Sparsification Enables Predicting Kissing Hairpin Pseudoknot Structures of Long RNAs in Practice

Hosna Jabbari[1], Ian Wark[1], Carlo Montemagno[1], and Sebastian Will[4]

1 **Ingenuity Lab, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada**
2 **Ingenuity Lab, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada**
3 **Ingenuity Lab, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada**
4 **Theoretical Biochemistry Group (TBI), Institute for Theoretical Chemistry, University of Vienna, Vienna, Austria**
`will@tbi.univie.ac.at`

Abstract

While computational RNA secondary structure prediction is an important tool in RNA research, it is still fundamentally limited to pseudoknot-free structures (or at best very simple pseudoknots) in practice. Here, we make the prediction of complex pseudoknots – including kissing hairpin structures – practically applicable by reducing the originally high space consumption. For this aim, we apply the technique of sparsification and other space-saving modifications to the recurrences of the pseudoknot prediction algorithm by Chen, Condon and Jabbari (CCJ algorithm). Thus, the theoretical space complexity of free energy minimization is reduced to $\Theta(n^3 + Z)$, in the sequence length n and the number of non-optimally decomposable fragments ("candidates") Z. The sparsified CCJ algorithm, `sparseCCJ`, is presented in detail. Moreover, we provide and compare three generations of CCJ implementations, which continuously improve the space requirements: the original CCJ implementation, our first modified implementation, and our final sparsified implementation. The two latest implementations implement the established HotKnots DP09 energy model. In our experiments, using 244GB of RAM, the original CCJ implementation failed to handle sequences longer than 195 bases; `sparseCCJ` handles our pseudoknot data set (up to about length 400 bases) in this space limit. All three CCJ implementations are available at `https://github.com/HosnaJabbari/CCJ`.

1998 ACM Subject Classification J.3 Life and Medical Sciences

Keywords and phrases RNA, secondary structure prediction, pseudoknots, space efficiency, sparsification

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.12

1 Introduction

Computational RNA secondary structure prediction has become an indispensable tool in the research on non-coding RNAs. Such RNAs perform essential roles – most prominently in regulating gene expression – in all kingdoms of live, in many cases mediated by their three-dimensional structures [10]. Despite the ubiquity of pseudoknots in these RNAs, most often only pseudoknot-free structure prediction methods are applied in biological research – severely limiting the practical capabilities to correctly predict, recognize and compare pseudoknotted structures.

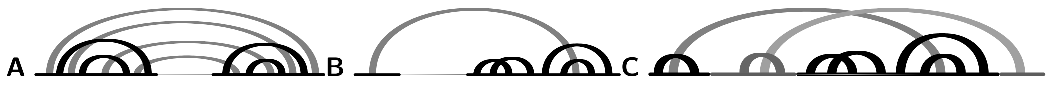


Figure 1 Examples of TGB and CCJ structures. Each arc represents a band of base pairs, which cross other bands. (A) TGB structure with two left, two right, and four middle bands; (B) TGB with nested substructure; (C) CCJ structure composed of two TGB structures (see decomposition of P).

The fundamental cause of this limitation is the computational complexity of pseudoknot prediction – an NP-hard problem [1, 7, 8]. Thus, the high complexity of pseudoknot prediction can be overcome only by either heuristics without optimality-guarantees or restrictions on the predictable pseudoknot class. In comparison, predicting minimum free energy (MFE) structures without pseudoknots is easy (due to tree-like dependencies): a pseudoknot-free secondary structure is either closed by a base pair connecting the first and last base in the sequence, or can be partitioned into two independent substructures on a prefix and suffix of the sequence, where energies of substructures add up to the total energy. As a result of the simple decomposition scheme, the MFE pseudoknot-free secondary structure prediction problem is solved by dynamic programming in $\Theta(n^3)$ time and $\Theta(n^2)$ space for standard energy loop models [12].

The most general dynamic programming algorithm for MFE prediction of pseudoknotted structures, Pknots, was proposed by Rivas and Eddy [14]. Pknots is a complex dynamic programming algorithm with time complexity of $\Theta(n^6)$, and space complexity of $\Theta(n^4)$. There are algorithms for predicting MFE pseudoknotted secondary structures that run in $\Theta(n^5)$ time and $\Theta(n^4)$ space [18, 8]. These algorithms handle a severely limited class compared to the Rivas and Eddy's algorithm. All can handle H-type pseudoknots, and some can handle kissing hairpin structures when these do not have arbitrary nested substructures [18]. There are also some algorithms that run in $\Theta(n^4)$ time [13, 8]; these handle classes of structures that are even more restricted than the $\Theta(n^5)$ algorithms. However, none of these algorithms handle kissing hairpin structures with arbitrary nested substructures. This is a serious, practically relevant limitation, given the biological importance of such structures [4, 9, 19].

