A Propagator for Maximum Weight String Alignment with Arbitrary Pairwise Dependencies

Alessandro Dal Palù[1], Mathias Möhl[2], and Sebastian Will[2,3]

[1] Dipartimento di Matematica, Università degli Studi di Parma, Parma, Italy, alessandro.dalpalu@unipr.it
[2] Bioinformatics, Institute of Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany, {mmohl,will}@informatik.uni-freiburg.de
[3] Computation and Biology Lab, CSAIL, MIT, Cambridge MA, USA, swill@csail.mit.edu

Abstract. The optimization of weighted string alignments is a well studied problem recurring in a number of application domains and can be solved efficiently. The problem becomes MAX-SNP-hard as soon as arbitrary pairwise dependencies among the alignment edges are introduced. We present a global propagator for this problem which is based on efficiently solving a relaxation of it. In the context of bioinformatics, the problem is known as alignment of arc-annotated sequences, which is e.g. used for comparing RNA molecules. For a restricted version of this alignment problem, we show that a constraint program based on our propagator is on par with state of the art methods. For the general problem with unrestricted dependencies, our tool constitutes the first available method with promising applications in this field.

The maximum weight string alignment problem for strings S_a and S_b asks for a partial matching of positions in S_a and S_b that preserves the string order and has maximum weight. This problem is efficiently solved by dynamic programming (DP) [10]. An extended variant of the problem introduces pairwise dependencies among positions in each string S_a and S_b. In this problem variant, one optimizes the sum of weights, where a weight is associated with each pair of matched positions and with each pair of matched dependencies. In general, this problem is MAX-SNP-hard [2].

In bioinformatics, the problem has been studied as alignment of arc-annotated sequences, where each arc represents a dependency. There, the problem has applications in aligning RNA or protein molecules that can be abstracted as sequences of monomers and structural dependencies among those that represent a proximity of the respective positions in the molecules structure.

Due to the hardness of the problem, restricted versions with limited dependencies, in particular nested and crossing ones, have been considered. For nested and also certain restricted crossing dependencies the alignment problem can be solved efficiently with DP algorithms [6, 9]. Heuristic approaches based on Integer Linear Programming (ILP) are available for crossing dependencies specialized to

RNA [1] and for unlimited dependencies specialized to proteins [3]. By using DP-based propagation, our approach is similar to Trick [11]. Furthermore, there is prior work applying this idea in an optimization setting [5].

Contribution In this work, we consider a constraint programming approach to the arc-annotated sequence alignment problem with unlimited dependencies. Our main contribution is a general propagator for the maximum weight string alignment with arbitrary pairwise dependencies. The propagator is based on a relaxation that resolves the dependencies by bounding their weight contribution. It propagates on the upper bound of the total weight and prunes the valid alignments accordingly. Furthermore, we discuss decomposition into independent subproblems for optimization approaches using our propagator.

We apply the technique to RNA sequence-structure alignment and show that results are comparable to the state-of-the-art ILP approach Lara [1]. Finally, we discuss a new method to compare riboswitches, which was not applicable before, because prior RNA alignment approaches score at most crossing dependencies.

1 Preliminaries

An *arc-annotated sequence* is a pair (S, P), where the *sequence* S is a string over some alphabet Σ and P is a set of *arcs* (i, j) with $1 \leq i < j \leq |S|$. We denote the i-th symbol of S by $S[i]$ and $S[i..j]$ is the subsequence $S_i S_{i+1} \dots S_j$.

We distinguish crossing and unlimited sets of arcs. A set P, where each sequence position is involved in at most one arc, i.e. $\forall (i,j) \neq (i',j') \in P : i \neq i' \wedge j \neq j' \wedge i \neq j \wedge i' \neq j'$, is called *crossing*. Otherwise it is called *unlimited*.

