energy or a partition function for RNA folding.

One obtains the probability of matching each pair $\binom{i}{j}$ via $S \to MM^* \equiv M^*M$. Each M_{ij} indicates an alignment ending in a match with indices $\binom{i}{j}$, while M^*_{ij} yields alignments where the matching characters at $\binom{i}{j}$ "have just been transitioned from".

2.3 Inside and Outside: RNA folding

State-of-the-art RNA folding programs (as implemented e.g. in the `ViennaRNA` package [19]) incorporate the nearest-neighbour model. For the purpose of a more compact presentation, we restrict ourselves here to the much simpler model



(12)

The full model [20] amounts to a more complicated decomposition of secondary structure enclosed by a base pair (2nd decomposition). In a more conventional, but mnemonically less pleasing form the productions in equ.(12) read

$$S \to U \qquad U \to cU \mid BU \mid \epsilon \qquad B \to cUc' \quad (13)$$

This is a conventional CFG acting on the input string, here an RNA sequence. As usual we write $c \ldots c'$ to mean all 6 combinations of canonical base pairs **gc**, **cg**, **au**, **ua**, **gu**, and **ug**. In terms of recursions with explicit indices, its interpretation is

$$\begin{array}{rcl} S_{1,N} & \mapsto & U_{1,N} \\ U_{ij} & \mapsto & \epsilon_{i>j} \quad \texttt{empty parse} \\ U_{ij} & \mapsto & c_i U_{i+1,j} \\ B_{ij} & \mapsto & c_i U_{i+1,j-1} c'_j \end{array} \qquad U_{ij} \mapsto B_{ik} U_{k+1,j} \quad (14)$$

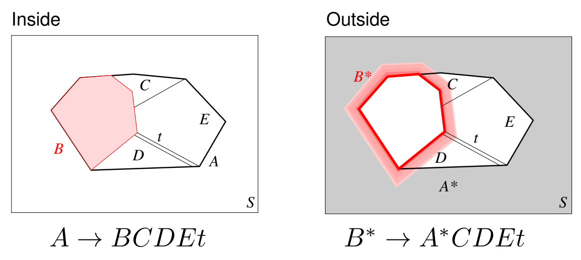


Figure 1 Conversion of Inside productions into decompositions of the corresponding Outside objects. A production rule $A \to \ldots$ describes in which ways A can be partitioned, here into 5 parts B, C, D, E, and t. The corresponding outside objects A^* and B^* are the complements of A and B w.r.t. to input, represented by the box S. A rule $B^* \to \ldots$ describes the decomposition of B^*, i.e., $S \setminus B$. Borrowing from $A \to BCDEt$ we see that B^* consists of complement A^* of A as well as the pars of A with the exception of B itself.

The indices here explicitly designate a substring of the input to which a particular non-terminal or terminal symbol refers.

The non-terminal S, which refers to unconstrained secondary structures, has an empty boundary. In contrast, it is natural to think about B_{ij} as having the closing base pair $\langle i,j \rangle$ as its boundary. The reason is that the standard energy model for RNAs in general evaluates the "loop" enclosed by $\langle i,j \rangle$ rather than the pair itself. This is also true for corresponding outside objects, i.e., $\langle i,j \rangle$ naturally contributes both inside and outside "loops" that it delimits. In the natural way to define outside objects this is different for S and B. The complement of S_{ij} is $S^*_{i-1,j+1}$ and refers to secondary structures on $[1..i-1] \cup [j+1..n]$. In contrast, the complement of B_{ij} is B^*_{ij}, referring to secondary structures on the union $[1..i] \cup [j..n]$. Thus $S \cap S^* = \emptyset$, while $B \cap B^*$ is the common enclosing base pair.

In order to understand how the inside grammar (14) gives rise to recursions for the outside variables S^* and B^* we first consider the conceptual picture illustrated in Fig. 1. Since a production corresponds to a decomposition of an object in smaller constituents, we may pick one of these parts and ask for a decomposition of its complement. The complement of course is formed with respect to some "ground set", in our case the complete input.

This idea is easily made precise for an arbitrary CFG. Consider a derivation of the form $A \to \alpha B \beta$, where B is a non-terminal and α, β are strings of terminals and non-terminals. Here A is decomposed into B as well as the rest $\alpha \cup \beta$. Looking at the complements, therefore, B^* consists of A^* and the rest $\alpha \cup \beta$. Since A and B are intervals in a CFG setting, their complements are disjoint unions of a prefix and a suffix of the input.

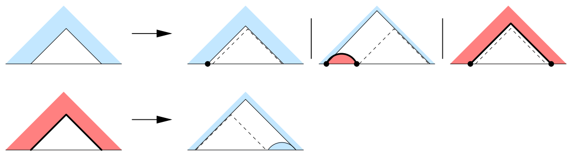
Thus $A \to \alpha B \beta$ transforms to

$$A \to \alpha B \beta \quad \rightsquigarrow \quad B^* \to \alpha A^* \beta \tag{15}$$

where the string of forward (non)terminals α is filled from the left into the hole of A^* and β is filled in from the right. This is always well-defined because the outside rules always contain exactly one outside non-terminal on both the l.h.s. and the r.h.s. of the derived rule. For an inside production $A \to \gamma$ we obtain one outside production for every non-terminal $B \in \gamma$. Thus we have to split $\gamma = \alpha B \beta$ for all non-terminals $B \in \gamma$. This can easily be achieved in a completely mechanical way. For the RNA example this yields

$$\begin{aligned} &\varepsilon^* \to U^* \\ &U^* \to cU^* \mid BU^* \mid cB^*c' \mid S^* \\ &B^* \to U^*U \end{aligned} \tag{16}$$