We previously proposed a novel MFE-based algorithm, called CCJ [5], which significantly expands the class of structures that can be handled in $O(n^5)$ time. We described a more general method of formulating the dynamic programming recurrences for prediction of pseudoknotted RNA secondary structures that cover gapped regions. To improve the time complexity to $O(n^5)$ we introduced two new ideas into the dynamic programming recurrences: (i) a new class of structures called TGB structures (see Figs. 1A and 1B), with at most three groups of bands, and (ii) recurrences that handle TGB structures by transferring to the left, right, middle or the outer bands. By overlaying TGB structures (Fig. 1C), CCJ covers H-type pseudoknotted structures, kissing hairpins, and chains of four interleaved stems; moreover it recursively handles nested substructures of these types; we simply called this class of structures, *CCJ structures.*

We previously compared CCJ's prediction accuracy versus HotKnots V2.0 [2] and IPknot [16], and showed that CCJ outperforms these algorithms on some of our data sets [6]. While CCJ predicts such complex pseudoknotted secondary structures in $\Theta(n^5)$ time and $\Theta(n^4)$ space, which is a significant improvement over the existing MFE-based algorithms [18, 8], in practice it fails to run on sequences longer than 195 bases even given 244GB of RAM (as in our experiments).

Here, we apply the technique of sparsification [20, 3] to the CCJ algorithm with the main goal to reduce its extreme space complexity of $\Theta(n^4)$. Devising sparsified recurrences of

CCJ, resulting in the algorithm `sparseCCJ`, we improve the space complexity to $\Theta(n^3 + Z)$, where Z is the total number of candidates. This complexity is the result of replacing all four-dimensional dynamic programming matrices by (a constant number of) three-dimensional matrix slices and lists of *candidates*. This is still sufficient to calculate exactly the same optimal solutions as before, which can be shown based on inverse triangle inequality. The number of candidates is expected to be much smaller than the number of replaced matrix entries; moreover it cannot be larger.

The space-efficient retrieval of the optimal structure from this algorithm is enabled by implementing our recently presented technique of space-efficient sparse traceback [21]; this method requires additional space for trace arrows, but keeps the number of trace arrows low due to techniques like garbage collection.

Previous applications of the sparsification technique to RNA structure prediction discussed pseudoknot-free methods [20, 3] or used pseudoknotted methods with simplified energy models (base pair maximization only) [11]. Sparsified RNA–RNA interaction prediction [15] is the most complex case of structure prediction so far that was implemented for a realistic interaction energy model. While space-efficient sparsification is discussed in the paper, the space-efficient variant of the implementation could not recover the optimal interaction structure by traceback.

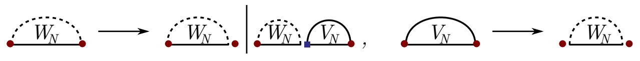
Contributions

We present – to the best of our knowledge – the first space-efficient sparsification of any pseudoknot prediction algorithm that uses a realistic, practically relevant pseudoknot energy model (with according parametrization). Moreover, we sparsify the particularly powerful pseudoknot prediction algorithm CCJ that covers the biologically important kissing hairpins. We implemented – in addition to the original CCJ implementation – a first space-improved CCJ variant and the resulting sparsified algorithm `sparseCCJ`, both using the current HotKnots DP09 energy model [2]. By comparing all three CCJ implementations, we study the (length-dependent) impact of various space savings and finally show that sparsification significantly improves the space requirements over non-sparse implementations of CCJ.

2 The original CCJ pseudoknot prediction algorithm

The original CCJ algorithm [5] is a dynamic programming algorithm (DP) that minimizes the free energy over all CCJ structures for a given input sequence S. As stated in the introduction, CCJ structures comprise kissing hairpins and chains of four interleaved stems, which can recursively contain CCJ structures as substructures. The optimal CCJ structure is then determined by standard traceback through the DP matrices.

Generally, DP algorithms can be described by presenting the recurrences that are used to calculate the entries of their DP matrices. In the case of RNA structure prediction, the DP matrices store minimum free energy (MFE) values for sequence fragments (under specific conditions). The reccurrences correspond to decompositions of these fragments such that the matrix entries can be recursively inferred from energies of smaller subproblems. For example, assuming an energy value of -1 for each canonical base pair, i.e. C–G, A–U or G–U, the MFE structure of an RNA sequence, S, can be found using matrices W_N and V_N, where the respective entries $W_N(i,j)$ and $V_N(i,j)$ are decomposed as follows



(1)

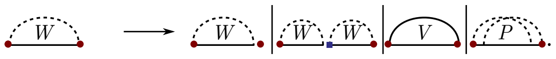
These grammar-rules represent a complete case distinction of possible structures. In the example of Eq. (1), $W_N(i,j)$ corresponds to the MFE structure from base i to base j; this structure can be decomposed according to Eq. (1), since either j is unpaired (left case) or j is paired to some inner position (recursion to V_N, in which solid arc represent a base pair); the closed structure corresponding to $V_N(i,j)$ is reduced to $W_N(i+1,j-1)$ (rightmost case). Moreover, the grammar rules allow – almost mechanically – inferring the recursion equations for base pair energy minimization. The graphical notation is designed to encode the required information: red dots on the right side always correspond to red dots on the left side of the rule, while blue squares mark *free* position indices. Generally, our recursions marginalize (i.e. minimize) over the recursion cases and the free indices in their respective range limited by the fixed indices. Thus, we translate the rules of Eq. (1) to

$$W_N(i,j) = \min\{W_N(i,j-1), \min_{i\le k<j} W_N(i,k-1) + V_N(k,j)\}\,,$$
$$V_N(i,j) = W_N(i+1,j-1) - 1 \qquad \text{if } S_i\text{–}S_j \text{ is canonical, } V_N(i,j) = \infty \text{ otherwise.}$$

In our discussion of CCJ and its sparsification, this level of presentation allows us to focus on the sparsification and avoid distracting details like the exact added energy contributions in each single step, which is not necessary for understanding the sparsification.