An *alignment* A of two arc-annotated sequences (S_a, P_a) and (S_b, P_b) is a ordered partial matching between the positions of S_a and S_b. More precisely, $A \subseteq \{1, \dots, |S_a|\} \times \{1, \dots, |S_b|\}$ has to satisfy for all $(i,i'), (j,j') \in A$ that 1.) $i > j$ implies $i' > j'$ and 2.) $i = j$ if and only if $i' = j'$. We define the (i,i')-*prefix of* A as $A \cap \{ (j,j') \mid j \leq i, j' \leq i' \}$ and the (i,i')-*suffix of* A as $A \cap \{ (j,j') \mid j > i, j' > i' \}$.

Fix two arc-annotated sequences (S_a, P_a) and (S_b, P_b) with unlimited structures P_a and P_b. Define the *weight of alignment* A *of* (S_a, P_a) *and* (S_b, P_b) as

$$\text{weight}(A) := \sum_{(i,i') \in A} \sigma(i,i') + \sum_{\substack{(i,j) \in P_a, (i',j') \in P_b, \\ (i,i') \in A, (j,j') \in A}} \tau(i,j,i',j') \quad + \quad \gamma(|S_a| + |S_b| - 2|A|),$$

where $\sigma(i,i')$ is the similarity of sequence positions $S_a[i]$ and $S_b[i']$, $\tau(i,j,i',j')$ is the similarity of arcs $(i,j) \in P_a$ and $(i',j') \in P_b$ and γ is the gap cost associated with each sequence position that is not matched.

The *alignment problem* is to determine

$$\underset{A \text{ alignment of } (S_a, P_a) \text{ and } (S_b, P_b)}{\text{argmax}} \text{weight}(A).$$

Note that on crossing arc annotation, the ILP approach Lara [1] solves essentially the same problem. On unlimited input, Lara scores only a crossing subset of the matched arcs whereas our approach scores all matches of arcs.

2 Constraint Model

We model an alignment of arc-annotated sequences (S_a, P_a) and (S_b, P_b) by variables $\texttt{MD}_i$ and $\texttt{M}_i$ for $1 \leq i \leq |S_a|$ with initial domains $D(\texttt{MD}_i) = \{1, \ldots, |S_b|\}$ and $D(\texttt{M}_i) = \{0, 1\}$. We write $\boldsymbol{MD}$ and $\boldsymbol{M}$ to denote the vectors of respective variables $\texttt{MD}_i$ and $\texttt{M}_i$.

A valuation V of these variables corresponds to at most one alignment A_V of (S_a, P_a) and (S_b, P_b) as defined by

$$V(\texttt{MD}_i) = j \wedge V(\texttt{M}_i) = 1 \text{ iff } (i,j) \in A$$
$$V(\texttt{MD}_i) = j \wedge V(\texttt{M}_i) = 0 \text{ iff } \nexists j \text{ with } (i,j) \in A$$
$$\wedge \forall (i',j') \in A : i' < i \rightarrow j' \leq j \wedge i' > i \rightarrow j' > j.$$

In this way, M_i tells whether i is matched or deleted and the value j of MD_i tells that i is matched to j or deleted after j. One can show that for each alignment A of (S_a, P_a) and (S_b, P_b) there is a corresponding valuation V with $A = A_V$.

For example, the following alignment $A = \{(1,1),(2,4),(4,5)\}$ of $S_a = ACUG$ and $S_b = ACACG$, which is often written as $\begin{matrix}\texttt{A--CUG}\\ \texttt{ACAC-G}\end{matrix}$, corresponds to the valuation $\boldsymbol{MD} = (1,4,4,5)$ and $\boldsymbol{M} = (1,1,0,1)$.

We introduce a constraint $\mathsf{StringAlignment}(\boldsymbol{MD}, \boldsymbol{M})$ that is satisfied by any valuation with a corresponding alignment. For modeling the weight of the alignment, we introduce a variable `Weight` and a constraint $\mathsf{StringAlignmentWeight}(\boldsymbol{MD}, \boldsymbol{M}, \texttt{Weight})$. This constraint relates a valuation of `MD` and `M` to the weight of its corresponding alignment. Note that, formally, (S_a, P_a) and (S_b, P_b) are parameters of the constraints but we omit them to simplify notation.