In diagrammatic form this can be written in the following way


(17)

It is instructive to translate this outside grammar-style into a more conventional recursive form that explicitly exposes the indices:

$$\begin{aligned} &U^*_{ij} \mapsto c_{i-1}U^*_{i-1,j} \\ &U^*_{ij} \mapsto c_{i-1}B^*_{i-1,j+1}c'_{j+1} \\ &U^*_{ij} \mapsto B_{k,i-1}U^*_{k,j} && k < i \\ &B^*_{ij} \mapsto U^*_{i,k}U_{j+1,k} && k > j \\ &U^*_{ij} \mapsto \epsilon_{i=1,j=n} \end{aligned} \tag{18}$$

We can now derive McCaskill's algorithm [21] for computing the base pairing probabilities by (1) writing down the inside grammar (see e.g. [22] for several variants), (2) specifying the evaluation algebra for the partition function (see sect. 3), (3) generating the outside recursions, and (4) producing the list $S \to BB^*$.

2.4 Shortest Hamiltonian Paths

We now leave the realm of classical ADP behind and consider dynamic programming algorithms on unordered data structures. This most clearly requires us to rethink what we mean by a "grammar" and by "parsing". Since shortest Hamiltonian paths play a key role in the show-case example in section 4 we introduce them here as an illustrative example.

Shortest Hamiltonian Path (SHP). Given a graph G with vertex set V and edge set E and a dissimilarity matrix $d : E \to \mathbb{R}_+$ the task is to find a path π in V that runs through every vertex and minimizes the total length $\ell(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)}$.

SHP is a well known NP-complete combinatorial optimization problem. It can be solved exactly by a simple DP algorithm [14, 15], which of course in general has exponential runtime. Denote by $[i, A, j]$ with $i, j \in A \subseteq V$ the set of all Hamiltonian paths through a subset A that have i and j as its endpoints. For every $k \in A \setminus \{i, j\}$ we can decompose $[i, A, j]$ into the edge $\langle k, j\rangle$ and the set $[i, A, k]$ of shorter Hamiltonian paths. We can write this decomposition in the form

$$A \to Av \tag{19}$$

in complete analogy to a linear grammar. The point of this section is that we can make this analogy precise and useful.

Let us first consider this rule for an arbitrary set. Then $A \to Av$ tells us to split off a single element (atom) from A. On string data structures there is essentially only a single way of doing this, namely to remove a single-character suffix. Removal of a prefix would be encoded by $A \to vA$, i.e. a distinct production. On sets, we now have $|A|$ possibilities, i.e., we obtain a list of possible decompositions. Since the underlying data structure has no intrinsic order, the productions $A \to Av$ and $A \to vA$ are of course equivalent. This is not different from CFGs, in fact: A production of the form $A \to BC$ returns $|A| + 1$ partitionings of A into a prefix B and a suffix C. Of course $A \to BC$ also makes perfect sense for sets: B and C now form the bipartitions of A, i.e., there are $2^{|A|}$ alternative decompositions. The only difference between strings and sets thus is the number of alternative parses.

In the SHP example there is a further complication: we have to keep track of the end points i and j. Instead of regular sets, we thus have an additional "punctuation" structure that defines a start and end point. The parsing rule $A \to Av$ now has to know that (i) the end point has to be split off, and in doing so, (ii) a new endpoint distinct from the startpoint has to be determined so that we obtain again a properly punctuated set. Furthermore it becomes the parser's job to know that (iii) the terminal v is the connection between new and old endpoint, rather than just a split-off vertex. We note in passing that simply re-interpreting the punctuated sets A as connected components so that neither end point is a cut vertex of the input graph may increase practical efficiency since it prunes early those subsets through which no path connecting the end points can be constructed. Whether we want to think of the start-

and endpoints as distinct features, or whether either one can be used to decompose the set, is a matter of modelling and defines two distinct data types.

Formally, furthermore, we see that there is a second type of decomposition operations that seems useful in general ADP. We may simply write

$$V \to A \tag{20}$$

and assume that our machinery knows that V is an unstructured set, while A is a set with start and endpoint. The trivial-looking rule therefore provides a list of $|V|(|V|-1)$ or $|V|(|V|-1)/2$ punctuated sets, depending on whether start and end are distinguished or not.

Of course we can immediately construct A^* as a complementary punctuated set with the same endpoint, i.e., so that $[i, A, j][j, A^*, k]$ overlap in the common point j. We then have

$$A^* \to A^* v \tag{21}$$

as the corresponding grammar for the outside objects.