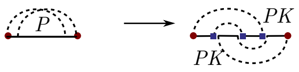
While pseudoknot-free recursions generally use only fragments that are connected at the sequence level, the CCJ algorithm requires ‘gapped’ fragments, where two subsequences are disconnected by a gap. Consequently, defining such fragments requires four sequence positions in total.

The MFE of the CCJ structure for subsequence $S_{i..l}$ is calculated in $W(i,l)$, which is decomposed as



The recurrence for $V(i,l)$ handles different types of loops closed by i and l; $P(i,l)$ is the minimum free energy of a CCJ pseudoknot for region $[i,l]$.

$P(i,l)$ is decomposed by the rule

 (2)

into two TGB structures (with MFEs in the PK matrix). Representing TGB structures requires using gapped fragments.

As indicated by the three blue boxes (free indices) in Eq. (2), each entry of P is minimized over three indices. Thus, already this step bounds the time and space complexity of the CCJ algorithm to $O(n^5)$ and $O(n^4)$ respectively.

$PK(i,j,k,l)$ is the MFE over all TGB structures of the gapped fragment $[i,j]\cup[k,l]$. Note that such structures have the additional restrictions: the positions i and l must be involved in some base pairs, which are not part of nested substructures; moreover, some base pair of a TGB structure must span the gap. The recurrence for PK uses terms P_L, P_M, P_O, and P_R, which (put informally) handle bands on the left, middle and right groups of the TGB structure, respectively. Both P_M and P_O are needed to handle bands in the middle group. The matrix entries $P_L(i,j,k,l)$, $P_M(i,j,k,l)$, $P_R(i,j,k,l)$, $P_O(i,j,k,l)$ are decomposed as illustrated in Fig. 2, in which WP, handles nested substructures in a pseudoloop. (For invalid index combinations, the matrix entries are set to $+\infty$, and do not have to be stored.)

Each of the matrices P_L, P_R, P_M, and P_O requires a base pair between the two ends at the respective positions left, right, middle, or outer. Each such matrix distinguishes the three

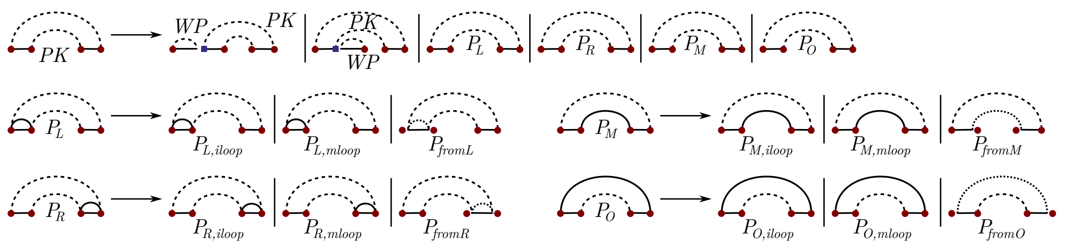


Figure 2 Decompositions for $PK(i,j,k,l)$, $P_L(i,j,k,l)$, $P_M(i,j,k,l)$, $P_R(i,j,k,l)$, $P_O(i,j,k,l)$ in grammar-rule like graphical notation.

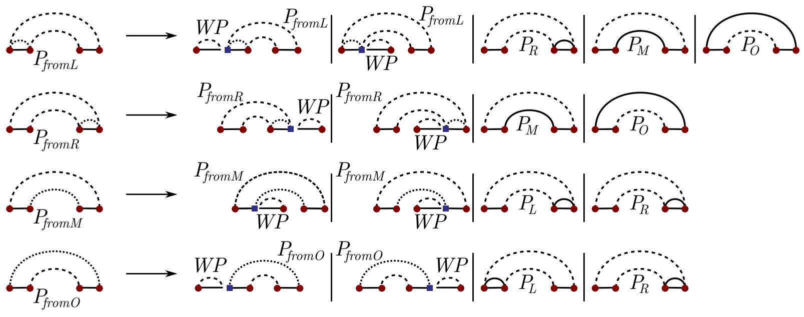
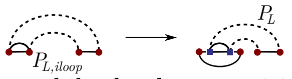


Figure 3 Decompositions of $P_{fromX}(i,j,k,l)$ for $X \in \{L,R,M,O\}$, which handle transitions from P_X in other matrices P_Y in graphical notation.

cases that this base pair closes an interior loop or a multi-loop or is the inner border of a band. In the latter case, the respective terms P_{fromX} ($X \in \{L,R,M,O\}$) handle transitions from base pairs in one group to base pairs in some other group (see Fig. 3).

Whenever we change a band via one of the matrices P_{fromX}, we must allow for nested substructures around the band. This is handled by the first two cases of the respective recurrence P_{fromX}. Moreover, in P_{fromX} we recurse to matrices $P_X(i,j,k,l)$ only if the requirements of these matrices are met.

Note that in P_{fromL}, it is not possible to transition to P_L. This is because the recurrences are designed so that bands are handled in rounds. Within a round, bands in the left are handled first, if any, then those on the right, if any, and then those on the middle, with bands handled by P_M (if any) handled before those handled by P_O. A middle band *must* be handled in each round; otherwise, for example, two "bands" on the left group, added in different rounds, would collapse into one, causing the recurrences to incorrectly add penalty terms for band "borders" that are not actually borders. For this reason, no row in P_{fromL} has a P_L term, and so a band on the left group cannot be handled directly after a band on the right group. Also, P_{fromO} does not have a row with a P_M term, to ensure that P_M cannot be used twice on the same round.

