Infrared: a declarative tree decomposition-powered framework for bioinformatics

Hua-Ting Yao, Bertrand Marchand, Sarah J. Berkemer, Yann Ponty, Sebastian Will

Infrared: a declarative tree decomposition-powered framework for bioinformatics

Hua-Ting Yao[1,2,3*], Bertrand Marchand[1], Sarah J. Berkemer[1,4], Yann Ponty[1], Sebastian Will[1*]

[1*]LIX, CNRS UMR 7161, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France .
[2*]Department of Theoretical Chemistry, University of Vienna, Vienna, Austria .
[3]School of Computer Science, McGill University, Montreal, Canada .
[4]Earth-Life Science Institute, Tokyo Institute of Technology, Tokyo, Japan .

*Corresponding author(s). E-mail(s): htyao@tbi.univie.ac.at; sebastian.will@polytechnique.edu;

Abstract

Motivation: Many bioinformatics problems can be approached as optimization or controlled sampling tasks, and solved exactly and efficiently using Dynamic Programming (DP). However, such exact methods are typically tailored towards specific settings, complex to develop, and hard to implement and adapt to problem variations.
Methods: We introduce the Infrared framework to overcome such hindrances for a large class of problems. Its underlying paradigm is tailored toward problems that can be declaratively formalized as sparse feature networks, a generalization of constraint networks. Classic Boolean constraints specify a search space, consisting of putative solutions whose evaluation is performed through a combination of features. Problems are then solved using generic cluster tree elimination algorithms over a tree decomposition of the feature network. Their overall complexities are linear on the number of variables, and only exponential on the treewidth of the feature network. For sparse feature networks, associated with low to moderate treewidths, these algorithms allow to find optimal solutions, or generate controlled samples, with practical empirical efficiency.
Results: Implementing these methods, the Infrared software allows Python programmers to rapidly develop exact optimization and sampling applications based on a tree decomposition-based efficient processing. Instead of directly coding specialized algorithms, problems are declaratively modeled as sets of variables over finite domains, whose dependencies are captured by constraints and functions. Such models are then automatically solved by generic DP algorithms. To illustrate the applicability of Infrared in bioinformatics and guide new users, we model and discuss variants of bioinformatics applications. We provide reimplementations (and extensions) for methods targeting RNA design, RNA sequence-structure alignment, parsimony-driven inference of ancestral traits in phylogenetic trees/networks, and coding sequence design demonstrate multidimensional Boltzmann sampling. Previous work together with novel results demonstrate the practical relevance of the framework, whose complexity is typically equivalent or better than specialized algorithms and implementations.
Availability: Infrared is available at https://www.lix.polytechnique.fr/~will/Software/Infrared with extensive documentation, including various usage examples and API reference; it can be installed using Conda or from source.

Keywords: Bioinformatics, Fixed-parameter tractable algorithms, Tree decomposition, Boltzmann sampling, Network phylogeny, RNA sequence design, RNA alignment, Pseudoknots

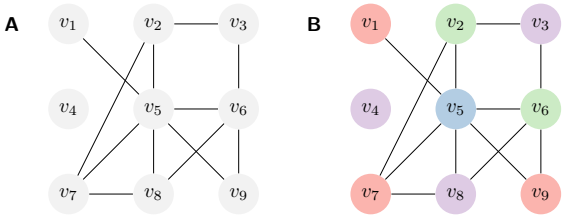

Fig. 1 The running example: graph coloring. **A**) Example input graph. **B**) One valid coloring with 4 colors, corresponding to an assignment of variables to colors (domain values) that satisfies all the inequality constraints along the edges. In our example extension, which minimizes the *feature* counting the different colors in each of its four cycles of length 4, (v_2, v_3, v_5, v_6), (v_2, v_5, v_7, v_8), (v_5, v_6, v_7, v_8) and v_5, v_6, v_8, v_9, this coloring is not optimal (e.g. recolor v_3).

1 Background

Typical applications of computational, "bioinformatical" methods to real world biological problems have inherently high complexity at different levels. For example, these comprise the design of functional control elements for biotechnology [1, 2], identifying homologies in the context of RNA pseudoknots [3, 4], or the prediction of phylogenies considering complex inheritance patterns [5]. First, *modeling complexity* is directly inherited from the complexity of the biological backdrop. This requires bioinformatics approaches to deal with hard constraints and soft requirements. Moreover, many approaches need to target complex scores, often composed of multiple interdependent objectives, e.g. for predicting optimal solutions or generating designs. In turn, the high modeling complexity is reflected in coding challenges and leads to high *computational complexity* of exact solutions. Frequently, this turns bioinformaticians away from exact, combinatorial methods to less controlled heuristics, for example optimization by local search or genetic algorithms [6], or sampling by MCMC (Markov Chain Monte Carlo)-like approaches [7, 8], which sacrifice guarantees on the optimality of solutions or the time and space complexity of the computations.

Here, we introduce the framework Infrared to cope with these levels of complexity. This system lets users specify a large class of computational problems and solves them by combinatorial algorithms with parameterized complexity [9]. These methods guarantee exactness and work efficiently, when the "complexity" (treewidth) of the problem instance is fixed. In practice, this limits the system to problems with moderately sparse dependencies. The system combines various concepts of computer science, comprising constraint modeling [10], constraint and scoring networks [11, 12], tree decomposition [13], parameterized complexity [9, 14], random generation, and (multidimensional) Boltzmann sampling [15]. Along with Infrared, we advocate and hope to promote the use of exact methods. In place of heuristic methods, many NP-hard problems can be solved by algorithms of parameterized complexity, which our system makes more accessible due to proper abstractions. Other problems profit from building heuristic methods on top of exact algorithms.

Historically, the presented framework originated as a generalization of our own previous work on multitarget RNA design (RNARedprint [16]) and our original approach has been completely reimplemented and extended based on Infrared (RNARedprint v2[1]). In parallel to the presented research, we used an early version of the system for original research in the area of negative RNA design (RNAPond, [17]). Other recent work has strong conceptual ties: Tree-Diet [18] (by using RNAPond and LicoRNA [3]) and AutoDP [19]). Moreover, as we show in this text, sequence and RNA sequence-structure alignment can be implemented following the models of LicoRNA [3] and [20]; both papers introduced closely related solving strategies for alignment.

Our framework aims to facilitate the implementation of complex algorithms based on the declarative modeling paradigm. Instead of implementing a concrete algorithm, it allows users to formally describe the problem by specifying the admissible solutions and their quality assessments. Similar to, e.g. constraint programming or integer linear programming systems, those models are solved automatically by a built-in general mechanism.

[1] https://gitlab.inria.fr/amibio/RNARedPrint

Example. Let us illustrate this idea by modeling graph coloring as a Constraint Satisfaction Problem (CSP). We use this 'toy problem' as our running example to formally introduce our main concepts. For this purpose, we will later extend it beyond constraint satisfaction (introducing some quality of colorings).

Given a graph $G = (V, E)$, see Figure 1A, the graph coloring problem asks for a vertex labeling by k colors, such that adjacent vertices are colored differently (Fig. 1B). To solve this classical problem in our system, we model it as a CSP, i.e. as a triple of a set of variables, one domain per variable, and constraints. This CSP introduces one variable per vertex, resulting in the set of variables $\{X_1, \ldots, X_{|V|}\}$. Each variable encodes the label of the corresponding vertex, i.e. it takes values from 1 to k, expressed by choosing the domain $D(X_i) = \{1, \ldots, k\}$ for each X_i. Finally, we define the constraint set consisting of one inequality constraint NotEquals between the variables X_i and X_j for every edge $(i, j) \in E$.

Solving the problem means finding a *valid* assignment of values to variables that satisfies the constraints. Our system supports constraint solving, even if pure constraint solving serves mostly as a basis for further extensions. We can directly express our graph coloring model in Python code.

```
import infrared as ir
# model 9 variables, colors 1..4
model = ir.Model(9,(1,4))
# define NotEquals constraints in Python
ir.def_constraint_class('NotEquals',
    # signature and dependencies
    lambda i,j: [i,j],
    # constraint semantics
    lambda x, y: x!=y)
model.add_constraints(NotEquals(i,j) for i,j in edges)
```

◁

Based on this model, Infrared finds a valid coloring automatically due to its built-in parameterized algorithms in a time that depends on the size of the graph, the number of colors, and the *complexity of* G, i.e. its treewidth. For this purpose, one passes the model to the solver and asks for a valid solution. Since Infrared handles constraint satisfaction and optimization in the same way, its solver is called `Optimizer`.

```
solution = ir.Optimizer(model).optimize()
```

Extending CSPs by features.