Both constraints are propagated by our propagator described in the next section. For finding optimal alignments we perform a branch-and-bound search enumerating `MD` and `M` according to a specific search strategy described after introducing the propagator itself.

3 The Alignment Propagator

Our propagator computes hyper-arc consistency for $\mathsf{StringAlignment}(\boldsymbol{MD}, \boldsymbol{M})$ and propagates $\mathsf{StringAlignmentWeight}(\boldsymbol{MD}, \boldsymbol{M}, \texttt{Weight})$.

It prunes $\boldsymbol{MD}$ and $\boldsymbol{M}$ due to the weight by computing upper bounds of weights for single variable assignments and furthermore computes lower and upper bounds for `Weight` based on $\boldsymbol{MD}$ and $\boldsymbol{M}$. Computing such bounds efficiently is essential for branch-and-bound optimization.

Define the class $\mathcal{A}(D)$ as union of A_V over all valuations V that satisfy D. The computation of bounds is based on a relaxation of the alignment problem. In this relaxation the two ends of each arc match are decoupled. Thus in the *relaxed optimization problem for* D, we maximize a relaxed weight

$$\text{weight}^{n,m}_{\text{relaxed}}(A) := \sum_{(i,i') \in A} [\sigma(i,i') + \text{ub}_D(i,i')] \quad + \quad \gamma(n + m - |A|),$$

over all alignments in $\mathcal{A}(D)$, where $n = |S_a|$ and $m = |S_b|$ and

$$\mathrm{ub}_D(i,i') := \frac{1}{2} \max_{A \in \mathcal{A}(D)} [\sum_{\substack{(i,j)\in P_a,(i',j')\in P_b,\\ (i,i')\in A,(j,j')\in A}} \tau(i,j,i',j') \; + \sum_{\substack{(j,i)\in P_a,(j',i')\in P_b,\\ (i,i')\in A,(j,j')\in A}} \tau(j,i,j',i')].$$

Here, ub_D works as an upper bound for the weight contributions by arc matches involving (i,i') and consequently $\mathrm{weight}^{|S_a|,|S_b|}_{\mathrm{relaxed}}(A) \geq \mathrm{weight}(A)$ for $A \in \mathcal{A}(D)$. Thus, solving the relaxed problem yields an upper bound of `Weight`.

For a moment, postpone how to efficiently compute $\mathrm{ub}_D(i,i')$. Then, because the relaxed weight has the form of a sequence similarity score, one can apply the Smith-Waterman algorithm [10] to maximize the relaxed weight in $O(n^2)$ by dynamic programming, where $n = \max(|S_a|,|S_b|)$. The optimization problem is easily constrained due to domain D, because domains directly restrict the valid cases in the dynamic programming recursion.

Tracing back through the dynamic programming matrix yields an alignment A^l. If A_l also satisfies all other constraints of the constraint problem, then $\mathrm{weight}(A^l)$ is a lower bound of `Weight`. For the later studied RNA alignment problem, this bound can always be propagated, since there are no other constraints. Furthermore, we compute upper bounds for each single variable valuation. This requires to complement the above "forward algorithm" that computes the matrix entries

$$\mathrm{Prefix}\,(i,i') := \max_{(i,i')\text{-prefix } A^p_{ii'} \text{ of } A\in\mathcal{A}(D)} \mathrm{weight}^{i,i'}_{\mathrm{relaxed}}(A^p_{ii'})$$

by a symmetric "backward algorithm" that computes the matrix entries

$$\mathrm{Suffix}\,(i,i') := \max_{(i,i')\text{-suffix } A^s_{ii'} \text{ of } A\in\mathcal{A}(D)} \mathrm{weight}^{|S_a|-i,|S_b|-i'}_{\mathrm{relaxed}}(A^s_{ii'}).$$

Now the variables $\boldsymbol{MD}$ can be pruned efficiently, because $\mathrm{Prefix}\,(i,i')+\mathrm{Suffix}\,(i,i')$ is an upper bound of `Weight` for the assignment $\mathtt{MD}_i = j$. Hence, j can be removed from the domain of $\mathtt{MD}_i$, if $\mathrm{Prefix}\,(i,i') + \mathrm{Suffix}\,(i,i')$ is larger than the upper bound of `Weight`. Similarly, we prune $\boldsymbol{M}$ using the two matrices.