Interior loops in the left band are decomposed by the rule 

; the remaining cases (right, middle, outer) are analogous. Note that, while the decomposition of $P_{L,iloop}(i,j)$ has two free indices, these indices are constraint by setting a constant maximum size of interior loops (here 30 bases) as it is common practice. For handling interior loops, the original CCJ algorithm introduced the five-ary function $P_{L,iloop5}$ and applied a clever scheme to still use only $\Theta(n^4)$ space. However, this space consumption could not easily be reduced further by sparsification. Thus, avoiding $P_{L,iloop5}$, which is possible due to the HotKnots energy model, is essential to reduce the space complexity.

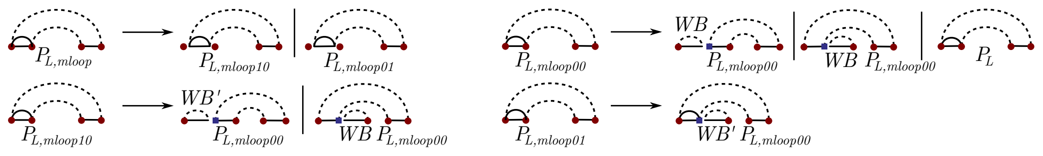


Figure 4 Decomposition of multi-loops in the left band.

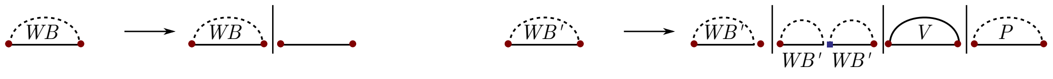


Figure 5 Decompositions of $WB(i,l)$ and $WB'(i,l)$.

Moreover, we modify the original handling of multi-loop cases to enable their sparsification. Fig. 4 shows our multi-loop handling for the left band. We handle multi-loops by passing through states $P_{L,mloop10}$, $P_{L,mloop01}$, and $P_{L,mloop00}$, which keep track of introduced inner multi-loop base pairs on the left or on the right. Finally, Fig. 5 shows the decompositions of $WB(i,l)$ and $WB'(i,l)$. The other "W"-matrices are decomposed analogously.

Further details of the original CCJ recurrences are available in the thesis [6], which also provides a detailed description of its sparsification.

3 Sparsification of the CCJ algorithm

By and large, "sparsification" allows keeping just the *required* dynamic programming matrix cells, which we refer to as *candidates*, (instead of the whole matrix) to find the MFE value. By storing a few candidates (as explained below) we avoid storing any four-dimensional CCJ matrix. (i) In recurrences in which the left-most index, i, does not change, we store the value of such recurrences in a constant number of three-dimensional matrix slices; we call the collection of these matrix slices *i-slices*. In many of these recursion cases, we recurse to matrix entries of the same i-slice (e.g. when inferring PK from P_L or, slightly more interestingly, P_R from P_{fromR}). In other cases, we recurse to the $(i+1)$-slice (e.g. P_L from P_{fromL}) or $(i+c)$-slice, where c is constantly bounded. The latter occurs in the handling of interior loops ($P_{L,iloop}$ and analogously $P_{O,iloop}$), where c does not exceed the maximum interior loop size MLS. (ii) This leaves us with the recursion cases that recurse to some d-slice, where $d-i$ cannot be constantly bounded. Instead of storing all slices, we store only certain candidate entries in such slices. These matrix entries (called *candidates*) are stored in *candidate lists* for specific recursion cases together with their corresponding second, third, and fourth matrix indices, j,k, and l to keep track of band borders. In some cases, candidate lists can be shared between recursion cases. We presented more details on candidate list requirements in [11].

Space representation of the four-dimensional matrices by `sparseCCJ`

Only matrices corresponding to P_L and P_O occur in interior loops that span a band, and require to recurse to a different i-slice; for them, we store slices $i..i+MLS$. For matrices corresponding to P_{fromL}, P_{fromO}, $P_{L,mloop10}$, $P_{L,mloop01}$, $P_{O,mloop10}$, and $P_{O,mloop01}$, we only need to store slices i and $i+1$. Matrices corresponding to recurrences of types $P_{X,iloop}$ and $P_{X,mloop}$ ($X \in \{L,R,M,O\}$) do not need to be stored and can be computed when needed. For the remaining matrices, we store only the current i-slice. Note that space is always reused in the next iteration; in case of ranges of slices, the memory access is 'rotated' without copying in memory. Matrices corresponding to the following recurrence cases require maintaining candidate lists: PK, P_L, P_O, P_{fromL}, P_{fromO}, $P_{L,mloop00}$, and $P_{O,mloop00}$.

To finalize sparsification, all recursion cases that recurse to these matrices – where the left-most index is not constantly bound – need to be modified. This affects all recursion cases, which insert any nested substructure to the left of the gapped region. This occurs exactly in the decompositions of PK, P_{fromL}, P_{fromO}, $P_{L,mloop10}$, $P_{L,mloop00}$, $P_{O,mloop10}$, and $P_{O,mloop00}$. Moreover, this affects the decomposition of P into two PK-fragments, where the latter is taken from the respective candidate list. We discuss the single modifications on the three examples of PK, $P_{L,mloop10}$, and P – the remaining cases are sufficiently similar to be sparsified analogously.