Beyond validity, Infrared addresses solution quality in terms of one or several *features*—conceptually, we extend Infrared's models from CSPs to feature networks. This allows users of the framework to more naturally model complex problems with multiple objectives, as they commonly occur in bioinformatics. Based on specified features, Infrared is then able to perform tasks such as optimization and weighted sampling.

Example. As a first feature example, let us pick up graph coloring and additionally minimize the use of different colors in cycles of length 4 (4-cycles). For this purpose, we specify a feature by imposing one 4-ary function `Card` for each 4-cycle that counts the different colors in the cycle (set cardinality); the sum of function values defines the value of the feature.

Infrared's syntax supports the compositional construction and extension of models. After defining the class of functions `Card`, we can therefore add them to the previous model.

```
ir.def_function_class('Card',
    # signature, dependencies
    lambda i,j,k,l: [i,j,k,l],
    # function evaluation
    lambda x,y,z,w: len({x,y,z,w}))
# add the 4-cycle cardinality functions
model.add_functions([Card(i,j,k,l) for i,j,k,l in cycles],'card')
```

Given this extended model, the solver automatically finds an assignment with optimal evaluation by the feature.

```
# set neg. weight -> minimize
model.set_feature_weight(-1,'card')
# create and run the solver
solution = ir.Optimizer(model).optimize()
```

◁

Due to the features, the dependencies between variables become more complex. Where we had a dependency between the variables of each edge in the basic graph coloring model, after the extension, all four variables of each 4-cycle depend on each other due to the functions `Card`. Infrared's solver automatically adapts to this increased complexity of the problem.

Boltzmann sampling.

Once specified by a model, a problem can be *solved* in different ways. In addition to finding optimal solutions, one can just as easily sample assignments from a uniform or Boltzmann distribution controlled by potentially multiple features and their weights.

Example. Continuing our example, we can generate random uniform colorings from the same model using a different solver.

In statistical mechanics, Boltzmann distributions describe the probabilities of states in a physical system depending on their energy. They are ubiquitous in physics and have numerous applications in bionformatics e.g. for describing the equilibrium of folding molecules [21] or generating energy weighted and near-optimal conformations [22]. Beyond physical interpretation, Boltzmann distributions have applications as general tools, e.g. in heuristic optimization [23], for deriving probabilities in alignments [24, 25] or for targeting properties [15].

```
# construct sampler from model
sampler = ir.Sampler(model)
# set weight 0 for uniform sampling
model.set_feature_weight(0,'card')
# generate a list of 100 samples
samples = [sampler.sample() for _ in range(100)]
```

Through the weight, we can control the expected value of the feature in the generated distribution. Setting a nonzero weight causes Infrared to sample from a nonuniform Boltzmann distribution, e.g. setting the weight to $+2$ shifts the expectation to a large cardinality while setting it to -2 induces smaller cardinalities. ◁

Positioning against prior work.

As already hinted by the introductory example, Infrared does **not** focus on general constraint solving as performed by constraint programming systems such as Gecode [26]. Adding evaluation to our models

ties this work closer to weighted constraint problems or cost networks, with some superficial relations to cost function optimizers such as Toulbar2 [27]. While such systems combine search with forms of constraint consistency, our solving strategies come from the area of constraint processing in constraint networks [12].

As such, our system is tailored to exactly and efficiently solve a specific class of problems, where it can algorithmically profit from a sufficiently tree-like structure (parameterized complexity for the parameter treewidth). This characteristic still allows broad and flexible use of the system, e.g. in bionformatics, where many relevant problems and problem instances have this structure. The capability to solve such bioinformatics problems by complete and exact algorithms with predictable complexity enables specific applications, e.g. it is essential for precisely controlled weighted sampling.

Overview and contributions.

In the next section, we formally define the core concepts of modeling problems in our framework; the models that characterize specific problem instances are formalized as feature networks. Moreover, we precisely state the tasks of optimization and sampling that are solved by the system. In Section 3, we describe the main algorithms to solve these tasks based on the model. Along with the algorithms we explain the underlying prerequisite key concepts of tree decomposition and cluster trees [12]. The given generic, cluster tree elimination-based, algorithms are efficient for fixed treewidth values of the feature network—in other words, they are exponential in the treewidth only. In the second part of the paper, we present several examples of modeling different classic bioinformatics problems as feature networks. Due to the declarative nature of the Infrared system, stating the feature networks is very close to actually implementing these algorithms. To increase the practical value, we put out documented Python code (in the form of Jupyter notebooks) for each application example as supplementary material. Starting with applications to showcase elementary use of Infrared, we move on to advanced topics, including interesting extensions to preceding examples and the targeting of features by multidimensional Boltzmann sampling. Finally, we discuss implications for the use of the system and its application range, as well as future developments within and beyond the framework.

2 Feature networks for modeling problems in Infrared

We conceptualize the declarative models of Infrared as *feature networks* (FNs). They are defined as a form of weighted CSP, explicitly distinguishing several real-valued features (instead of only a single or integer-valued score).

Definition 1 (Feature Network; FN)**.** A **feature network (FN)** $\mathcal{N}$ is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$, where

- $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of **variables**;
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of **domains**, one per variable, where each domain is a finite set of integers;
- $\mathcal{C}$ is a finite set of **constraints**;
- $\mathcal{F} = \{F_1, \ldots, F_\ell\}$ is a set of **features**.

Those networks specify *solutions* to a problem instance as specific assignments of domain values to variables.

Definition 2 (Assignment)**.** An **assignment**, for a given FN $\mathcal{N}$, is a set of single variable mappings $X_i \mapsto x_i$ such that X_i is a variable of $\mathcal{N}$, x_i is in the domain D_i of X_i and every X_i occurs at most once. An assignment is called **total**, if every variable of $\mathcal{N}$ occurs exactly once; otherwise, it is **partial**.

Given $\mathcal{X}' \subseteq \mathcal{X}$, we denote the set of all **assignments** x **of** $\mathcal{X}'$ as $\mathcal{A}_{\mathcal{X}'}$. When $\mathcal{X}'$ (and the order of its variables) is clear, one can write assignments as tuples, e.g. $(x_1, \ldots, x_n)$ in place of $X_1 \mapsto x_1, \ldots, X_n \mapsto x_n$ in the case of a total assignment ($\mathcal{X}' = \mathcal{X}$).

Example. Consider the graph $G = (V, E)$ of Figure 1. We model graph coloring for G and four colors as a feature network $\mathcal{N}_{\text{col}} = (\mathcal{X}_{\text{col}}, \mathcal{D}_{\text{col}}, \mathcal{C}_{\text{col}}, \mathcal{F}_{\text{col}})$. Let us first define $\mathcal{X}_{\text{col}} = \{X_1, \ldots, X_9\}$ and $\mathcal{D}_{\text{col}} = \{D_1, \ldots, D_9\}$; $D_i = \{1, 2, 3, 4\}$. This specifies one variable X_i for every vertex v_i in the graph and one domain per variable, encoding the colors as integer values.

A total assignment $x = \{X_1 \mapsto x_1, \ldots, \mathcal{X}_9 \mapsto x_9\} \in \mathcal{A}_{\mathcal{X}_{\text{col}}}$ describes a coloring where the vertex $v_i \in V$ has color x_i. ◁

Validity of assignments.

To distinguish valid from invalid assignments, we introduce constraints $\mathcal{C}$ that need to be satisfied by valid assignments. In our running example, this allows us to define valid colorings and thus completely specify graph coloring as CSP.

