Algebraic Dynamic Programming 2.0

Robert Giegerich
Faculty of Technology and Center for Biotechnology,
Bielefeld University, 33594 Bielefeld, Germany
robert@techfak.uni-bielefeld.de

Hélène Touzet
LIFL (UMR CNRS 8022, University of Lille) and INRIA,
59655 Villeneuve d'Ascq Cedex, France
helene.touzet@lifl.fr

April 21, 2013

Abstract

Haskell has played a major role – and it still does – in the implementation of the Algebraic Dynamic Programming framework as it emerged about a decade ago. Here, we present a—yet unpublished—generalization of the ADP framework which also accomodates problems on trees. The new framework is not an "add-on" extension, but suggests a reformulation of "classical" ADP. ADP 2.0 is not only more general, but (arguably) more elegant – and more difficult to implement. In fact, no general implementation technique is known at present.
An ensemble of about 30 problems – most classical, some new – from biosequence and structure analysis has been described in the new framework. In the workshop contribution, we will present some of them, according to audience's choice. We will point out the challenges in implementing the new framework, hoping to solicit cooperation from the Haskell community.

1 Outline of ideas

We introduce *Inverse Coupled Rewrite Systems* (ICOREs) as a high-level and unified view on a diverse set of combinatorial optimization problems on sequences and trees. ICOREs are based on the following ideas: Candidate solutions of optimization problems have a natural representation in some term algebra, whose function symbols reflect the case analysis required by the problem at hand. A tree grammar may be used to further refine the search space by describing a language of well-formed candidates. Optimization objectives are specified as interpretations of these terms in a suitable scoring algebra, together with an objective function. The relation between input terms and their candidate solutions is established by a term rewrite system. This rewrite system works in the *wrong* direction, mapping solutions back to the input(s) of the problem they solve. For problems with several inputs, the rewriting to these inputs is performed by different rules, but in a *coupled* manner.

These constituents provide a mathematically precise and complete problem specification. Implementing an algorithm to solve the problem is non-trivial. To actually solve a problem for given inputs, the coupled rewrite relation must be inverted. Candidates must be constructed, evaluated, and have the objective applied to them. To do so efficiently, all the dynamic programming machinery must eventually be brought in – but no allusion to dynamic programming appears in the problem specification.

The long-term goal of our work is twofold. We want to

- *describe optimization problems* on a declarative level of abstraction, where fundamental ideas are not obscured by implementation detail, and relationships between similar problems are transparent and can be exploited;

- *implement algorithmic solutions* to these problems in a systematic or even automated fashion, thus liberating algorithm designers from error-prone coding and tedious debugging work, enabling re-use of tried-and-tested components, and overall, enhancing programmer productivity and program reliability.

In the present contribution, we focus on the former of these goals. We will develop a substantial number of problem specifications expressed in our new framework, all drawn from the application domains of computaional biology. See Table 1. However, as much as sequences, trees and dynamic programming are ubiquitous in computer science, this covers only a small sub-domain of the potential scope of our approach. For example, the tree comparison techniques we use with RNA secondary structure have also been employed to compare objects assembled by robots, or to extract and compare documents from the web.

The expressive power of the ICORE concept is not formally circumscribed. Note, for example, that rewrite rules are allowed to copy variables, as in $f(X) \to X{\sim}X$, which covers some aspects of copy languages. By presenting a diverse set of real-world examples from bioinformatics, we are hoping to convince the reader that the second goal, developing implementation techniques for ICOREs, is a new and rewarding research challenge.

2 Examples

Definitions We skip all formalism, and rely on the reader's background with respect to signatures, algebras, tree grammars, and term rewrite rules. An ICORE problem is specified by

- a dimension k, indicating the number of inputs,
- k "satellite" signatures for the k inputs,
- a "core" signature ζ for the candidate solutions that make up the universal search space,
- a tree grammar $\mathcal{G}$ designating the legal candidates as a subset of the term algebra T_ζ,
- a set of k "coupled" rewrite systems; their rules share the lefthand sides and specify how a core term rewrites to each of its inputs,
- a ζ-algebra A (or several) specifying the evaluation (scoring) of core terms and an objective function to choose optimal answers.