1. Consider the case of the P_{fromL} recurrence:

$$\min_{i<d\leq j} WP(i,d-1) \ + \ PK(d,j,k,l).$$

It suffices to minimize only over certain candidates $PK(d,j,k,l)$ that are *not optimally decomposable* in the following sense:

$$\nexists e > d : PK(d,j,k,l) = WP(d,e-1) \ + \ PK(e,j,k,l). \quad (3)$$

It can be shown easily that whenever $PK(i,j,k,l)$ is optimally decomposable, there is a candidate (i.e. a smaller, not optimally decomposable fragment) which yields the same minimum value. Remarkably, the candidate criterion can be efficiently checked by the dynamic programming algorithm. This is more directly seen from the equivalent candidate criterion

$$PK(d,j,k,l) < min_{d<e\leq j} WP(d,e-1) \ + \ PK(e,j,k,l).$$

Also this minimum can be computed by running only over candidates; moreover it must be calculated by the dynamic programming algorithm for computing $PK(d,j,k,l)$, such that the check is performed without additional overhead. This idea holds analogously for the other candidate checks.

2. Similarly, the minimization

$$\min_{i<d\leq j} WB'(i,d-1) + P_{L,mloop00}(d,j,k,l)$$

that occurs in the recurrence of $P_{L,mloop10}$ is restricted to candidates that satisfy

$$\nexists e : P_{L,mloop00}(d,j,k,l) = WB(d,e-1) + P_{L,mloop00}(e,j,k,l). \quad (4)$$

We can even strengthen the criterion, such that candidates must further satisfy

$$\nexists e : P_{L,mloop00}(d,j,k,l) = P_{L,mloop00}(d,e,k,l) + WB(e+1,j). \quad (5)$$

3. In the minimization calculating $P(i,l)$, i.e.

$$\min_{i<j<d<k<l} PK(i,j-1,d+1,k-1) + PK(j,d,k,l),$$

it suffices to consider only candidates $PK(j,d,k,l)$. Entries $PK(j,d,k,l)$ are candidates if and only if they are *not optimally decomposable* in the following sense:

$$\nexists e(j \leq e < d) : PK(j,d,k,l) = PK(j,e,k,l) + WP(e+1,d). \quad (6)$$

We emphasize that certain cases share the same candidates (allowing space savings). For example, the candidate criterion for decomposing $P_{L,mloop10}$ into WB' and $P_L, mloop00$ is identical to the one of decomposing $P_{L,mloop00}$ into WB' and $P_L, mloop00$. Similarly, we share the candidate lists of $P_{O,mloop10}$ and $P_{O,mloop00}$.

Finally, the sparsified CCJ recurrences can be computed based on the (constantly bounded) matrix slices and the candidates alone. Their correctness is a consequence of inverse triangle inequalities; for example in case of W we have $\forall x < y \leq z : W(x,z) \leq W(x,y-1) + W(y,z)$, which follows from the definition of W. Analogous inequalities hold for WP, WB, and WB.

▶ **Theorem 1** (Correctness of the CCJ sparsification). *The sparsified CCJ recurrences are equivalent to the non-sparsified CCJ recurrences.*

Proof. We show for fragments i,l and i,j,k,l by simultaneous induction on the fragment size (respectively, $l-i$ and $j-i+l-k$) that all changes in the definition of `sparseCCJ` (from the original to the sparsified recurrences of CCJ) are equivalent, in particular the values of the CCJ recursions and their corresponding sparsified versions are identical for each fragment.

It suffices to show the equivalence of the changes of the minimization cases explicitly. Moreover in each case, it suffices to show that there exists a minimum that is a candidate. By the induction hypothesis, the sparsified recursion for all smaller fragments do not have to be distinguished from the non-sparsified ones. We prove the three example cases PK, $P_{L,mloop10}$, and P explicitly; the remaining cases can be shown analogously.

1. Choose the largest d, $i < d \leq j$, s.t. $WP(i,d-1) + PK(d,j,k,l)$ is minimal. We show – by contradiction – that $PK(d,j,k,l)$ is a candidate, i.e. it satisfies Eq. (3). Assume Eq. (3) does not hold and choose e ($e > d$) such that $PK(d,j,k,l) = WP(d,e-1) + PK(e,j,k,l)$. Now, $WP(i,d-1) + PK(d,j,k,l) = WP(i,d-1) + WP(d,e-1) + PK(e,j,k,l) \geq WP(i,e-1) + PK(e,j,k,l)$ (using the inverse triangle inequality for WP). This contradicts the choice of d; thus $PK(d,j,k,l)$ must be a candidate.
2. Choose the largest d s.t. $WB'(i,d-1) + P_{L,mloop00}(d,j,k,l)$ is minimal. We show – by contradiction – that the candidate criterion, i.e. Eqs. (4) and (5), hold.
   - Assume Eq. (4) does not hold, then there exists some e s.t. $WB'(i,d-1)+P_{L,mloop00}(d,j,k,l) = WB'(i,d-1)+WB(d,e-1)+P_{L,mloop00}(e,j,k,l) \geq WB'(i,e-1) + P_{L,mloop00}(e,j,k,l)$; the latter inequality holds due to the definition of WB'.
   - Assume Eq. (5) does not hold, then there exists some e s.t. $P_{L,mloop00}(d,j,k,l) = P_{L,mloop00}(d,e,k,l) + WB(e+1,j)$. Consequently, the corresponding case of $P_{L,mloop10}(i,j,k,l)$ yields a smaller or equal value, since $WB'(i,d-1) + P_{L,mloop00}(d,j,k,l) = WB'(i,d-1) + P_{L,mloop00}(d,e,k,l) + WB(e+1,j) \geq P_{L,mloop10}(i,e,k,l) + WB(e+1,j)$.
3. For fixed $i < j < k < l$, choose the smallest d ($j < d < k$) s.t. $PK(i,j-1,d+1,k-1) + PK(j,d,k,l)$ is minimal. Assume Eq. (6) is violated, then there is some $e > d$, s.t. $PK(i,j-1,d+1,k-1) + PK(j,d,k,l) = PK(i,j-1,d+1,k-1) + PK(j,e,k,l) + WP(e+1,d) \geq_{(**)} PK(i,j-1,e+1,k-1) + PK(j,e,k,l)$, which contradicts the choice of d. For inequality (**), we observe that $PK(i,j-1,d+1,k-1) + WP(e+1,d) \leq PK(i,j-1,e+1,k-1)$ by definition of PK. ◀