Definition 3 (Constraint)**.** Given $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$, each **constraint** $C \in \mathcal{C}$ is associated with a set of variables $X_{i_1}, \ldots, X_{i_k} \in \mathcal{X}$ and a Boolean function on values $x_{i_1}, \ldots, x_{i_k}$. Given an assignment x containing $X_{i_j} \mapsto x_{i_j}$ for all $j \in \{1, \ldots, k\}$, we **evaluate the constraint** C **w.r.t.** x by applying the Boolean function to $x_{i_1}, \ldots, x_{i_k}$. The resulting evaluation is denoted $C(x)$.

$$\begin{array}{lll} C: & D_{i_1} \times \cdots \times D_{i_k} & \to & \{\mathsf{True}, \mathsf{False}\} \\ & (x_{i_1}, \ldots, x_{i_k}) & \mapsto & C(x_{i_1}, \ldots, x_{i_k}) \end{array}$$

We say C is k**-ary** or has the **arity** k. Let $\text{vars}(C) = \{X_{i_1}, \ldots, X_{i_k}\}$ denote the **dependency** of C.

We call an assignment $x \in \mathcal{A}_{\mathcal{X}'}$, $\mathcal{X}' \subseteq \mathcal{X}$, **valid** iff all constraints $C \in \mathcal{C}$ with $\text{vars}(C) \subseteq \mathcal{X}'$ are **satisfied** (i.e. evaluated to True) by the assignment x.

Example. To enforce the neighbor coloring constraint in our example, it is sufficient to add the constraint below for each edge $(v_i, v_j) \in E$

$$\begin{aligned} \mathcal{C}_{\text{col}} &= \{\mathsf{NotEquals}_{[i,j]} \mid (v_i, v_j) \in E\} \\ &\text{with } \mathsf{NotEquals}_{[i,j]}(x_i, x_j) = \delta(x_i \neq x_j), \end{aligned}$$

where $\delta(p)$ is the truth value of the expression p.

The constraint $\mathsf{NotEquals}_{[i,j]}$ determines whether two given colors are distinct. Applying on all edges ensures that a valid assignment is a solution of the graph coloring problem. For example, both assignments $x_{\text{col}} = (1, 2, 3, 3, 4, 2, 1, 3, 1)$ and $x'_{\text{col}} = (1, 2, 4, 3, 4, 2, 1, 3, 1)$ satisfy the constraints $\mathcal{C}_{\text{col}}$.

In addition to finding valid assignments, one often wants to distinguish solutions by their quality. In graph coloring, we can e.g. aim for using fewer colors per 4-cycle; this would make x'_{col} preferable over x_{col}. ◁

Evaluation of assignments by features.

Each feature $F \in \mathcal{F}$ is a set of *network functions*. It specifies an *evaluation* as a sum over the values of the functions in this set; the single functions are defined in the same way as constraints but return real values (instead of Boolean ones).

Definition 4 (Network Function)**.** Each **network function** f of a feature network is associated with variables $X_{i_1}, \ldots, X_{i_k} \in \mathcal{X}$ and a real-valued function that, given an assignment x, maps the values $x_{i_1}, \ldots, x_{i_k}$ to a real number.

$$\begin{array}{lll} f: & D_{i_1} \times \cdots \times D_{i_k} & \to & \mathbb{R} \\ & (x_{i_1}, \ldots, x_{i_k}) & \mapsto & f(x_{i_1}, \ldots, x_{i_k}). \end{array}$$

Analogous to constraints, the returned value is called the **evaluation of** f **by** x, denoted $f(x)$, and the dependency is $\text{vars}(f) := \{X_{i_1}, \ldots, X_{i_k}\}$.

Overloading notation, we define the **(induced) feature evaluation** (of valid assignment x by feature F) by $F(x) = \sum_{f \in F} f(x)$. To account for multiple features, Infrared combines them linearly.

Definition 5 (Assignment evaluation)**.** Given a feature network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$ and **feature weights** α; α defines respective weights α_F for each feature F in $\mathcal{F}$. The evaluation of a valid assignment $x \in \mathcal{A}_\mathcal{X}$ is defined as a linear combination of the feature values w.r.t. α.

$$E_\mathcal{N}(x, \alpha) = \sum_{F \in \mathcal{F}} \alpha_F F(x).$$

Example. We can now express our objective in the extended graph coloring problem in terms of a feature. For this purpose, we introduce network functions that each count the different colors in a 4-cycle (v_i, v_j, v_k, v_l) of the example graph,

$$\mathsf{Card}_{[i,j,k,l]}(x_i, x_j, x_k, x_l) = |\{x_i, x_j, x_k, x_l\}|.$$

The corresponding feature set is then $\mathcal{F}_{\text{col}} = \{F_{\text{card}}\}$ with

$$F_{\text{card}} = \{\mathsf{Card}_{[2,3,5,6]}, \mathsf{Card}_{[2,5,7,8]}, \mathsf{Card}_{[5,6,7,8]}\}.$$

In feature network $\mathcal{N}_{\text{col}} = (\mathcal{X}_{\text{col}}, \mathcal{D}_{\text{col}}, \mathcal{C}_{\text{col}}, \mathcal{F}_{\text{col}})$, the two assignment examples x_{col} and x'_{col} are evaluated to $E_\mathcal{N}(x_{\text{col}}, 1) = 3 + 4 + 4 = 11$ and $E_\mathcal{N}(x'_{\text{col}}, 1) = 2 + 4 + 4 = 10$, respectively (for feature weight 1). ◁

Observe that a constraint satisfaction problem (CSP) is a special case of a feature network $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$, where $\mathcal{F}$ is empty. Feature networks are one of many possible extensions of CSPs known from the literature [12] that add forms of quality evaluation. For example, cost networks typically contain only a single set of functions, whereas we decided to explicitly distinguish a set of constraints from multiple sets of functions (features).

Infrared's modeling syntax

Recall the code snippets from the introduction. This code implements the feature network $\mathcal{N}_{\text{col}}$ that we formally developed above. As in our formal model description, the definition via Infrared's Python interface defines variables and domains, constraint and function types, and sets of constraints and functions. By providing the functionality to add constraints and functions to a model, we support compositional step-by-step construction and even extension of existing models.

Finally, our code examples demonstrate how models are fed to solvers, e.g. `ir.Optimizer` or `ir.Sampler`. This allows finding an optimal solution or generating controlled samples from the same model. We formally state the respective Problems 1 and 2 below; these solvers implement the algorithms of Sec. 3.

To keep this article concise, we refer the reader to our online reference and tutorials for syntactic aspects of using Infrared. For further reference, we recommend our coding-oriented introduction in Infrared in a book chapter [28], which focuses on modeling of sequence and RNA design problems. Moreover, recall that we maintain an online archive of documented Infrared application examples (covering all examples of this paper).

Problem statements

Given a feature network model $\mathcal{N}$, there are two tasks of immediate particular interest: optimization and sampling of the solution space. Our framework addresses both tasks explicitly and solves them automatically based on the specification of $\mathcal{N}$. First, we want to optimize assignments among all valid assignments of $\mathcal{N}$. Concretely, we ask for the assignment that optimizes the evaluation, i.e. the linear combination of the features given specific feature weights α.

Problem 1 (Assignment maximization)**.**

INPUT: Feature network $\mathcal{N}$, feature weights α

OUTPUT: Valid assignment $x^* \in \mathcal{A}_\mathcal{X}$ that is maximal w.r.t. $E_\mathcal{N}$ and α:

$$x^* = \arg\max_{\substack{x \in \mathcal{A}_\mathcal{X} \\ x \text{ is valid}}} E_\mathcal{N}(x, \alpha).$$

Furthermore, we want to use models to sample valid assignments from a Boltzmann distribution, i.e. each sample should be generated with a probability proportional to the Boltzmann weight of their evaluation w.r.t. a given α.