It remains to describe the efficient computation of $\mathrm{ub}_D(i,i')$. It suffices to describe the maximization of $\sum_{\substack{(i,j)\in P_a,(i',j')\in P_b,\\ (i,i')\in A,(j,j')\in A}} \tau(i,j,i',j')$ over alignments in $\mathcal{A}(D)$. A single match (j,j') can occur in an alignment in $\mathcal{A}(D)$ if $j' \in D(\mathtt{MD}_j)$ and $1 \in D(\mathtt{M}_j)$. However, we look for the best set of simultaneously valid matches (j,j'). The structure of this subproblem is analogous to sequence alignment. For solving it efficiently, we define sorted lists $j_1,\ldots,j_l$ and $j'_1,\ldots,j'_{l'}$ such that $(j_h,i) \in P_a$ for all $1 \leq h \leq l$ and $(j'_{h'},i') \in P_b$ for all $1 \leq h' \leq l'$. We apply dynamic programming for evaluating

$$\mathrm{UL}(0,0) = 0 \quad \mathrm{UL}(h,0) = 0 \quad \mathrm{UL}(0,h') = 0$$

$$\mathrm{UL}(h,h') = \max \begin{cases} UL(h-1,h') & \text{unless } \mathtt{MD}_{j_h} = j'_{h'} \text{ and } \mathtt{M}_{j_h} = 1 \\ UL(h,h'-1) & \text{unless } \mathtt{MD}_{j_h} = j'_{h'} \text{ and } \mathtt{M}_{j_h} = 1 \\ UL(h-1,h'-1) + \tau(j_h,j'_{h'},i,i') & j'_{h'} \in D(\mathtt{MD}_{j_h}) \text{ and } 1 \in D(\mathtt{M}_{j_h}) \end{cases}$$

for $1 \leq h \leq l$, $1 \leq h' \leq l'$. Then, we perform the same construction for the respective r and r' many arcs to the right of i and i' and evaluate the corresponding recursion equation $UR(h, h')$ for $1 \leq h \leq r$, $1 \leq h' \leq r'$. Then, $\mathrm{ub}_D(i, i') = \mathrm{UL}(l, l') + \mathrm{UL}(r, r')$.

Therefore, $\mathrm{ub}_D(i, i')$ is computed in $O(ll' + rr')$ time. For crossing arcs $l + r = l' + r' = 1$ and for many other applications $l + r$ and $l' + r'$ can be constantly bounded [12] such that the propagator runs in $O(n^2)$ time and space.

Affine Gap Cost In bioinformatics, penalizing unmatched positions using an affine weighting function yields more realistic results. Our method is straightforwardly extended to such scoring by using a Gotoh-like forward and backward algorithm [4] in the propagator without increasing its complexity. It appears that this modification comes more natural in our approach than the corresponding extension in ILP, because it does not require any change of the model.

Propagator-guided Search Strategy To maximize the use of the propagator in a branch-and-bound setting, we suggest a search strategy that aims at disproving overestimated bounds fast and finding valid good alignments quickly. To achieve this, information computed by the propagator in each propagation step can be used to guide the search. In particular, this allows to select a variable that yields a high undecided contribution to the upper bound of the total weight. Furthermore, the computed backtrace provides a good candidate for a solution, which can be favored by the search strategy.

In our application to RNA, we select a variable with highest undecided contribution to the upper bound and domain size as tie breaking. Its domain is split such that the 20% highest relaxed weights are chosen first.

4 Problem Decomposition

Certain constraint optimization problems can be solved faster by detecting independent subproblems during search and optimizing these subproblems independently. In our setting, independent means that the two parts of the alignment do not depend on each other due to the string order, arcs connecting the two parts, or due to additional constraints other than StringAlignment and StringAlignmentWeight. In the following, we denote anything that makes two subproblems dependent a dependency, not only dependencies introduced by arcs.