For the Hamiltonian paths through a particular adjacency (terminal) v we can now write $V \to AvA^*$, i.e., this is a top-level decomposition of the start symbol, i.e., acting on the input string. We simply have to use as scoring algebra the multiplication of partition functions from A and A^* (as $S \setminus A$) and fix the Boltzmann-weights $Z(\langle i,j\rangle) = \exp(-\alpha d_{ij})$ for the terminals. This computes the *a posteriori* probability of observing an adjacency $i \sim j$ in the path with fixed endpoints p and q, i.e.,

$$P(i \sim j|p,q) = \frac{1}{Z(S_{pq})} \sum_{A \subset S} Z([p,A,i]) Z(\langle i,j\rangle) Z([j, S\setminus A, q])\,. \tag{22}$$

in the index-based notation. Similarly, $P(\text{ends}{=}p,q) = Z(S_{pq})/Z(S)$ provides us with the probabilities that the path ends in p and q and $P(\text{end=p}) = \sum_q P(\text{ends} = p,q)$ measures how frequently we expect p to be an end point of a path. We make use here of the distinction between the start symbol S, which refers to a set without boundary, and S_{pq}, a set $[p, A, q]$ with two boundary points p and q defining the end points of the paths running through it. Thus $S \to S_{pq}$ with "sum" as choice function and the identity attribute function (attribute functions of an algebra evaluate individual parses) yields $Z(S)$, summing over all (p,q). On the other hand, $S \to S_{pq}$ with the identity choice function and attribute function $\lambda z.\frac{z}{Z(S)}$ (where $\lambda z.\mathcal{B}$ denotes an anonymous function with body $\mathcal{B}$ expecting z as its single argument) returns a list of all (p,q) start/end points, together with the probabilities that these points are start and end point. To obtain the end probabilities in our framework we need an additional type of non-terminals, say S_p, that have only a single point as boundary, i.e., it refers to sets of the form $[p, A]$. The production $S \to S_p$ with the identity choice and attribute $\lambda z.\frac{z}{Z(S)}$ then yields the desired end probabilities. $S_p \to S_{pq}$ folds over all S_{ip} and S_{pi} for all i, as we now do not distinguish between a 'starting' and 'ending' point in a path for S_p.

3 The formal framework

The key ingredient in our approach is to generalize grammars to decomposition schemes of a wide range of data models by redefining what exactly parsers do. Let us start with making explicit how this works in classical ADP, i.e., for (context free) grammars on strings:

1 Each (non)terminal corresponds to a substring.
2 Each terminal symbol matches a single character of the input string.
3 Each production defines a (list of) partitions. More precisely, the substrings corresponding to the r.h.s. of the production partition the substring corresponding to the single non-terminal on its l.h.s.
4 The partition is order preserving, i.e., the sequence of symbols on the r.h.s. matches the order of the corresponding substrings on the input string.

We have seen that it may be convenient already in the realm of strings to give up some of these requirements e.g. to treat problems in which terminals are naturally interpreted as transitions between adjacent positions as in the HMM case.

To formalize this we introduce objects with boundaries and allow that objects on the r.h.s. of a product overlap in their boundaries in a certain way. To this end we define for each object A its boundary ∂A and its interior $\text{int}(A) := A \setminus \partial A$. An object together with its boundary is denoted by $[A, \partial A]$. We also allow terminals to match more than an atomic constituent of the input data structure. An example are the pairs of adjacent characters in the HMM case and the edges of input graphs in the Hamiltonian paths. The productions take on the form of decomposition rules

$$[A, \partial A] \to \bigcup_i [A_i, \partial A_i] \tag{23}$$

for which we require the following properties:

(C1) $\bigcup_i A_i = A$, i.e., the decomposition products of A form a covering of A.

(C2) $\operatorname{int} A_i \cap \operatorname{int} A_j \neq \emptyset$ implies $i = j$, i.e., the interiors of the parts are disjoint.

(C3) $\operatorname{int} A_i \subseteq \operatorname{int} A$, i.e., the interiors behave like isotonic functions.

Note that these axioms recover the partition-style parsing if all data objects have empty boundaries. In this case (C3) follows from (C2). Whether we treat the $\bigcup_i [A_i, \partial A_i]$ as an ordered list or as a multiset (or as something inbetween) depends on the intrinsic internal order structure of A. Here we have encountered only total orders (on strings) and anti-chains (for sets). Non-trivial partial orders however may become important when dealing with tree structures.

The parsers can infer much of the necessary index handling from considering meta-rules for handling adjacent boundaries. For instance, it will be useful in many cases to declare that boundaries of adjacent objects can overlap only if they coincide. In the RNA example enclosing base pairs appear as object boundaries. Semantically, it make no sense to allow overlap of base pairs at one end but not at the other. In other cases, however, it is useful to require only that $\partial A_i \subseteq A_j$ or *vice versa*. This is the case in the HMM and SHP example, where we might want to interpret the terminals as transitions and edges as having empty interior and thus consisting of boundary only. DP algorithms where the index arithmetic in the decompositions is even more complex for instance appear in `RNAwolf` [23] and in the context of the coloring problems associated with RNA design in [24].

As we use complementarity w.r.t. the input to define the outside objects we have

$$S \to [A, \partial A][A^*, \partial A^*]\,. \tag{24}$$

since the start symbol S refers to the unprocessed, complete input. By construction, therefore, $\partial A = \partial A^*$, and A and A^* overlap at the boundary. The same complementarity is the basis for deriving the outside recursion in a well-defined manner from the inside recursion using equ.(15) in the ordered case or even simpler

$$A \to \alpha B \gamma \quad \rightsquigarrow \quad B^* \to \alpha A^* \gamma \tag{25}$$

where γ is a set of terminals and non-terminals. By construction there is exactly one syntactic outside variable on the r.h.s. of an outside production rule. All other symbols on the r.h.s. are either terminal symbols or inside symbols. From the perspective of the outside variables, they behave as "syntactic terminals", i.e., in a combined inside/outside grammar none of their derivations ever reaches an outside variable. As an immediate consequence we conclude that the outside grammar is bi-linear (and even linear in the unordered case) in its outside syntactic variables. Given a description of parsing we can now use the conventional ADP framework.