Problem 2 (Assignment sampling)**.**

INPUT: Feature network $\mathcal{N}$, feature weights α
OUTPUT: Valid assignment $x \in \mathcal{A}_\mathcal{X}$ generated with a probability that is proportional to its Boltzmann weight w.r.t. $E_\mathcal{N}$ and α:

$$\mathbb{P}(x) \propto \exp\left(E_\mathcal{N}(x, \alpha)\right). \tag{1}$$

Unfolding the assignment sampling problem, we realize that it implicitly asks for the **partition function**

$$Z_{\mathcal{N},\alpha} := \sum_{\substack{x \in \mathcal{A}_\mathcal{X} \\ x \text{ is valid}}} \exp\left(E_\mathcal{N}(x, \alpha)\right),$$

i.e. the proportionality factor in Eq. 1, such that

$$\mathbb{P}(x) = \exp\left(E_\mathcal{N}(x, \alpha)\right) / Z_{\mathcal{N},\alpha}.$$

3 Algorithms for solving feature networks

Given a feature network $\mathcal{N}$, Problem 1 asks for an optimal assignment to the variables. Here, the exponentially large assignment space forbids naive approaches. Based on a **tree decomposition** [29] of the network, we employ a form of dynamic programming that decomposes the computation into

- a 'forward' optimization phase to determine the optimal evaluation (i.e. only its numerical value), while storing the results of subproblems
- and a subsequent traceback algorithm to obtain an optimal assignment.

Our approach performs the optimization on a tree-like structure, namely, an annotated *tree decomposition* of the network, called the *cluster tree*. Instead of inefficiently searching through all total assignments, it enumerates value combinations of variable subsets at tree nodes and avoids redundant computation by storing the results of subproblems/subtrees; the evaluation of functions and constraints is interleaved with this enumeration. The optimization traverses the tree in bottom-up order; moving top-down in the same tree, based on the (intermediary) results of the first phase, the traceback algorithm identifies one optimal assignment. As such, the approach is a form of **cluster tree elimination (CTE)** [12].

3.1 Sampling resembles optimization

Assignment sampling (Problem 2) can be solved in a remarkably similar way to Problem 1. This task can also be split into two phases, namely, the computation of **partition functions** followed by stochastic traceback. Similar to standard traceback, stochastic traceback constructs solutions by tracing back through the partial results from the forward computation. However, it randomly selects values of variables based on partial partition functions. In this way it finally selects a total assignment from the intended distribution.

To emphasize the parallels between the problems, let us restate optimization as determining

$$E_{\max} = \max_{\substack{x \in \mathcal{A}_\mathcal{X} \\ x \text{ is valid}}} \sum_{F \in \mathcal{F}} \mathsf{Id}(\alpha_F F(x))$$

$$= \max_{\substack{x \in \mathcal{A}_\mathcal{X} \\ x \text{ is valid}}} \sum_{F \in \mathcal{F}} \sum_{f \in F} \mathsf{Id}(\alpha_F f(x))$$

where Id is the identity function, compared to the partition function

$$\begin{aligned} Z_{\mathcal{N},\alpha} &= \sum_{\substack{x \in \mathcal{A}_\mathcal{X} \\ x \text{ is valid}}} \prod_{F \in \mathcal{F}} \exp\left(\alpha_F F(x)\right) \\ &= \sum_{\substack{x \in \mathcal{A}_\mathcal{X} \\ x \text{ is valid}}} \prod_{F \in \mathcal{F}} \prod_{f \in F} \exp\left(\alpha_F f(x)\right). \end{aligned}$$

This breakdown into single network functions suggests that a general scheme can be applied to both problems, which specializes to either problem by the choice of algebra: $(\max, +, \mathsf{Id})$ for optimization and $(+, \times, \exp)$ for the partition function (and thus sampling).

3.2 Computation guided by cluster trees

We will define a cluster tree as an annotated tree decomposition of a feature network; it assigns the network functions and constraints to nodes (also called bags or clusters) where they should be evaluated. The computations process these clusters. Here, the tree decomposition determines the processing order. Processing the clusters bottom-up in the forward phase computes a result for the subtree of each cluster. For each cluster, this involves enumerating the assignments of cluster variables, while evaluating the constraints and functions of the cluster as well as previously computed results from the children clusters. The traceback follows the cluster tree top-down, partially re-evaluates the clusters and, on this basis, determines variables.

3.2.1 Dependency graphs, tree decompositions, and cluster trees

To properly guide the dynamic programming evaluation, cluster trees must reflect the network dependencies through tree decompositions.

Definition 6 (Dependency graph)**.** The **dependency graph of** $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$ is the hypergraph $G_\mathcal{N} = (\mathcal{X}, \mathcal{E}_\mathcal{N})$, where the hyperedges $\mathcal{E}_\mathcal{N}$ are the dependencies of the constraints and functions:

$$\mathcal{E}_\mathcal{N} := \Big\{ \operatorname{vars}(C) \mid C \in \mathcal{C} \Big\} \cup \Big\{ \operatorname{vars}(f) \mid f \in \bigcup_{F \in \mathcal{F}} F \Big\}.$$

Definition 7 (Tree decomposition, treewidth)**.** A **tree decomposition of a dependency graph** $G_\mathcal{N}$ is a pair (T, χ) of a (rooted) tree $T = (V, E)$ and a node labeling χ by subsets of variables, i.e. $\chi : V \to 2^\mathcal{X}$. These subsets are called the **bags** of the tree decomposition.

1. Each variable $X \in \mathcal{X}$ is in at least one bag;
2. For all hyperedge $e \in \mathcal{E}_\mathcal{N}$, there is a node $u \in V$, such that $e \subseteq \chi(u)$;
3. For all variables $X \in \mathcal{X}$, the set $\{u \in V \mid X \in \chi(u)\}$ induces a connected tree.

The **width** of a tree decomposition (T, χ) is

$$\max_{u \in V} |\chi(u)| - 1.$$

The **treewidth** of a hypergraph is the minimum width among all possible tree decompositions. The **tree decomposition and treewidth of a feature network** $\mathcal{N}$ are the respective tree decomposition and treewidth of its dependency graph $G_\mathcal{N}$.

For a tree decomposition (T, χ), $T = (V, E)$, consider two nodes $u, v \in V$, where v is the parent of u. Generally, we assume the tree edges to be oriented toward the root, such that $u \to v \in E$. We define two sets:

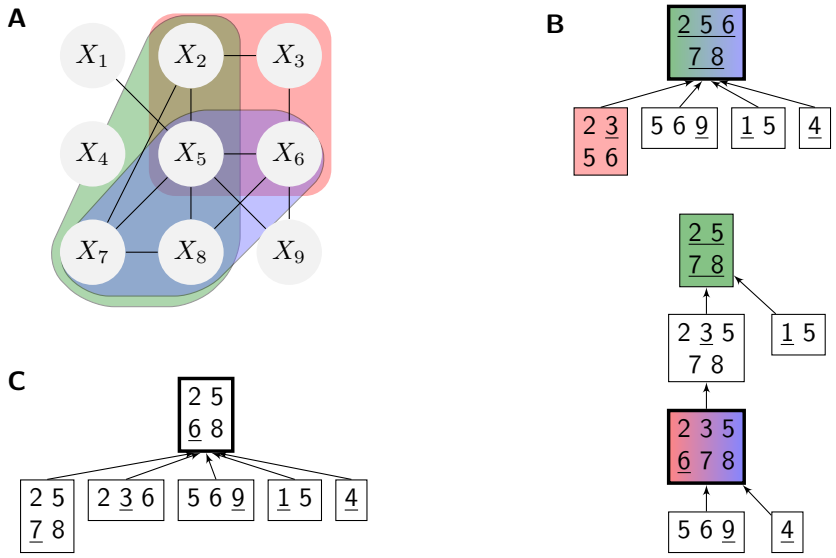

Fig. 2 Dependency graph and tree decompositions of the running example (feature network $\mathcal{N}_{\text{col}}$). **A)** The dependency graph contains one (binary) edge per dependency due to a constraint NotEquals $\in \mathcal{C}_{\text{col}}$. The dependency hyperedges due to the three network functions Card $\in F_{\text{card}}$ are colored. **B)** Two possible tree decompositions of this dependency graph (and therefore $\mathcal{N}_{\text{col}}$). The difference set is underlined in each bag. Solving of the network could be based on either one, but with different run time, which is dominated by the largest bag (bold). Due to their largest bags of size 5 and 6, the two tree decompositions have respective *width* 4 and 5. The bags handling the 4-ary functions are highlighted, where colors correspond to the hyperedges of A. **C)** Tree decomposition of the network without 4-ary functions Card. The functions don't allow any tree decomposition with width 3; thus they make the problem more complex.