Note that the presented sparsification would not have been possible for the multiloop handling of the original CCJ algorithm (Fig. 6), which required us to modify these decompositions. In the original decomposition of $P_{L,mloop}$, the unbound access to d-slices

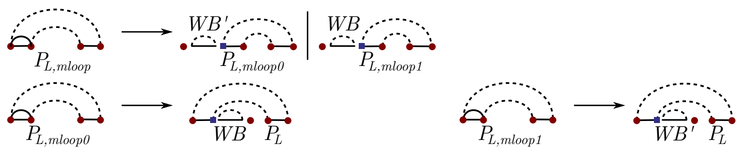


Figure 6 Decomposition of multi-loops in the left band by the original CCJ algorithm.

cannot easily be replaced by candidates – note the index offsets in the graphical notation, indicating that in the recurrence of $P_{L,mloop}(i,j,k,l)$, the WB' and WB fragments both start at $i+1$; similarly, there is a shift to $j-1$ in the recurrences of $P_{L,mloop0}(i,j,k,l)$ and $P_{L,mloop1}(i,j,k,l)$.

4 Space Complexity Analysis

In the previous sections, we sparsified all four-dimensional matrices of the CCJ algorithm with the goal of reducing its space complexity. As explained before, our sparsification allows us to replace all four-dimensional matrices by three-dimensional matrix slices. In seven recursion cases, we needed to rewrite minimizations, such that they compute equivalent results by recursing only to candidates (or entries of the same i-slice). In two of these cases, candidate lists can be shared.

Even if only a small fraction of the respective four-dimensional fragments are optimally decomposable (i.e. are not candidates), these changes will save space over the non-sparsified version. However, experience from previous sparsification (and our results) show that a large number of fragments is optimally decomposable, such that number of candidates is small.

We define Z as the total number of candidates. For traceback, we store a number of trace arrows, dynamically limiting their number; denote their maximum number by T. Then, the total space complexity of `sparseCCJ` is $O(n^3 + Z + T)$.

5 Results

In this section we provide implementation and data set details, and show a comparison of `sparseCCJ` in terms of time and memory usage to its predecessors, original CCJ and modified CCJ.

5.1 Implementation

We implemented three versions of CCJ algorithm in C++; the first version is strictly based on the original CCJ recurrences, but the energy values are similar to that of DP09 energy model in HotKnots V2.0 [2]; we refer to this version as “original”. There are few energy functions in the original CCJ energy model that are not explicitly in the DP09 model. We have set values of these functions to 0, in order to make the models as similar as possible. The second version has modified recurrences to *match* the energy model of HotKnots V2.0, and is referred to as “modified”. The main difference between this version and the original version is in calculating energy of interior loops that span the band. The energy of an interior loop or multi-loop depends on whether or not the external base pair of the loop is pseudoknotted. If it is not, we call the loop *ordinary*, and otherwise say that the loop *spans a band.* If the external base pair of an ordinary interior loop, not including stacks, is $i.l$ and the other closing base pair is $d.e$, then similar to the DP09 energy model, the energy of the interior

Table 1 Summary of data sets used in this work.

Name	# of sequences	Structure Type	sequence lengths	Reference
HK-PK	88	pseudoknotted	26–400	test set of [2]
HK-PK-free	337	pseudoknot-free	10–194	test set of [2]
IP-pk168	168	pseudoknotted	21–137	test set of [16]
DK-pk16	16	pseudoknotted	34–377	test set of [17]

loop is calculated by a call to the function $e_{int}(i,d,e,l)$. If the internal loop spans a band we call the function $e_{intP}(i,d,e,l)$ instead. The third version uses the sparsification technique, and is referred to as "sparse" (`sparseCCJ`). Similar to [21], we utilize trace arrows to keep track of accessible cells and employ a garbage collection technique to remove trace arrows from unreachable cells. Using these techniques in `sparseCCJ` required extensions like trace arrows between different matrices, which were only briefly mentioned in [21]; otherwise it demonstrates the generality of this approach.

5.2 Data sets

We use a large data set of over 600 RNA strands of length between 10–400 bases to analyze the performance of our algorithm. This data set was compiled from three non-overlapping data sets with various pseudoknotted and pseudoknot-free structures. Table 1 provides a summary of these data sets.