So in general, inputs are trees (terms) over some signature. Of course, sequences are an especially simple type of tree, made up from an alphabet of characters, the empty word ε, and a concatenation operator, which we denote $\sim$ whenever we write it explicitly. Generally, inputs can be a mixture of sequences and trees over different alphabets and signatures.

An instance of an ICORE problem is specified by its k inputs. Its solution is the multiset of candidates $t \in T_\zeta$ (resp. their scores $A(t)$) which

- are in $L(\mathcal{G})$,
- rewrite to the inputs in all k dimensions,
- are chosen from all the above by the objective function in A.

ICORE / Grammar	Dim.	Problem addressed	in Section	Pag
EditDistance	2	simple edit distance/alignment	??	??
Affine	2	edit distance, affine gaps	??	??
AffiOsci	2	oscillating affine gaps	??	??
AffiTrace	2	Sequence traces, affine gaps	??	??
LocalSearch	2	Generic Local Alignment	??	??
MotifSearch	2	short in long alignment	??	??
SemiGlobAlignment	2	semi-global alignment	??	??
LocalAlignment	2	local alignment	??	??
MatchSeq_S	1	Hardwired sequence matching	??	??
MatchAffi_S	1	same with affine gap model	??	??
MatchSeq_S with position-specific scores	1	Profile HMM	??	??
RNAfold	1	RNA folding	??	??
StructAli	2	struct. Alignment prototype	??	??
SimultaneousFolding	2	generalized fold and align	??	??
ExactConsensusStructure	2	exact consensus structure for two RNA sequences	??	??
Sankoff	2	simultaneous fold and align	??	??
S2SGeneric	2	covariance model, generic	??	??
S2SExact	2	match RNA sequence to target structure	??	??
Match_S2S_r	2	exact local motif matcher	??	??
Match_S2S_r'	1	motif matcher, hard coded	??	??
SCFG	1	stochastic context free grammar	??	??
CovarianceModel_r	1	covariance model, hard coded	??	??
TreeAlign	2	classical tree alignment	??	??
TreeAliGeneric	2	tree alignment protoype + variants	??	??
OsciSubforest	2	tree ali. with oscillating gaps	??	??
TreeEdit	2	classical tree edit	??	??
RNATreeAli	2	generalized tree alignment	??	??