- $\text{sep}(u) := \chi(u) \cap \chi(v)$ the **separator set** of shared variables between u and its parent; this set describes the variables whose values are passed between parent and child in a tree traversal;
- $\text{diff}(u) := \chi(u) \setminus \text{sep}(u)$ the **difference set** between the child and its parent. These are the 'introduced' variables by the child; they will be assigned at a child in the top-down traversal of the traceback.

To simplify the traceback step, we require tree decompositions to have empty root and difference sets of size 1 (i.e. $|\text{diff}(u)| = 1$ for every child node u); we call this property **gentle**. Gentle tree decompositions have exactly one edge per variable. Note that any tree decomposition (as defined above) can be efficiently turned into a gentle decomposition of the same width by inserting additional bags wherever the difference set is too large and contracting edges where no variables are introduced.

Definition 8 (Cluster Tree)**.** A **cluster tree** (T, χ, ϕ) **of an FN** $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$ is a tree decomposition $(T = (V, E), \chi)$ of $G_{\mathcal{N}}$ together with an annotation $\phi : V \to 2^{\mathcal{C}} \cup 2^{\mathcal{F}}$; it associates every $f \in \bigcup \mathcal{F}$ and $C \in \mathcal{C}$ to exactly one node $u \in V$ such that $\text{vars}(f)$ and $\text{vars}(C) \subseteq \chi(u)$.

We use notations $C \in \phi(u)$ and $f \in \phi(u)$ to refer to the constraints and network functions assigned to node u of the cluster tree, respectively. Given a cluster tree, we write α_f to refer to the weight α_F of the feature $F \in \mathcal{F}$ that contains f.

In addition to the general cluster tree definition, we assume that constraints and functions are assigned to the lowest possible bag (corresponding to the smallest possible subtree).

3.2.2 Evaluation following the cluster tree

For efficiency, Infrared evaluates constraints and network functions as soon as sufficient partial information is available (in contrast to a *generate-and-evaluate* strategy, which would enumerate assignments and evaluate only total assignments).

Recall the notion of *partial assignments* from Definition 2. The evaluation of a constraint (resp. network function) w.r.t. the partial assignment $\bar{x}$ is denoted $C(\bar{x})$ (resp. $f(\bar{x})$); it is defined if the assignment assigns all variables of the dependency $\text{vars}(C)$ (resp. $\text{vars}(f)$). Moreover, the union of

partial assignments is well-defined if their variable sets are disjoint; this allows for the composition of larger partial assignments from smaller ones.

Example. Consider the bag $\{X_2, X_5, X_6, X_7, X_8\}$ of Figure 2B (root of first tree) from the running graph-coloring example with cardinality network functions Card. The set $\bar{x}_{\text{col}} = \{X_2 \mapsto 2, X_5 \mapsto 4, X_6 \mapsto 2, X_7 \mapsto 1, X_8 \mapsto 3\}$ is a partial assignment of the bag's variables; it lets us evaluate $\mathsf{Card}_{[2,5,7,8]}$, $\mathsf{Card}_{[2,5,7,8]}(\bar{x}_{\text{col}}) = 4$, since the bag contains the dependency variables of this function. Consider another partial assignment $\bar{x}'_{\text{col}} = \{X_2 \mapsto 2, X_5 \mapsto 4, X_6 \mapsto 3, X_7 \mapsto 1, X_8 \mapsto 3\}$; $\bar{x}'_{\text{col}}$ is not valid because $\mathsf{NotEquals}_{[6,8]}$ evaluates to False w.r.t. $\bar{x}'_{\text{col}}$. ◁

Given a cluster tree (T, χ, ϕ) and a node u with parent v, the forward optimization algorithm successively computes optimal evaluations for subtrees T_u below nodes u (constituting subproblems of Problem 1).

The **optimal evaluation of subtree** T_u is

$$\max_{\substack{\bar{x} \in \mathcal{A}_{\chi(T_u)} \\ \bar{x} \text{ is valid}}} \sum_{f \in \phi(T_u)} \alpha_f f(\bar{x}) \tag{2}$$

where $\chi(T_u) := \bigcup_{c \in T_u} \chi(c)$ and $\phi(T_u) := \bigcup_{c \in T_u} \phi(c)$.

To obtain total subtree evaluations, the algorithm computes and stores *conditional* optimal subtree evaluations, which depend on the partial assignment to $\text{sep}(u)$. Thus, they allow decoupling the subtree from the remaining tree.

For a node u, these conditional evaluations are computed for all such valid partial assignments, such that they specify network functions $m_{u \to v}$. We call them **messages** since they are used to pass information from child u to parent v.

Additionally, define $\text{diff}(T_u) := \chi(T_u) \setminus \text{sep}(u)$ as the set of **variables introduced by** T_u.

Definition 9 (Conditional optimal subtree evaluation)**.** Let u be a node in the cluster tree (T, χ, ϕ); denote its parent by v. The **conditional optimal subtree evaluation** at u under condition $\bar{x} \in \mathcal{A}_{\text{sep}(u)}$ is

$$m_{u \to v}(\bar{x}) = \max_{\substack{\bar{x}' \in \mathcal{A}_{\text{diff}(T_u)} \\ \bar{x} \cup \bar{x}' \text{ is valid}}} \sum_{f \in \phi(T_u)} \alpha_f f(\bar{x} \cup \bar{x}') \tag{3}$$

Since the root of T is empty, conditional optimal subtree evaluations allow us to directly infer the total evaluation at the root. For every child u of the root, $\text{sep}(u)$ is empty; moreover, the children of the root do not have any variables in common (due to the definition of tree decomposition). Consequently, we obtain the total evaluation by summing the 0-ary messages sent to the root from its children

$$E_{\max} = \sum_{\text{child } u \text{ of root}} m_{u \to \text{ root}}(\varnothing).$$

See Figure 3 for an illustration of the bottom-up computation and the subsequent top-down traceback. Following Proposition 1 each bag u can be processed together with the messages sent to it from its children; thus, we can understand the full computation as bottom-up message passing (Algorithm 1). The notion of message passing stems from more general formulations of CTE on unrooted cluster trees [30]. The algorithm is correct due to the following proposition (shown in Additional file 1).

Proposition 1. Let $u \to v$ be a cluster tree edge and $\bar{x} \in \mathcal{A}_{\text{sep}(u)}$ be a partial assignment of $\text{sep}(u)$. The conditional optimal subtree evaluations $m_{u \to v}(\bar{x})$ (Eq. 3) can be recursively computed as

$$m_{u \to v}(\bar{x}) = \max_{\substack{\bar{x}' \in \mathcal{A}_{\text{diff}(u)} \\ \bar{x} \cup \bar{x}' \text{ is valid}}} \begin{bmatrix} \sum_{f \in \phi(u)} \alpha_f f(\bar{x} \cup \bar{x}') \\ + \sum_{c \text{ child of } u} m_{c \to u}(\bar{x} \cup \bar{x}') \end{bmatrix}. \tag{4}$$