5.3 Benchmark Results

We ran all three versions of CCJ implementations on Amazon Cloud (r4.8xlarge instance consisting of 32 Xeon E5-2686 Broadwell 2.3 GHz CPUs, and 244GB of DDR3 RAM) and compared their time and memory requirements for instances of our data set. First, we verified that the two versions that implement DP09 energy model of HotKnots V2.0, i.e. the modified CCJ and the sparse CCJ implementation, indeed produced exactly the same results. This equivalence must hold in correct implementations due to Theorem 1. We focus on (run time and space) *performance* of sparse CCJ, in this work. Fig. 7 shows – in log scale – the memory consumption (left), our main objective in this work, and run times (right), both as functions of sequence length. We observe significant improvements in space from the original implementation over the modified one to the sparse implementation. At the same time, one observes a comparably small run time penalty in our (little run time-optimized) sparse implementation. However, given today's heavily parallel computation platforms (with comparably costly main memory), differences in run-time are generally less relevant than space improvements.

We further investigated whether a specific class of structures (i.e. pseudoknotted vs. pseudoknot-free) would benefit stronger than the other from sparsification, and found out that both classes benefit equally from sparsification (see Fig. 8); in general longer sequences benefit more (as seen in Fig. 7). Closer look at Fig. 7 shows that while `sparseCCJ` has variation in memory usage within the same length, these variations are minimal. We looked at numbers of candidates and trace arrows (9) in sparse CCJ, which together explain the space requirements.

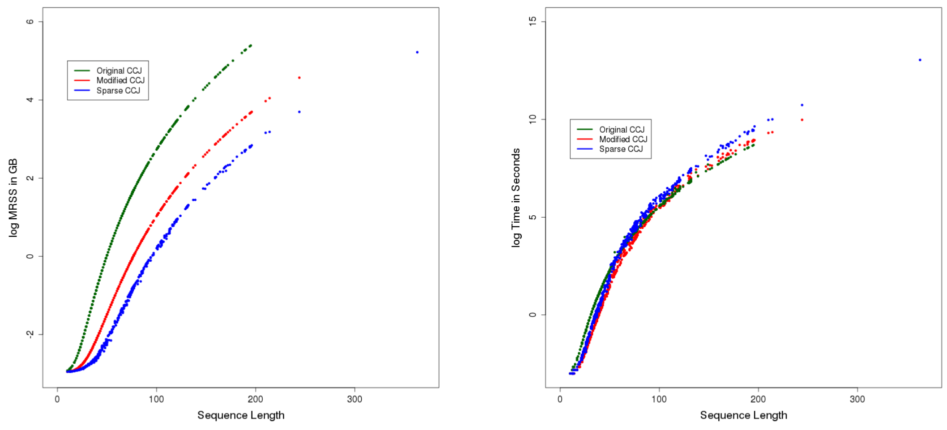


Figure 7 Memory usage (left, presented as log of maximum resident set size in GB) and run time (right, presented as log of time in second) vs. length for the three CCJ implementations.

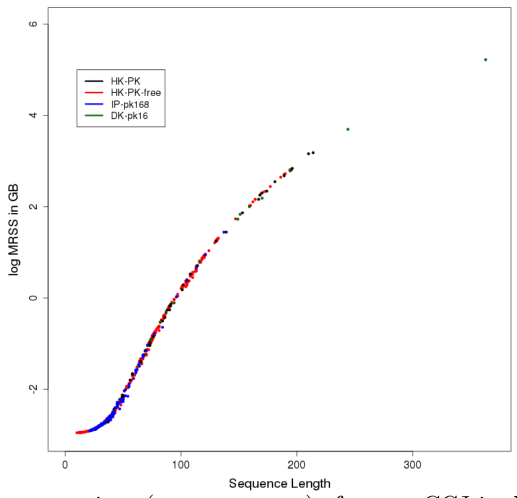


Figure 8 Performance comparison (memory usage) of sparse CCJ in different class of structures.

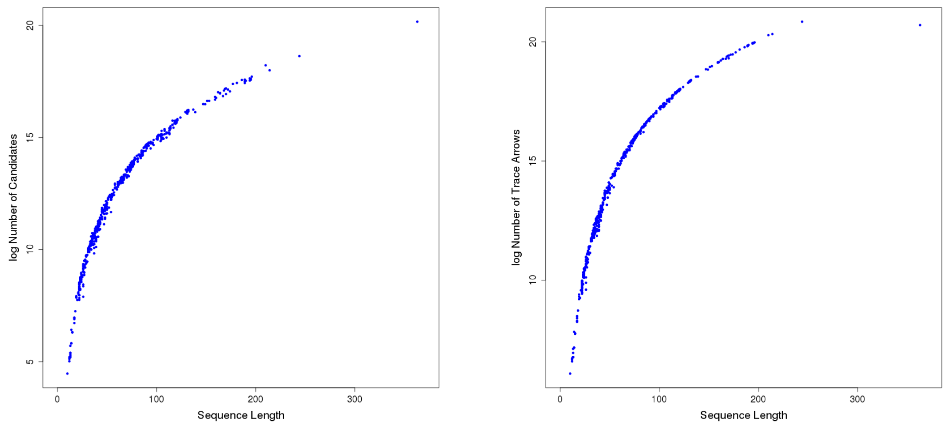


Figure 9 Total number of candidates (measured as log(total number of candidates)) and trace arrows (measured as log(maximum number of trace arrows)) used in SparseCCJ versus sequence length.