Start symbols, stop symbols, and normalized grammars

The start symbol S and stop symbol ε are complementary in a natural manner: S refers to the complete, unprocessed input, while ε recognizes that the input has been used up and there is nothing left to parse. Thus $S \cong \varepsilon^*$ and $\varepsilon \cong S^*$. More precisely, each inside rule of the form $S \to A$ has a corresponding outside rule $A^* \to \sigma^*$, and each inside rule $A \to \varepsilon$ yields translates into the outside rule $E^* \to A^*$. We say that a grammar is *normalized* if

(1) Every production rule $S \to \alpha$ with the start symbol S on the l.h.s. has a single non-terminal (or syntactic variable) on the r.h.s.

(2) All production rules with only terminals on the r.h.s. have just a single ε (and no other symbol) on the r.h.s.

While it is not strictly necessary to work with normalized grammars, they are practically convenient because normalization guarantees that start and stop rules that isomorphic evaluation function types in their respective inside and outside version. It is easy to see that if $\mathcal{G}$ is normalized, then its outside variant $\mathcal{G}^*$ is also normalized.

As an illustration we complete here the outside recursions for Gotoh's alignment algorithm. Recalling eqns. (10) and (11) we first have to explain the rules for the outside start symbol. From the inside rules $M \to \varepsilon$, $D \to \varepsilon$, and $D \to \varepsilon$ we obtain the expected productions $E^* \to M^* \mid D^* \mid I^*$. Furthermore, $S \to M \mid D \mid I$ yields the termination rules $M^* \to \sigma^*$, $D^* \to \sigma^*$, and $I^* \to \sigma^*$. The final outside variant of Gotohs algorithm now reads:

$$\begin{array}{rcl|c|c|c}
E^* & \to & M^* & D^* & I^* & \\
M^* & \to & M^*\binom{u}{v} & D^*\binom{u}{-} & I^*\binom{-}{v} & \binom{\sigma}{\sigma}^* \\
D^* & \to & M^*\binom{u}{v} & D^*\binom{u}{\cdot} & I^*\binom{-}{v} & \binom{\sigma}{\sigma}^* \\
I^* & \to & M^*\binom{u}{v} & D^*\binom{u}{-} & I^*\binom{\cdot}{v} & \binom{\sigma}{\sigma}^*
\end{array} \tag{26}$$

In the context of multi-dimensional grammars for alignments we have to deal with gap symbols referring to an empty input on one or more input tapes. Although gaps are superficially similar to stop symbols they only appear in the context of actually parsing an input symbol, albeit on another tape, they are handled just like any other character-parsing terminal. In particular they do not give rise to a start symbol in the outside grammar.

Combining Inside and Outside variables: *a posteriori* probabilities

ADP grammars come with a signature that describes the types of the attribute functions attached to each production rule. One of the fringe benefits of constructing the outside grammar automatically according to equ.(15) or equ.(25) is that the inside and outside grammars are guaranteed to be isomorphic with respect to their signature. This, in turn, simplifies re-use of evaluation algebras between inside and outside.

The formal production $S \rightarrow P^*P \equiv PP^*$, as an inside rule, states that parses p^* should be combined with corresponding parses p. This is, however not a normal context-free rule. If P and its outside complement P^* come from a linear grammar with a set-based index type, then the intuition is correct. For a linear grammar on a string index type, this looks intuitively correct, but the underlying index type does not admit a context-free grammar at all as linear grammars have a fixed left or right end point for each sub-parse. This issue can be resolved by observing that $S \rightarrow PP^*$ in an inside context translates into generating all possibilities of splitting (the index representation of) the complete input into an inside and an outside part. For linear string grammars there are the $O(n)$ ways to split $0 \leq k \leq n$ at k; for string CFGs there are $O(n^2)$ ways to split $0 \leq k \leq l \leq n$ at (k,l), and linear set grammars yield $O(2^n)$ different split points of a set with n bit. For multi-tape grammars, the behaviour follows in an analog fashion.

The production $S \rightarrow PP^*$ requires an attribute function evaluating each parse $(p,p^*) \in P,P^*$, and a choice function. The evaluation algebra for probabilities or partition functions (which are essentially unnormalized probabilities) comprises multiplication for terms appearing in decompositions and addition for alternative productions from the same non-terminal. For the terminals, score values are tabulated as parameters. In the case of RNA folding, these are the Boltzmann factors $\exp(-E(t)/RT)$ of the energies $E(t)$ associated with the terminal t. For the RNA toy model of Sec. 2.3 we have $E(t) = 1$ for a base pair terminal and $E(t) = 0$ for an unpaired terminal. In the more realistic setting, the loop energies of the Turner model are used. The practical evaluation will typically be along the lines of $\lambda pq.\frac{p \times q}{Z}$ to yield the probability, where the normalization constant Z is obtained by evaluating the start nonterminal.