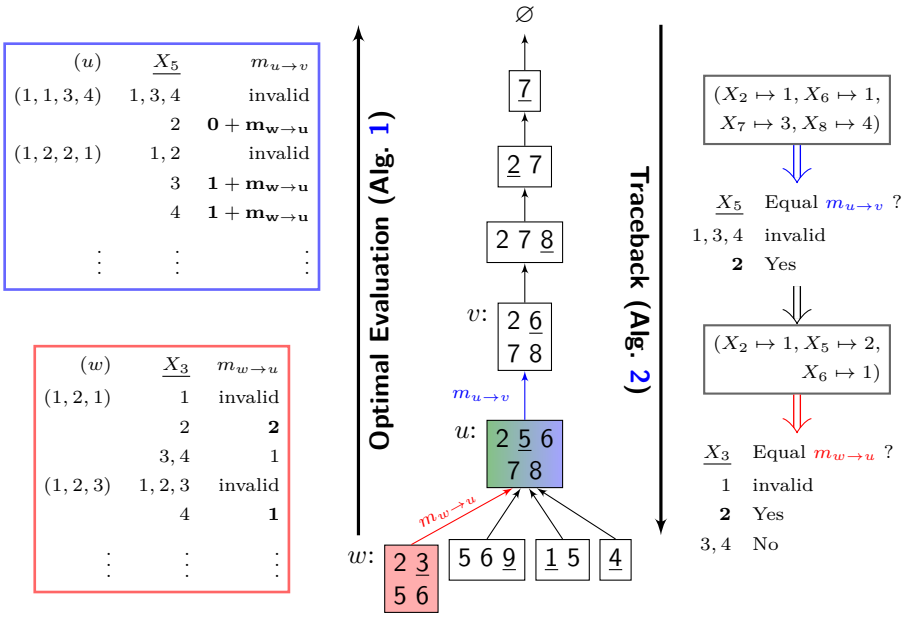

Fig. 3 Illustration of the forward optimal evaluation and traceback algorithms (by the running example of graph coloring; Figure 1). We elaborate steps of the computation guided by the gentle tree decomposition corresponding to Figure 2B (top). The indices of variables in the difference set are underlined. On the left, we sketch the computation of the messages $m_{w\to u}$ and $m_{u\to v}$: For every assignment of the separator set, the algorithm maximizes over assignments of the difference variable (it dismisses invalid assignments); in the computation of $m_{u\to v}$, it used the already computed message $m_{w\to u}$. On the right, we show the corresponding computations to assign values to the *underlined* variables during traceback: given an optimal assignment to the variables in v, we first infer that $X_5 = 2$ is an optimal continuation, and finally $X_3 = 2$.

Algorithms for partition functions and sampling

As argued, the computation of partition functions (Problem 2) follows the same algorithmic structure, changing the algebra in Algorithm 1 from $(\max, +, \mathsf{Id})$ to $(+, \times, \exp)$ and setting the initial value of t to 0. Consequently, the partition function $Z_{\mathcal{N},\alpha}$ is obtained by multiplying all 0-ary messages sent to the root.

Analogous to partial optimal evaluations, the modified Algorithm 1 with $(+, \times, \exp)$-algebra computes partial partition functions.

Definition 10 (Conditional subtree partition functions)**.** Let u be a node in a cluster tree (T, ξ, ϕ), where v is its parent. The **conditional partition function** at u under condition $\bar{x} \in \mathcal{A}_{\mathrm{sep}(u)}$ is

$$m_{u\to v}(\bar{x}) = \sum_{\substack{\bar{x}' \in \mathcal{A}_{\mathrm{diff}(T_u)} \\ \bar{x}\cup\bar{x}' \text{ is valid}}} \prod_{f\in\phi(T_u)} \exp(\alpha_f f(\bar{x}\cup\bar{x}')) \tag{5}$$

for all $\bar{x} \in \mathcal{A}_{\mathrm{sep}(u)}$.

Partition functions are computed by a recursive algorithm analogous to 1; its correctness is stated in analogy to Proposition 1 (shown in Additional file 1).

Proposition 2. Let $u \to v$ be a cluster tree edge and $\bar{x} \in \mathcal{A}_{\mathrm{sep}(u)}$ be a partial assignment of $\mathrm{sep}(u)$. The conditional subtree partition functions (Eq. 5) can be recursively computed as

$$m_{u\to v}(\bar{x}) = \sum_{\substack{\bar{x}' \in \mathcal{A}_{\mathrm{diff}(u)} \\ \bar{x}\cup\bar{x}' \text{ is valid}}} \left[\begin{array}{l} \prod_{f\in\phi(u)} \exp(\alpha_f f(\bar{x}\cup\bar{x}')) \\ \times \prod_{c \text{ child of } u} m_{c\to u}(\bar{x}\cup\bar{x}') \end{array}\right] \tag{6}$$

with α_f as in Prop. 1.

3.3 Traceback

Once all messages of partial optimal score are computed by Algorithm 1, the optimal assignment is obtained by a traceback traversing the cluster tree top-down in preorder (Algorithm 2). At each edge $u \to v$, an optimal assignment of the variables in the parent v is known. Infrared then determines the optimal assignment to the difference variables (in the singleton set $\mathrm{diff}(u)$) such that the evaluation of bag u equals the message sent to the parent bag v. Let x be the partial optimal assignment determined thus far in the algorithm (assigning the variables of v); the algorithm searches through $\bar{x} \in \mathcal{A}_{\mathrm{diff}(u)}$ and selects one assignment $\bar{x}$ that yields the optimal message. This choice is optimal, such that the algorithm can continue its top-down traversal after updating x by $x \cup \bar{x}$.

For sampling, Infrared performs a stochastic traceback (Algorithm 3), requiring the messages from the computation of the partition function. Whereas the general structure resembles that of optimal traceback, at each edge $u \to v$ the algorithm randomly chooses a tracking value t uniformly from the range 0 and $m_{u\to v}$. While iterating through the possible assignments for $\mathrm{diff}(u)$, t is gradually decreased with partial Boltzmann factors. The value of the difference variable is selected once t becomes negative. This selects the value following the desired Boltzmann distribution. We show the following correctness claim in Additional file 1.

Proposition 3. Algorithm 3 solves Problem 2 by sampling assignments from the Boltzmann distribution of Eq. (1).

Algorithm 1: Optimal evaluation

Input : Cluster tree (T, χ, ϕ), $T = (V, E)$, weights α
Output: Messages $m_{u\to v}$ for all $u \to v \in E$
forall *edge* $u \to v \in E$ *in postorder* **do**
  forall *partial assignment* $\bar{x} \in \mathcal{A}_{\mathrm{sep}(u)}$ **do**
    $t \leftarrow -\infty$;
    forall *partial assignment* $\bar{x}' \in \mathcal{A}_{\mathrm{diff}(u)}$ **do**
      if $\bar{x} \cup \bar{x}'$ *is valid for all constraints* $C \in \phi(u)$ **then**
        $p \leftarrow \mathtt{sum}\{\mathsf{ld}(\alpha_f f(\bar{x} \cup \bar{x}'))$
          $\mid$ function $f \in \phi(u)\}$
          $+\mathtt{sum}\ \{m_{c\to u}(\bar{x} \cup \bar{x}') \mid$ child c of $u\}$;
        $t \leftarrow \max(t, p)$;
    $m_{u\to v}(\bar{x}) \leftarrow t$;

3.4 Computational complexity

Note that while computational complexities can be interpreted as corollaries from CTE [12], we rephrase the arguments adapted to our concrete algorithms.

For a feature network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$, we state complexities in terms of additional parameters: the **largest domain size** $d := \max_{D\in\mathcal{D}} |D|$, the **number of variables** $\mathfrak{n} := |\mathcal{X}|$, and the total **number of constraints and network functions** $\mathfrak{m} := |\mathcal{C}| + \sum_{F\in\mathcal{F}} |F|$. Let w denote the **treewidth** of dependency graph $G_{\mathcal{N}}$. Furthermore, we assume that single constraints and network functions are evaluated in constant time. We will see later from the applications in Section 4 and Section 5 the assumption holds in practice.

Proposition 4. Algorithm 1 takes $\mathcal{O}\left(d^{w+1} \cdot (\mathfrak{m} + \mathfrak{n})\right)$ time and $\mathcal{O}\left(d^w \cdot \mathfrak{n}\right)$ space.

Proof. Algorithm 1 computes one message $m_{u\to v}$ for each edge $u \to v$ of the tree decomposition of $\mathcal{N}$. In every child bag u, the algorithm computes message values for each assignment of the variables in $\mathrm{sep}(u)$, for each value optimizing over assignments of $\mathrm{diff}(u)$. In every iteration, it evaluates the constraints and functions in $\phi(u)$, as well as the messages from the children.