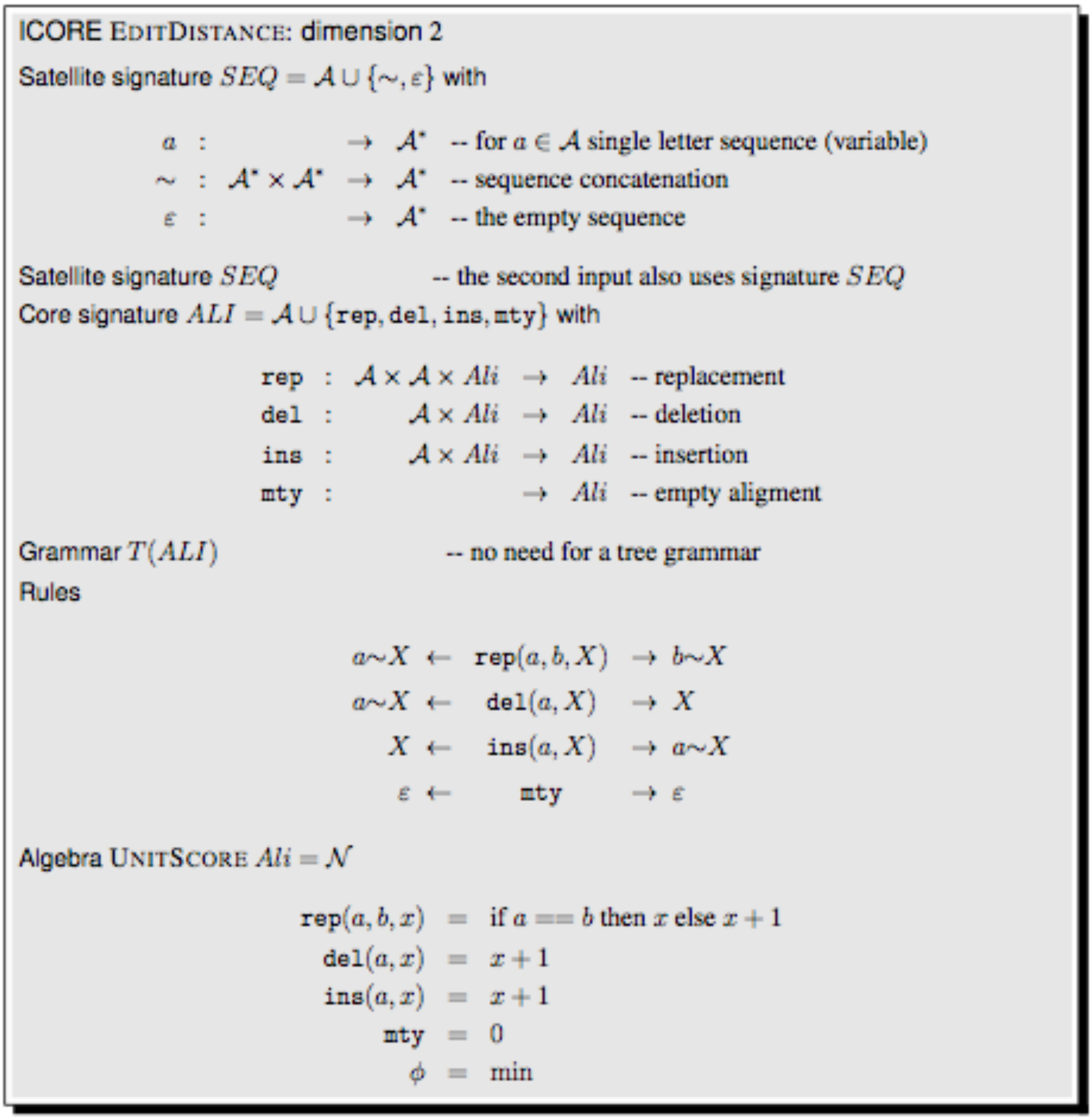
Table 1: A summary of the ICOREs available for the workshop presentation. The unresolved links refer to a manuscript currently under submission.

Comparing this setup to “first generation” ADP, note that the tree grammar no longer relates solutions to inputs. As a consequence, the tree grammar is often not required, and the full term algebra T_ζ is used in its place. Connection between outputs and inputs is established solely by the rewrite relation – in the “wrong” direction, which makes ICORE specification easy and their implementation hard.

Often, the core as well as the satellite signatures can be infered from the rewrite rules alone. In such a case, ICOREs lend themselves to a very terse presentation, consisting only of rewrite system and algebra(s). This is convenient for initial ICORE design, whereas in a practical programming envorinment, a certain degree of redundancy is desirable.

Next, we show two ICORE examples, relying on the reader’s intuition to guess some detail not given here explicitly.

String edit distance Here is the ICORE formulation for the classical edit distance problem. $\mathcal{A}$ denotes the underlying alphabet.