4 Application: Shortest Hamiltonian Paths and Gene Cluster Histories

Local duplication of DNA segments via unequal crossover is the most plausible mechanism for the emergence and expansion of local clusters of evolutionary related genes. Although there are polynomial-time algorithms to reconstruct duplication trees from pairwise evolutionary distance data [25] this approach often fails to resolve the ancient history of gene clusters. The reason is the limited amount of phylogenetic information in a single gene. The situation is often aggravated by the extreme time scales leading to a decay of the phylogenetic signal so that only a few, very well-conserved sequence domains can be compared. A large number of trees then fits the data almost equally well. A meaningful analysis of the phylogeny thus must resort to some form of summary information that is less detailed than a fully resolved duplication tree. In the absence of genome rearrangements, and if duplication events are restricted to copying single genes to adjacent positions, we expect genetic distance to vary monotonically with genomic distance, i.e., we expect – at least approximately – to have $d_{ik} \geq \max(d_{ij}, d_{jk})$ whenever gene j lies between i and k on the genome. The same is true if gene duplications arise by unequal crossover and subsequent divergence rates are comparable. This so-called Robinson property ensures that a shortest Hamiltonian path through the genetic distance matrix conforms to the linear arrangement of the genes on the genome [26]. A mathematically more precise exposition of the role of short Hamiltonian paths in clusters of paralogous genes can be found in a forthcoming manuscript [27].

The same high noise level that suggests to avoid duplication trees should also make us distrust the shortest path. More robust results can be expected by considering the information on the ensemble of all Hamiltonian paths. We therefore compute the probabilities $P(i \sim j)$ of the individual adjacencies assuming a Boltzmann weighting $p(\pi) \propto \exp(-\ell(\pi)/RT)$ of the Hamiltonian paths π. The parameter T is a fictitious temperature governing the relative importance of short *versus* long paths π. For $T \rightarrow 0$ we focus on the (co)optimal paths only, while $T \rightarrow \infty$ leads to a uniform distribution of adjacencies. The normalization constant is conveniently set to $R = (n-1)\overline{d}$, where $\overline{d}$ is the average of the genetic distance between genes. The path length $\ell(\pi)$ plays the role of the energy in the partition function of RNA secondary structures and of the dissimilarity score in probabilistic alignment algorithms. As we have seen in sect. 2.4, the The ADP-style framework provides us with an easy and efficient way to compute the probabilites $P(i \sim j)$ of adjacencies along short Hamiltonian paths and the probabilities $P(\text{end} = i)$ that gene i is the endpoint of a short path. In intact clusters we expect that the ends of genomic cluster also appear as the most probable ends of the Hamiltonian paths. High probabilities in the interior, by contrast, are a good indicator of rearrangments.