Algorithm 2: Optimal traceback

Input : Cluster tree $(T = (V, E), \chi, \phi)$, weights α
messages $m_{u\to v}$ for all $u \to v \in E$
as computed by Algorithm 1
Output: Optimal assignment x with score $E_{\max}$

$x \leftarrow \varnothing$;
forall *edge* $u \to v \in E$ *in preorder* **do**
  $t \leftarrow m_{u\to v}(x)$;
  forall *partial assignment* $\bar{x} \in \mathcal{A}_{\mathrm{diff}(u)}$ **do**
    if $x \cup \bar{x}$ *is valid for all constraints* $C \in \phi(u)$ **then**
      $p \leftarrow \mathtt{sum}\{\mathsf{ld}(\alpha_f f(x \cup \bar{x}))$
       $\mid$ function $f \in \phi(u)\}$
       $+\mathtt{sum}\ \{m_{c\to u}(x \cup \bar{x}) \mid \text{child } c \text{ of } u\}$;
      if $p = t$ **then**
        $x \leftarrow x \cup \bar{x}$;
        break;

return x

Algorithm 3: Stochastic traceback: sampling

Input : Cluster tree $(T = (V, E), \chi, \phi)$, weights α,
messages $m_{u\to v}$ for all $u \to v \in E$ due to
modified Algorithm 1 for partition functions
Output: Assignment x sampled from Boltzmann weighted distribution

$x \leftarrow \varnothing$;
forall *edge* $u \to v \in E$ *in preorder* **do**
  $t \leftarrow$ uniform random number between 0 and $m_{u\to v}(x)$;
  forall *partial assignment* $\bar{x} \in \mathcal{A}_{\mathrm{diff}(u)}$ **do**
    if $x \cup \bar{x}$ *is valid for all constraints* $C \in \phi(u)$ **then**
      $p \leftarrow \mathtt{product}\{\exp(\alpha_f f(x \cup \bar{x}))$
       $\mid$ function $f \in \phi(u)\}$
       $\times\mathtt{product}\ \{m_{c\to u}(x \cup \bar{x}) \mid \text{child } c \text{ of } u\}$;
      $t \leftarrow t - p$;
      if $t < 0$ **then**
        $x \leftarrow x \cup \bar{x}$;
        break;

return x

We thus bound the computation by

$$\sum_{u\to v} \prod_{x_i \in \mathrm{sep}(u)} |\mathcal{D}_i| \cdot \prod_{x_i \in \mathrm{diff}(u)} |\mathcal{D}_i| \cdot (|\phi(u)| + n_u),$$

where n_u counts the children of u. Since $\mathrm{sep}(u)$ and $\mathrm{diff}(u)$ are disjoint and contain exactly the variables of in the bag u, there are at most d^{w+1} iterations per bag. We relax the bound to

$$d^{w+1} \cdot \sum_{u\to v} (|\phi(u)| + n_u)$$

Every constraint and function is evaluated in the iterations of exactly one bag u; thus, we can amortize the contributions due to $\sum_{u\to v} |\phi(u)| = \mathfrak{m}$. Moreover, every message from a child is accessed (in

constant time) in the iterations of exactly one bag; we can thus amortize due to $\sum_{u \to v} n_u = |X|$. This lets us simplify the last bound further to derive the claim on the time complexity.

Concerning space, the algorithm stores a message at each edge of the tree decomposition. Per edge $u \to v$, this takes space $\mathcal{O}(d^{\mathrm{sep}(u)})$. This bounds the space by $\mathcal{O}(d^s \cdot |E|)$, where $s = \max_u \mathrm{sep}(u)$. For gentle tree decompositions, $|E| = |X|$ and $s \leq w$, showing the claim. □

Proposition 5. Algo. 2 runs in $\mathcal{O}\left(d \cdot (\mathfrak{m} + \mathfrak{n})\right)$ time.

Proof. For each edge $u \to v$ of the tree decomposition of $\mathcal{N}$, the task is to determine the best assignment for variables in $\mathrm{diff}(u)$, given that variables in $\mathrm{sep}(u)$ are already assigned (as guaranteed by the iteration in preorder). Deciding if an assignment is valid requires computing constraints, while scoring them requires computing network functions (each in constant time due to our assumption). It is also required to sum up n_u messages $m_{c \to u}$ for c children of u, where n_u denotes the number of children. Given that $|\,\mathrm{diff}(u)| = 1$ in a gentle tree decomposition, we obtain as an upper bound of time complexity in Algorithm 2:

$$\begin{aligned}
&\sum_{u \to v} \prod_{x_i \in \mathrm{diff}(u)} |\mathcal{D}_i| \cdot (|\phi(u)| + n_u) \\
&\leq d \sum_{u \to v} (|\phi(u)| + n_u) \leq d(\mathfrak{m} + \mathfrak{n}).
\end{aligned}$$

□

Note that the complexity results for optimization and optimal traceback directly apply to partition function computation and stochastic traceback, which evaluate exactly the same numbers of constraints, functions and messages.

3.5 Complexity analysis for nonuniform domain sizes

For nonuniform domain sizes, the previous analysis can strongly overestimate the complexity (assuming the worst-case maximum domain size d for all variables). In several of our application examples, we can tighten the analysis considering that $\mathcal{X}$ is composed of two (analogously extensible to several) 'series' of variables in the way

$$\mathcal{X} = \{X_1, \ldots, X_{n_X}, Y_1, \ldots, Y_{n_Y}\}$$

with respective maximum domain sizes d_X and d_Y. For a given tree decomposition, we can define subset widths w_X and w_Y as the maximum number of respective X and Y variables in a bag minus 1.

Then, we bound more tightly as follows:

$$\begin{aligned}
&\sum_{u \to v} \prod_{x_i \in \mathrm{sep}(u)} |\mathcal{D}_i| \cdot \prod_{x_i \in \mathrm{diff}(u)} |\mathcal{D}_i| \cdot (|\phi(u)| + n_u) \\
&\leq d_X^{w_X+1} \cdot d_Y^{w_Y+1} \cdot \sum_{u \to v} (|\phi(u)| + n_u) \\
&\leq d_X^{w_X+1} \cdot d_Y^{w_Y+1} \cdot (\mathfrak{m} + \mathfrak{n}).
\end{aligned}$$

Corollary 1. The runtime of Algo. 1 is in

$$\mathcal{O}\left(d_1^{w_1+1} \cdots d_k^{w_k+1} \cdot (\mathfrak{m} + \mathfrak{n})\right),$$

given a feature network where $\mathcal{X}$ is a disjoint union of subsets $\mathcal{X}_1, \ldots, \mathcal{X}_k$ and a tree decomposition (T, χ), where $w_i = \max_{v \in T} |\{X \in \mathcal{X}_i \mid X \in \chi(v)\}|$ are the respective subset widths of $\mathcal{X}_1, \ldots, \mathcal{X}_k$ w.r.t. (T, χ).

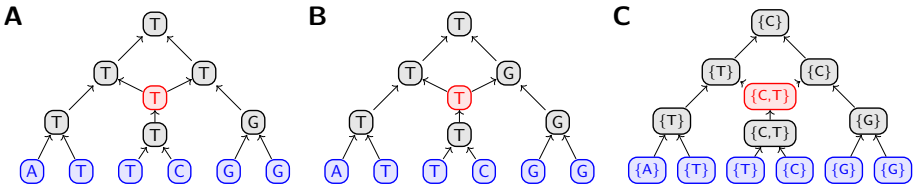

Fig. 4 Example phylogenetic network. Optimal solutions for A) hardwired parsimony. B) softwired parsimony. C) parental parsimony; nodes are labeled by character sets. The input for these problems consists of the network and the labels of only the leaves (blue).The other labels are inferred. The example contains one reticulation node (red).

It becomes apparent that tree decompositions with minimal width do not necessarily yield best performance in this context (e.g. [31]). We can take a shortcut in special cases, where variables X_i and Y_i for the same $1 \leq i \leq n$ ($n = n_X = n_Y$) depend on each other. Then, one can collapse the nodes of X_i and Y_i in the dependency graph, generate a standard tree decomposition optimizing its width w', and infer a tree decomposition of the original dependency graph with $w_X = w_Y = w'$.

3.6 Parameterized complexity classes

Based on our complexity results (Sec. 3.4), the Optimization (Prob. 1) and Sampling (Prob. 2) can be solved efficiently in the *input size* n at fixed values of the treewidth. Assuming that the number of variables and number of edges is on the order of n, i.e. $\mathfrak{n} + \mathfrak{m} \in \mathcal{O}(n)$, the input-dependence of the maximum domain size d, $d \in \mathcal{O}(1)$ or $d \in \mathcal{O}(n)$, determines the theoretical parameterized complexity class.