In general, problem decomposition introduces overhead for detecting dependencies and even interferes with Branch-and-Bound when subproblems cannot be bounded well (confer AND/OR search [7], which however doesn't discuss decomposition in the presence of global propagators). However, the string alignment problem suggests a special form of decomposition along the string order, where our propagator provides upper bounds for the partial problems. The dependency due to the string order between the two subproblems for variables $\mathtt{MD}_1, \ldots, \mathtt{MD}_{i-1}$, $\mathtt{M}_1, \ldots, \mathtt{M}_{i-1}$ and $\mathtt{MD}_{i+1}, \ldots, \mathtt{MD}_{|S_a|}$, $\mathtt{M}_{i+1}, \ldots, \mathtt{M}_{|S_a|}$ is resolved as soon as a matching edge (i, j) is assigned (i.e. $\mathtt{MD}_i = j$ and $\mathtt{M}_i = 1$). The problem

can be decomposed, if in addition the dependencies due to arcs are resolved and the corresponding variables do not have dependencies by other constraints. Notably, for resolving the dependencies due to an arc (i, j) it suffices that one of its ends is matched, i.e. $\forall A \in D(\mathcal{A}) : (i, i') \in A$ (or $\forall A \in D(\mathcal{A}) : (j, j') \in A$). Then one can move the weight of each arc match, $\tau(i, j, i', j')$, to the match weight of the other end $\sigma(j, j')$ (or $\sigma(i, i')$ respectively) and discard the dependency.

In the RNA alignment problem, all dependencies can be checked within the propagator.[1] To avoid overhead, we apply the decomposition only if an independent subproblem for $\mathtt{MD}_i, \ldots, \mathtt{MD}_j$, $\mathtt{M}_i, \ldots, \mathtt{M}_j$ can be solved to optimality by the propagator alone. This is the case, if all arc dependencies in the subproblem are resolved. Then, the problem reduces to maximum weighted string alignment without dependencies and the traceback alignment represents an optimal solution. Assigning the traceback of the subproblem to $\mathtt{MD}_i, \ldots, \mathtt{MD}_j$, $\mathtt{M}_i, \ldots, \mathtt{M}_j$ is a form of symmetry breaking, because it discards alignments with less or equal weight that are systematically generated from other alignments in $\mathcal{A}(D)$.

5 Results

The propagator and its application to RNA sequence-structure alignment, called Carna, is implemented in C++ using the constraint programming system Gecode. For handling input and output as well as for special data structures we reused code of LocARNA [12].

We run tests for two application scenarios. All experiments were performed under 32-bit Linux on a T400s notebook with Intel P9600 CPU. First, we explore Carna's behavior on crossing input structure using instances from all 16 Rfam families with crossing structure. Table 1 compares our results to Lara [1]. The table omits all 8 instances where both approaches run in less than 0.1 seconds. In all but one of the omitted cases, Carna solves the problem without backtracking. In terms of performance, with the single exception of tmRNA, both programs are on a par for the simpler class of crossing structures.

In our second scenario, we evaluate the behavior on the general class of unlimited structures. Therefore, we apply the approach to the alignment of riboswitches, which are RNA molecules with more than one evolutionary conserved structure. We annotate the RNA sequences by the set of all base pairs with sufficiently high probability in the RNA's structure ensemble [8]. This set approximates the overlay of the different riboswitch structures. In consequence, the alignment is optimized with respect to all these structures simultaneously. Since the set of base pairs is unlimited this application has not been possible for existing approaches, which at most score crossing structure. The Rfam database contains 10 RNA riboswitch families that are confirmed by literature. To benchmark the approach, we align 100 random instances from each of those families. Since in large scale studies an instance is rather skipped than spending much time on it, we set a time limit of 1 minute for each instance. The results are

[1] This is important for our implementation, because checking for independent subproblems in the presence of arbitrary propagators is expensive in Gecode.