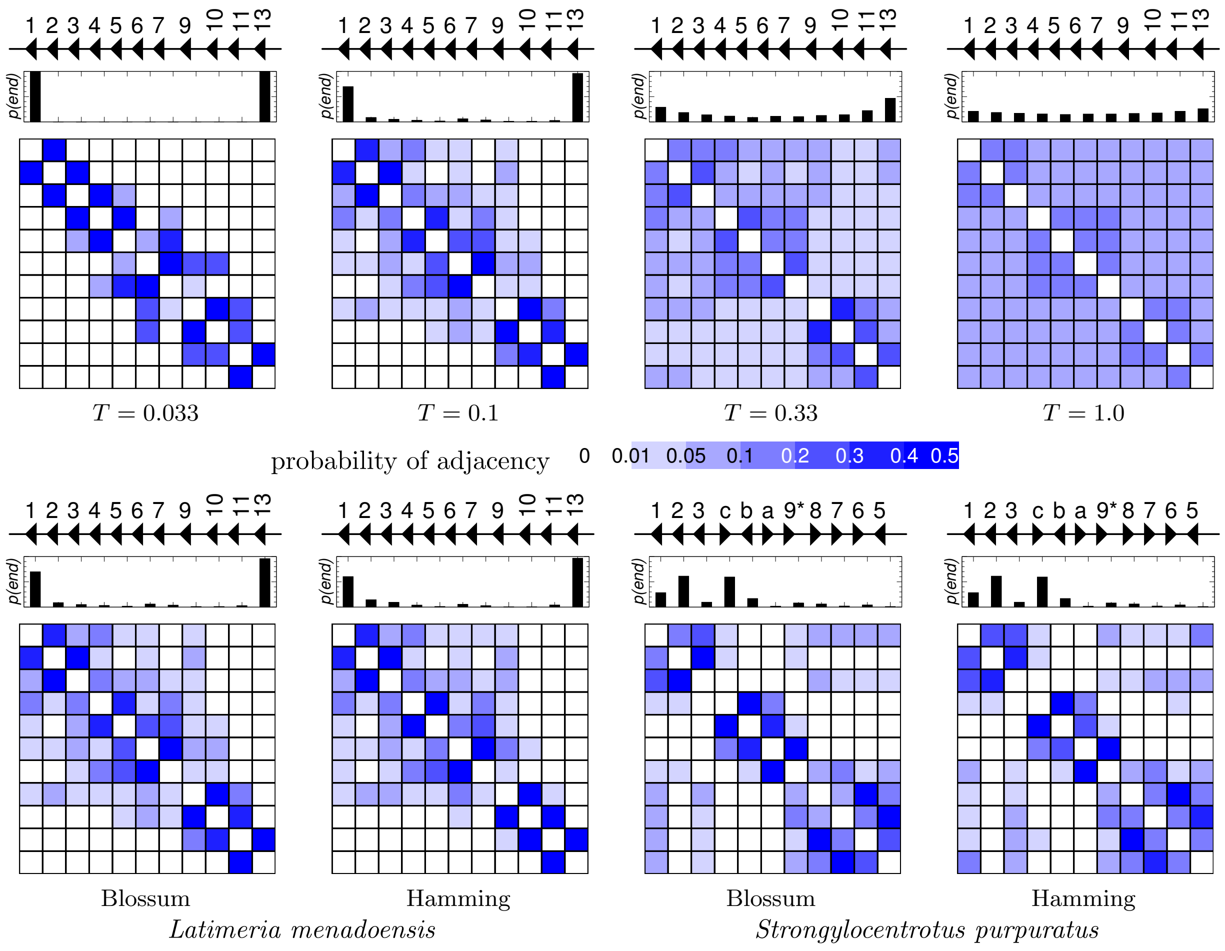


Figure 2 Analysis of two Hox gene cluster in terms of shortest Hamiltonian paths w.r.t. genetic distance. Small diagrams indicate the genomic order and reading direction of the genes (black triangles). The small histograms show the probability that a gene is endpoint of a Hamiltonian path, the square panels below display the posterior probabilities of adjacencies of Hox genes along shortest Hamiltonian paths w.r.t. to genetic distance. **Top:** effect of the temperature parameter T for distances between *Latimeria menadoensis* homeobox sequences. **Below:** Comparison of adjacencies for two different metrics (Hamming distance, and BLOSSUM45 derived dissimilarities) in *L. menadoensis* (left) and *S. purpuratus* for $T = 0.1$ to emphasize the structure of the ambiguities. Gene names are abbreviated 1-13 = HoxA1-HoxA13 for *L. menadoensis* and 1-8 = HoxA1-HoxA8, 9* = HoxA9/10, a-c = HoxA11/13a-HoxA11/13c for *S. purpuratus*.

Hox genes are ancient regulators originating from a single Hox gene in the metazoan ancestor. Over the course of animal evolution the Hox cluster gradually expanded to 14 genes in the vertebrate ancestor [28]. Timing and positioning of Hox gene expression along the body axis of an embryo is co-linear with the genomic arrangement in most species. Only the 60 amino acids of the so-called homeodomain can be reliably compared at the extreme evolutionary distances involved in the evolution of the Hox system. We quantitatively measure the genetic distance of the homeodomain sequences either using the Hamming distance, i.e. the number of different amino-acids, or the transformation $d_{ab} = s(a, a) + s(b, b) - 2s(a, b)$ of the BLOSSUM45 similarity scores.

We analyzed here the Hox A cluster of *Latimeria menadoensis* (famous as a particularly slowly evolving "living fossil"), which has sufferered the fewest gene losses among vertebrates. The 11 HoxA genes are arranged in the same order and orientation reflecting the gene order of the vertebrate ancestor: HoxA13, HoxA11 to HoxA9 and HoxA7 to HoxA1. In contrast, the Hox cluster of the sea urchin *Strongylocentrus purpuratus* has undergone fairly recent rearrangements of its gene order [29]. The putative ancestral cluster most likely had three anterior, five middle and one to five posterior genes. The exact number is not known because the time point of the posterior expansion is uncertain. The gene set of *S. purpuratus* is reminiscent of the ancestral configuration.

However, it reveals a gene order wherein the anterior genes (Hox1, Hox2 and Hox3) lie nearest to the posterior genes (Hox11/13c, Hox11/13b, Hox11/13a and Hox9/10), see Fig. 2. Several rearrangement schemes have been proposed, a minimum of one translocation, two gene inversions and the loss of Hox4 is required to reach the current configuration. Fig. 2 shows the posterior probabilities of adjacencies. Both, the coelacanth and the sea urchin examples reflect the well-known clustering into anterior (Hox1-3), middle group genes (Hox4-8), and posterior ones (Hox9-13). The shortest Hamiltonian paths in *L. menadoensis* connect the Hox genes in their genomic order. The high endpoint probability values $p(\text{hoxA1}, T = 0.1) = 0.699$ and $p(\text{hoxA13}, T = 0.1) = 0.960$ correctly identify HoxA1 and HoxA13 as cluster endpoints. In the sea urchin, however, we see adjacencies connecting the anterior subcluster (Hox1-3) with the genomic end of the cluster, i.e., the middle group genes (Hox8-Hox5). This is indicative of the recent cluster rearrangement. With a factor of about 2 the endpoint probability value favors hoxA2 over hoxA1 (the true endpoint). Note also that independent posterior expansion in Chordata (such as *L. menadoensis*) and Ambulacraria (such *S. purpuratus*) has lead to paralogs with greater genetic distance than observed among the anterior and middle group genes.