ICORE EDITDISTANCE: dimension 2

Satellite signature $SEQ = \mathcal{A} \cup \{\sim, \varepsilon\}$ with

$$\begin{array}{lclcll}
a & : & & \to & \mathcal{A}^* & \text{-- for } a \in \mathcal{A} \text{ single letter sequence (variable)} \\
\sim & : & \mathcal{A}^* \times \mathcal{A}^* & \to & \mathcal{A}^* & \text{-- sequence concatenation} \\
\varepsilon & : & & \to & \mathcal{A}^* & \text{-- the empty sequence}
\end{array}$$

Satellite signature SEQ -- the second input also uses signature SEQ

Core signature $ALI = \mathcal{A} \cup \{\texttt{rep}, \texttt{del}, \texttt{ins}, \texttt{mty}\}$ with

$$\begin{array}{lcrcll}
\texttt{rep} & : & \mathcal{A} \times \mathcal{A} \times Ali & \to & Ali & \text{-- replacement} \\
\texttt{del} & : & \mathcal{A} \times Ali & \to & Ali & \text{-- deletion} \\
\texttt{ins} & : & \mathcal{A} \times Ali & \to & Ali & \text{-- insertion} \\
\texttt{mty} & : & & \to & Ali & \text{-- empty aligment}
\end{array}$$

Grammar $T(ALI)$ -- no need for a tree grammar

Rules

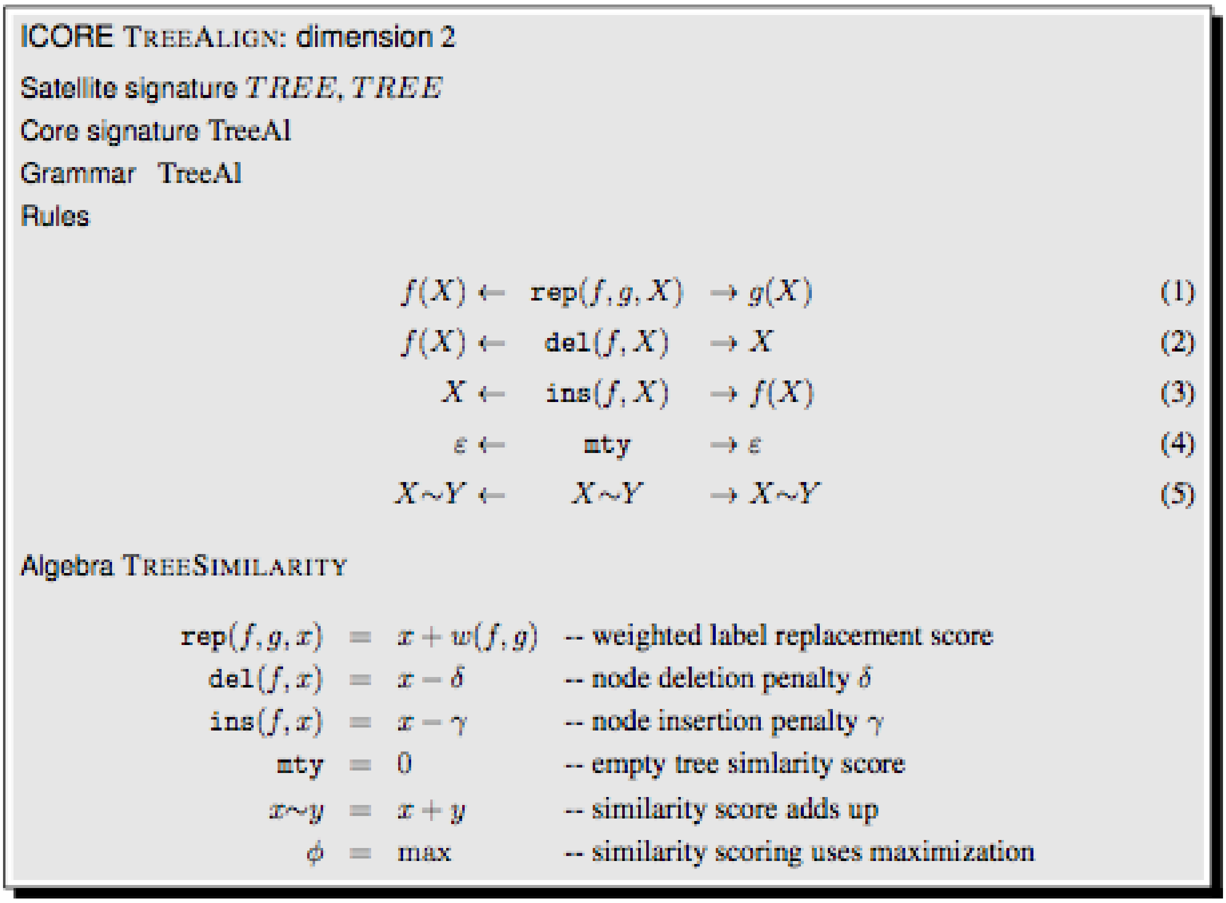
$$\begin{array}{rclcl}
a{\sim}X & \leftarrow & \texttt{rep}(a, b, X) & \to & b{\sim}X \\
a{\sim}X & \leftarrow & \texttt{del}(a, X) & \to & X \\
X & \leftarrow & \texttt{ins}(a, X) & \to & a{\sim}X \\
\varepsilon & \leftarrow & \texttt{mty} & \to & \varepsilon
\end{array}$$