Family	Lengths		Run-time (s)		Carna Search Tree		
	S_a	S_b	Carna	Lara	Depth	Fails	Size
Entero_OriR	126	130	0.03	0.18	38	13	50
Intron_gpI	443	436	0.1	0.2	0	0	1
IRES_Cripavirus	202	199	0.2	0.04	157	127	296
RNaseP_arch	303	367	0.46	1.4	63	8	64
RNaseP_bact_b	408	401	3.0	2.3	370	677	1463
RNaseP_nuc	317	346	0.07	2.9	14	4	16
Telomerase-vert	448	451	0.47	2.3	146	32	161
tmRNA	384	367	63	3.7	433	14347	28785

Table 1. Results for the eight harder instances of the benchmark set with crossing structures. We omit details for 8 instances where both programs run in less than 0.1 seconds. All times are given as user times.

given in Table 2. For this new problem, not all instances could be solved within our strict time limit. However, the results show that the approach handles all Riboswitch families sufficiently well for bioinformatics applications with only some limitations for the very longest sequences.

Family	Length	Base pairs	Time (s)	Memory (MB)	Limit
SAH_riboswitch	79	81	0.13	3.3	2%
SAM_alpha	79	96	0.03	1.9	0%
Purine	101	74	0.07	2.3	0%
Glycine	101	83	0.44	5.0	3%
SAM	106	74	0.06	6.0	0%
TPP	107	96	0.43	8.7	13%
SAM-IV	116	128	0.05	3.7	2%
MOCO_RNA_motif	141	111	0.24	9.4	10%
Lysine	181	210	60	14.5	60%
Cobalamin	204	237	60	18.7	71%

Table 2. Rfam Riboswitches. For 100 instances from each Rfam family annotated as riboswitch and confirmed by literature: Medians of average sequence length, average number of base pairs, and run-time (user time), maximal memory requirement and percentage of instances not solved to optimality within a given time limit of 1 min.

6 Discussion

We presented a propagator for the problem of weighted string alignment with arbitrary pairwise dependencies, which is also known as the alignment problem for arc-annotated sequences with unlimited structure. Whereas the problem itself is MAX-SNP-hard, our propagator allows for an effective constraint programming approach by efficiently solving relaxations of the problem. Futhermore, we

proposed a search strategy that improves the benefit due to the propagator. Finally, we showed that the weighted string alignment problem can be decomposed into independent subproblems during search. This allows for a AND/OR-type optimization in the context of our global propagator.

To evaluate the applicability of our method in practice, we apply it to the alignment of RNA structures. While all previous approaches in this area are limited to score at most crossing subsets of the input structures, our approach is able to align unlimited structures. This is useful to align Riboswitches and other molecules with more than one conserved structure, because it allows considering all their potential structural conformations simultaneously.

Acknowledgments This work is partially supported by DFG grants WI 3628/1-1 and BA 2168/3-1 and PRIN-2008 (*Innovative multi-disciplinary approaches for constraint and preference reasoning*).

References

1. Markus Bauer, Gunnar W. Klau, and Knut Reinert. Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. *BMC Bioinformatics*, 8:271, 2007.
2. Guillaume Blin, Guillaume Fertin, Irena Rusu, and Christine Sinoquet. Extending the hardness of RNA secondary structure comparison. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, volume 4614 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2007.
3. Alberto Caprara and Giuseppe Lancia. Structural alignment of large-size proteins via lagrangian relaxation. In *Proceedings of the Sixth Annual International Conference on Computational Biology (RECOMB2002)*, pages 100–108. ACM Press, 2002.
4. O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.
5. Willem-Jan Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4-5):347–373, 2006.
6. Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–88, 2002.
7. Radu Marinescu and Rina Dechter. And/or branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17):1457–1491, 2009.
8. J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–19, 1990.
9. Mathias Möhl, Sebastian Will, and Rolf Backofen. Lifting prediction to alignment of RNA pseudoknots. *Journal of Computational Biology*, 2010. Accepted.
10. T.F. Smith and M.S. Waterman. Comparison of biosequences. *Adv. appl. Math.*, 2:482–489, 1981.
11. Michael A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals OR*, 118(1-4):73–84, 2003.
12. Sebastian Will, Kristin Reiche, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. *PLOS Computational Biology*, 3(4):e65, 2007.