5 Discussion

We have taken here the first step towards extending algebraic dynamic programming (ADP) beyond the realm of string-like data structures. Our focus is an efficient, yet notationally friendly way to treat DP on unordered sets. As a showcase application we used ADP on sets to demonstrate that statistics over Hamiltonian paths can be computed efficiently as means of analyzing the ancient evolution of gene clusters. This extension of ADP builds on the same foundation (namely `ADPfusion` [2]) as our grammar product formalism [7, 30]. The key idea is to redefine the rules of parsing to match the natural subdivisions of the data type that now may be much more general than strings. In the case of sets, these are bipartitions and the splitting of individual elements, rather than the subdivision of an interval or the removal of a boundary element that are at the heart of string grammars. A particularly useful feature of our work is the ADP-style implementation and a principled approach to constructing outside algorithms, which is a rather straightforward consequence of defining the complement of a substructure relative to the input data object. There are several advantages to this approach:

- One cannot forget contributions to outside recursions. Such missing rules render the algorithm invalid, sometimes in non-obvious ways. This is of particular relevance for complex grammars and when existing algorithms are to be modified.
- Together with the `ADPfusion` framework the most annoying type of bugs in practical implementations of DP, namely index errors, can be avoided altogether because all index arithmetic is implicit and hidden completely from the user.
- Outside grammar construction is independent of syntactic variable and terminal types. As long as the abstract grammar is a context-free grammar, an outside version can be constructed.
- Our mechanistic construction interacts smoothly with other systems that automate creation of formal grammars, e.g. grammar products [7].

Our current framework still lacks generality and completeness in several respects. It is evident from our example above that data objects of different types can be obtained in the decomposition. For these, parsing may then mean different, type-dependent things. For instance, in the context of forest alignment and forest editing, reviewed in [31], it may be useful to distinguish trees from general forests. This suggests the possibility to develop an algebraic formalism of parsing/decomposition for complex data objects and thus an even higher-level way of specifying the intricacies of parsing schemes underlying DP algorithms. McBride's notion of a derivative operator acting on data types [32] appears to be a relevant starting point in this direction, although it does not seem to be directly applicable.

Although our present framework requires that parsing methods have to be specified for novel data types such as the punctuated sets of Section 2.4, this has to be done only once and can reused without additional overhead for all DP scenarios on the same data types. In particular, our system already handles all CFGs (and thereby also all linear grammars) on either strings or (punctuated) sets and automatically provides the associated outside algorithms. The high-level framework described here does not require much of a compromise in terms of computational efficiency. While we have to accept a decrease in theoretical performance by a moderate constant factor the gains in ease of algorithm design and actual software development are well worth this price. In the `ADPfusion` framework we currently have to accomodate approximately a doubling of the running time compared to expert-optimized implementations. Conceptually, the framework extends to multi-context-free grammars (MCFGs) and thus holds promise to drastically simplify the implementation of algorithms for RNA folding with pseudoknots and complex RNA-RNA interaction structures. Ongoing work in this area aims at formalizing MCFG-ADP theory [33] and the efficient implementation of the necessary parsers in `ADPfusion` [34].

Competing interests
The authors declare that they have no competing interests.

Author's contributions
All three authors conceived the study and wrote the manuscript. CHzS implemented the ADP framework, SJP designed and analyzed the gene duplication case study. This work is a revised and extended version of a manuscript presented at the *Brazilian Symposion on Bioinformatics* in Belo Horizonte, Oct 2014 [35].

Availability
The algorithms described in this work are part of the *generalized ADP* framework. Available here:
`http://www.bioinf.uni-leipzig.de/Software/gADP/`

Acknowledgements
This work was funded, in part, by the Austrian FWF, project "SFB F43 RNA regulation of the transcriptome", the Templeton Foundation, grant # 24332 "Origins and Evolution of Regulation in Biological Systems", and the DFG project "MI439/14-1".

Author details
[1]Bioinformatics Group, Department of Computer Science, Universität Leipzig, Härtelstraße 16–18, D-04107 Leipzig, Germany. [2]Department of Theoretical Chemistry, University of Vienna Währinger Straße 17, A-1090 Vienna, Austria. [3]Computational EvoDevo Group, Department of Computer Science, Universität Leipzig, Härtelstraße 16–18, D-04107 Leipzig, Germany. [4]Interdisciplinary Center for Bioinformatics, Universität Leipzig, Härtelstraße 16–18, D-04107 Leipzig, Germany. [5] Max Planck Institute for Mathematics in the Sciences, Inselstraße 22, D-04103 Leipzig, Germany. [6] Fraunhofer Institut for Cell Therapy and Immunology, Perlickstraße 1, D-04103 Leipzig, Germany. [7] Center for non-coding RNA in Technology and Health, Grønegårdsvej 3, DK-1870 Frederiksberg C, Denmark. [8] Santa Fe Institute, 1399 Hyde Park Rd., NM87501 Santa Fe, USA.