Algebra UNITSCORE $Ali = \mathcal{N}$

$$\begin{array}{rcl}
\texttt{rep}(a, b, x) & = & \text{if } a == b \text{ then } x \text{ else } x + 1 \\
\texttt{del}(a, x) & = & x + 1 \\
\texttt{ins}(a, x) & = & x + 1 \\
\texttt{mty} & = & 0 \\
\phi & = & \min
\end{array}$$

The basic string edit ICORE smoothly generalizes to affine gap scoring, local alignment, small-in-large alignment, alignments of genomic sequence against amino acid sequences, sequence motif search, and profile HMMs.

Tree alignment Here is the ICORE formulation for the classical tree alignment problem. Satellite signature $TREE$ allows for rooted, ordered, node-labeled trees. Node labels are treated as unary operators, where forests of subtrees are concatenated by $\sim$. Such an operator also exists in the core signature, where forests of sub-alignments are also formed with $\sim$. Rule (5) rewrites between the two instances of $\sim$.



ICORE TreeAlign: dimension 2

Satellite signature $TREE, TREE$

Core signature TreeAl

Grammar TreeAl

Rules

$$
\begin{array}{rcll}
f(X) \leftarrow & \texttt{rep}(f,g,X) & \rightarrow g(X) & (1)\\
f(X) \leftarrow & \texttt{del}(f,X) & \rightarrow X & (2)\\
X \leftarrow & \texttt{ins}(f,X) & \rightarrow f(X) & (3)\\
\varepsilon \leftarrow & \texttt{mty} & \rightarrow \varepsilon & (4)\\
X{\sim}Y \leftarrow & X{\sim}Y & \rightarrow X{\sim}Y & (5)
\end{array}
$$

Algebra TreeSimilarity

$$
\begin{array}{rcll}
\texttt{rep}(f,g,x) & = & x + w(f,g) & \text{-- weighted label replacement score}\\
\texttt{del}(f,x) & = & x - \delta & \text{-- node deletion penalty } \delta\\
\texttt{ins}(f,x) & = & x - \gamma & \text{-- node insertion penalty } \gamma\\
\texttt{mty} & = & 0 & \text{-- empty tree simlarity score}\\
x{\sim}y & = & x + y & \text{-- similarity score adds up}\\
\phi & = & \max & \text{-- similarity scoring uses maximization}
\end{array}
$$

The tree aligment ICORE generalizes to several variants of composite gaps and affine gap scoring. Classical tree edit distance, however, starts from a different ICORE and uses associative rewriting. A more expressive tree alignment model, semantically richer than the classical, node-by-node tree edit model, starts from a set of bidirectional rewrite rules, from which an ICORE implementing this model can be constructed in a systematic fashion.