6 Conclusion

We have presented the first application of the sparsification to a complex pseudoknot structure prediction algorithm – supporting kissing hairpins with arbitrarily nested substructures – with a realistic energy model. While previous applications of the sparsification technique mainly focused on speed improvements, we solely aimed at reducing the space requirements, which is the main factor limiting the practical applicability of complex RNA pseudoknotted secondary structure prediction. Our comparison to two previous CCJ variants provides interesting insights into general potentials for space improvements of complex RNA-related algorithms. Finally, our space savings in `sparseCCJ` open the door for large scale biologically-relevant application of pseudoknot structure prediction covering all important pseudoknot classes.

Acknowledgements. We thank Anne Condon for discussing details of the CCJ algorithm and first ideas on space savings.

References

1 T. Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Disc. App. Math.*, 104(1–3):45–62, 2000.

2 M. S. Andronescu, C. Pop, and A. E. Condon. Improved free energy parameters for RNA pseudoknotted secondary structure prediction. *RNA (New York, N.Y.)*, 16(1):26–42, January 2010.

3 R. Backofen, D. Tsur, S. Zakov, and M. Ziv-Ukelson. Sparse RNA folding: Time and space efficient algorithms. *Journal of Discrete Algorithms*, 9(1):12–31, March 2011. `doi:10.1016/j.jda.2010.09.001`.

4 K.-Y. Chang and I. Tinoco. The structure of an RNA kissing hairpin complex of the HIV TAR hairpin loop and its complement. *Journal of Molecular Biology*, 269(1):52–66, May 1997. `doi:10.1006/jmbi.1997.1021`.

5 H. L. Chen, A. Condon, and H. Jabbari. An o(n(5)) algorithm for MFE prediction of kissing hairpins and 4-chains in nucleic acids. *Journal of computational biology : a journal of computational molecular cell biology*, 16(6):803–815, June 2009. `doi:10.1089/cmb.2008.0219`.

6 H. Jabbari. *Algorithms for prediction of RNA pseudoknotted secondary structures.* PhD thesis, University of British Columbia, March 2015.

7 R. B. Lyngsø. Complexity of pseudoknot prediction in simple models. In *ICALP'04*, pages 919–931, 2004.

8 R. B. Lyngsø and C. N. Pedersen. RNA pseudoknot prediction in energy-based models. *J. Comput. Biol.*, 7(3-4):409–427, 2000.

9 W. J. Melchers, J. G. Hoenderop, H. J. Bruins Slot, C. W. Pleij, E. V. Pilipenko, V. İ. Agol, and J. M. Galama. Kissing of the two predominant hairpin loops in the coxsackie B virus 3' untranslated region is the essential structural feature of the origin of replication required for negative-strand RNA synthesis. *Journal of Virology*, 71(1):686–696, January 1997.

10 T. R. Mercer, M. E. Dinger, and J. S. Mattick. Long non-coding RNAs: insights into functions. *Nature Reviews Genetics*, 10(3):155–159, March 2009. `doi:10.1038/nrg2521`.

11 M. Möhl, R. Salari, S. Will, R. Backofen, and S. C. Sahinalp. Sparsification of RNA structure prediction including pseudoknots. *Algorithms for Molecular Biology*, 5(1):39+, December 2010. `doi:10.1186/1748-7188-5-39`.

12 R. Nussinov and A. B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, November 1980. `doi:10.1073/pnas.77.11.6309`.

13 J. Reeder and R. Giegerich. Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, 5, 2004.

14 E. Rivas and S. R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.*, 285(5):2053–2068, 1999.

15 R. Salari, M. Möhl, S. Will, S. C. Sahinalp, and R. Backofen. Time and Space Efficient RNA-RNA Interaction Prediction via Sparse Folding. In Bonnie Berger, editor, *Research in Computational Molecular Biology*, volume 6044 of *Lecture Notes in Computer Science*, chapter 31, pages 473–490. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010. `doi:10.1007/978-3-642-12683-3_31`.

16 K. Sato, Y. Kato, M. Hamada, T. Akutsu, and K. Asai. IPknot: fast and accurate prediction of RNA secondary structures with pseudoknots using integer programming. *Bioinformatics*, 27(13):i85–i93, July 2011.

17 J. Sperschneider, A. Datta, and M. J. Wise. Predicting pseudoknotted structures across two RNA sequences. *Bioinformatics (Oxford, England)*, 28(23):3058–3065, December 2012.

18 Y. Uemura, A. Hasegawa, S. Kobayashi, and T. Yokomori. Tree adjoining grammars for RNA structure prediction. *Theor. Comput. Sci.*, 210(2):277–303, 1999.

19 M. H. Verheije, R. C. L. Olsthoorn, M. V. Kroese, P. J. M. Rottier, and J. J. M. Meulenberg. Kissing interaction between 3' noncoding and coding sequences is essential for porcine arterivirus RNA replication. *Journal of Virology*, 76(3):1521–1526, February 2002. `doi:10.1128/jvi.76.3.1521-1526.2002`.

20 Y. Wexler, C. Zilberstein, and M. Ziv-Ukelson. A study of accessible motifs and RNA folding complexity. *Journal of computational biology: a journal of computational molecular cell biology*, 14(6):856–872, 2007. `doi:10.1089/cmb.2007.r020`.

21 S. Will and H. Jabbari. Sparse RNA folding revisited: space-efficient minimum free energy structure prediction. *Algorithms for molecular biology: AMB*, 11, 2016.