References

1. Giegerich, R., Meyer, C.: Algebraic dynamic programming. In: Kirchner, H., Ringeissen, C. (eds.) Algebraic Methodology And Software Technology. Lect. Notes Comp. Sci., vol. 2422, pp. 349–364. Springer, Berlin, Heidelberg (2002)
2. Höner zu Siederdissen, C.: Sneaking around concatMap: efficient combinators for dynamic programming. In: Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming (ICFP'12), pp. 215–226. ACM, New York (2012)
3. Sauthoff, G., Janssen, S., Giegerich, R.: Bellman's GAP – a declarative language for dynamic programming. In: Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming (PPDP'11), pp. 29–40. ACM, New York (2011)
4. Giegerich, R., Touzet, H.: Modeling dynamic programming problems over sequences and trees with inverse coupled rewrite systems. Algorithms, 62–144 (2014)
5. Höner zu Siederdissen, C., Hofacker, I.L.: Discriminatory power of RNA family models. Bioinformatics **26**, 453–459 (2010)
6. Voß, B., Giegerich, R., Rehmsmeier, M.: Complete probabilistic analysis of RNA shapes. BMC Biology **4**, 5 (2006)
7. Höner zu Siederdissen, C., Hofacker, I.L., Stadler, P.F.: Product grammars for alignment and folding. IEEE/ACM Trans. Comp. Biol. Bioinf. **99** (2014)
8. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE **77**, 257–286 (1989)
9. Baker, J.K.: Trainable grammars for speech recognition. J. Acoust. Soc. Am. **65**, 132 (1979)
10. Huang, F.W.D., Qin, J., Reidys, C.M., Stadler, P.F.: Partition function and base pairing probabilities for RNA-RNA interaction prediction. Bioinformatics **25**, 2646–2654 (2009)
11. Huang, F.W.D., Qin, J., Reidys, C.M., Stadler, P.F.: Target prediction and a statistical sampling algorithm for RNA-RNA interaction. Bioinformatics **26**, 175–181 (2010)
12. Reidys, C.M., Huang, F.W.D., Andersen, J.E., Penner, R.C., Stadler, P.F., Nebel, M.E.: Topology and prediction of RNA pseudoknots. Bioinformatics **27**, 1076–1085 (2011). Addendum in: Bioinformatics 28:300 (2012)
13. Janssen, S.: Kisses, ambivalent models and more: Contributions to the analysis of RNA secondary structure. PhD thesis, Univ. Bielefeld (2014). urn: nbn:de:hbz:361-26821318
14. Bellman, R.: Dynamic programming treatment of the travelling salesman problem. J. ACM **9**, 61–63 (1962)
15. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. SIAM **10**, 196–201 (1962)
16. Durbin, R., Eddy, S.R., Krogh, A., G., M.: Biological Sequence Analysis. Cambridge University Press, Cambridge (1998)
17. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**, 443–453 (1970)
18. Gotoh, O.: Alignment of three biological sequences with an efficient traceback procedure. J. theor. Biol. **121**, 327–337 (1986)
19. Lorenz, R., Bernhart, S.H., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P.F., Hofacker, I.L.: ViennaRNA Package 2.0. Alg. Mol. Biol. **6**, 26 (2011)
20. Wuchty, S., Fontana, W., Hofacker, I.L., Schuster, P.: Complete suboptimal folding of RNA and the stability of secondary structures. Biopolymers **49**(2), 145–165 (1999)
21. McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers **29**, 1105–1119 (1990)
22. Dowell, R.D., Eddy, S.R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics **5**, 71 (2004)
23. Höner zu Siederdissen, C., Berhart, S.H., Stadler, P.F., Hofacker, I.L.: A folding algorithm for extended RNA secondary structures. Bioinformatics **27**, 129–137 (2011)
24. Höner zu Siederdissen, C., Hammer, S., Abfalter, I., Hofacker, I.L., Flamm, C., Stadler, P.F.: Computational design of RNAs with complex energy landscapes. Biopolymers **99**, 1124–1136 (2013)
25. Elemento, O., Gascuel, O.: An efficient and accurate distance based algorithm to reconstruct tandem duplication trees. Bioinformatics **8 Suppl. 2**, 92–99 (2002)
26. Robinson, W.S.: A method for chronologically ordering archaeological deposits. Amer. Antiquity **16**, 293–301 (1951)
27. Prohaska, S.J., Höner zu Siederdissen, C., Stadler, P.F.: Expansion of gene clusters and the shortest Hamiltonian path problem, 2015
28. Garcia-Fernàndez, J.: Hox, parahox, protohox: facts and guesses. Heredity **94**, 145–152 (2005)
29. Cameron, R.A., Rowen, L., Nesbitt, R., Bloom, S., Rast, J.P., Berney, K., Arenas-Mena, C., Martinez, P., Lucas, S., Richardson, P.M., Davidson, E.H., Peterson, K.J., Hood, L.: Unusual gene order and organization of the sea urchin Hox cluster. J Exp Zoolog B Mol Dev Evol **306**, 45–58 (2006)
30. Höner zu Siederdissen, C., Hofacker, I.L., Stadler, P.F.: How to multiply Dynamic Programming algorithms. In: Brazilian Symposium on Bioinformatics (BSB 2013). Lect. Notes Bioinf., vol. 8213, pp. 82–93. Springer, Heidelberg (2013)
31. Billie, P.: A survey on tree edit distance and related problems. Theor. Comp. Sci. **337**, 217–239 (2005)
32. McBride, C.: Clowns to the left of me, jokers to the right (pearl): dissecting data structures **43**, 287–295 (2008)
33. Riechert, M.: Algebraic Dynamic Programming for Multiple Context-Free Languages
34. Riechert, M., Höner zu Siederdissen, C., Stadler, P.F., Waldmann, J.: Algebraic dynamic programming for multiple context-free languages. in preparation (2015)
35. Höner zu Siederdissen, C., Prohaska, S.J., Stadler, P.F.: Dynamic programming for set data types. In: Campos, S. (ed.) Advances in Bioinformatics and Computational Biology: BSB 2014. Lect. Notes Comp. Sci., vol. 8826, pp. 57–64 (2014)