For problems parameterized by k, one distinguishes the class **FPT** (**fixed parameter tractable**), where problems can be solved in time $f(k)n^{\mathcal{O}(1)}$ for some computable function f, from the class **XP** with a bound of $f(k)n^{g(k)}$ [9, 14] for some computable functions f, g. The latter class strictly includes the former. **XP** algorithms are also called **slicewise polynomial**, having polynomial complexity for each fixed value of the parameter.

For constant d, it follows that solving in Infrared is in the class **FPT** parameterized by the treewidth of the dependency graph. This is the case for the presented applications to RNA design, where the domain size is the number of nucleotides, i.e. typically 4. In our applications to pseudoknotted RNA alignment, the domain size d is in $\mathcal{O}(n)$; consequently, we obtain an **XP** solving algorithm.

3.7 Computing tree decompositions

The problem of computing a tree decomposition of minimal treewidth for an input graph/network is NP-hard [32]. However, multiple heuristics [13] and even efficient exact solvers [33] have been designed, motivated by the wide applicability of treewidth-based methods.

From a theoretical perspective, treewidth is FPT to compute, albeit with a prohibitive complexity of $2^{O(w^3)}$ [34]. A $4 \cdot w + 4$ approximation in $O(8^w \cdot w^2 \cdot |\mathcal{X}|^2)$ is also possible [14]. Both of these results guarantee that FPT results remain FPT when including the computation of a tree decomposition prior to applying Algorithm 1 and 2. However, the actual complexity may be affected, becoming the worst of the two.

Despite these theoretical results, virtually all treewidth-based implementations, including Infrared, use the beforementioned heuristics or solvers to compute tree decompositions.

4 Applications to concrete bionformatics problems

4.1 Network parsimony

Parsimony for phylogenetic reconstruction.

For inferring phylogenies, one of the central missions of bioinformatics, parsimony methods determine the most parsimonious explanations for evolutionary relationships. In the classical small parsimony problem the relation between n taxa is given as their phylogenetic tree. The leaves are labeled by

'characters', i.e. the taxa, and we ask for a labeling of the internal nodes such that the number of label differences over all tree edges is minimized. However, due to *reticulate evolution*, where lineages can be influenced by two or more ancestors, many real phylogenies are better represented by phylogenetic networks than trees [35]. This model captures diverse phenomena such as hybrid speciation, horizontal gene transfer, and allopolyploidity due to sexual recombination. While tree parsimony has well-established polynomial-time solutions [36, 37], network parsimony is a topic of current algorithmic research. For example, Scornavacca and Weller [5] present artfully hand-crafted fixed-parameter tractable (FPT) algorithms for three variants of network parsimony. We will discuss modeling network parsimony directly in Infrared and, in this way, immediately obtain FPT solutions.

Definition 11 (Phylogenetic Network)**.** A **phylogenetic network** is a rooted, connected directed acyclic graph $G = (V, E)$. Edges point from **children** to their **parents**. The unique **root** $r \in V$ is the only node without parents; the **leaves** $L \subseteq V$ are the nodes without children. **Reticulation nodes** have more than one parent.

Hardwired parsimony can be seen as a direct extension from tree to network parsimony that minimizes a parsimony score summing over all network edges, **softwired parsimony** inherits—in the case of multiple parents—only from the most favorable one, and **parental parsimony** allows embedding of different lineages in the network (one parent per allele) to cover allopolyploidy. [5]

In this text, we describe in detail the modelings of hardwired and softwired parsimony. For Infrared models of all three variants of network parsimony, we refer to our online documentation.

Problem 3 (Hardwired network parsimony)**.**

INPUT: Phylogenetic network $G = (V, E)$ with leaves L, set of characters Σ, and leave labeling $\phi : L \to \Sigma$.
OUTPUT: Minimal parsimony score $\mathrm{PS}^*_{\mathrm{hw}}$ and corresponding labeling $\psi : V \to \Sigma$, where

$$\mathrm{PS}^*_{\mathrm{hw}} = \min_{\substack{\text{labeling } \psi \\ \forall v \in L : \psi(v) = \phi(v)}} \sum_{(u,v) \in E} \mathrm{d}(\psi(u), \psi(v)),$$

here limiting ourselves, for simplicity, to distance $\mathrm{d}(x, y) = \begin{cases} 1 & x \neq y \\ 0 & x = y. \end{cases}$

Infrared network model.

We model labellings as assignments, i.e. we use one variable X_i per node of G, whose value encodes its label, i.e. the domain of internal nodes is Σ, while the domain of leave variables is restricted by the input labeling ϕ. We can thus specify the variables and domains of the feature network $\mathcal{N}_{\mathrm{hw}} = (\mathcal{X}_{\mathrm{hw}}, \mathcal{D}_{\mathrm{hw}}, \mathcal{C}_{\mathrm{hw}}, \mathcal{F}_{\mathrm{hw}})$, which models Problem 3:

- $\mathcal{X}_{\mathrm{hw}} = \{X_1, \ldots, X_{|V|}\}$
- $\mathcal{D}_{\mathrm{hw}} = \{D_1, \ldots, D_{|V|}\}$,
  where $D_i = \begin{cases} \{\phi(i)\} & v_i \in L \\ \Sigma & \text{otherwise} \end{cases}$

On this basis, we impose constraints and functions. In this case, there are no constraints (all constraints are expressed by restricting the domains, such that all assignments are valid labellings). To express the score (by a set of network functions), we introduce the network function $\mathsf{Distance}_{[i,j]}$ for the variables X_i and X_j is defined to encode the distance $d(x_i, x_j)$ between their values in an assignment x. We finalize the model by

- $\mathcal{C}_{\mathrm{hw}} = \{\}$
- $\mathcal{F}_{\mathrm{hw}} = \{F_{\mathrm{hwd}}\}$ with feature

$$F_{\mathrm{hwd}} = \{\mathsf{Distance}_{[X_i, X_j]} \mid (v_i, v_j) \in E\}.$$

To implement and solve the problem in Infrared, it suffices to *translate* the model to Infrared syntax and call its optimizer. According to Proposition 4, the framework determines a most parsimonious solution in time complexity $\mathcal{O}((|E| + |V|) \cdot |\Sigma|^{w+1})$ in the treewidth w of the input network $G = (V, E)$. For this corollary observe that the dependency graph of the modeled feature network is exactly the input network; moreover the model has $|E|$ functions and $|V|$ variables with maximum domain size $d = |\Sigma|$; functions are computed in constant time.

Beyond hardwired network parsimony.

The problem of **softwired network parsimony** redefines the score of hardwired parsimony, such that it asks for

$$\mathrm{PS}^*_{\mathrm{sw}} = \min_{\substack{\text{labeling } \psi \\ \forall v \in L : \psi(v) = \phi(v)}} \sum_{u \in V} \min_{v \in \mathrm{parents}(u)} \mathrm{d}(\psi(u), \psi(v)),$$

where $\mathrm{parents}(u)$ denotes the set of parents of u. This does not change the behavior at nonreticulation nodes, but offers a choice in the case of reticulation nodes.

Here, we restrict our model to **binary networks**, where nodes can have up to two children and up to two parents. Then, starting from the hardwired model, we enable this choice by adding a Boolean selector variable Y_i for every reticulation node v_i. The distance to the left parent is counted if $Y_i = 0$; to the right parent, if $Y_i = 1$. Then, we replace the distance network functions by special variants at all edges between a reticulate child u and one of its parents v; feature F_{hwd} is substituted by

$$\begin{aligned} F_{swd} = \{&\mathsf{RDistance}_{[X_i, X_j, Y_i; r]} \mid i, j \in (v_i, v_j) \in E, \\ &v_i \text{ is a reticulation node}, \\ &r = 1 \text{ if } v_j \text{ is right parent of } v_i \text{ else } 0\} \\ \cup \{&\mathsf{Distance}_{[X_i, X_j]} \mid i, j \in (v_i, v_j) \in E, \\ &v_i \text{ is not a reticulation node}\}, \end{aligned}$$

where r controls the selection, i.e.

$$\begin{aligned} &\mathsf{RDistance}_{[X_i, X_j, Y_i; r]}(x_i, x_j, y_i) \\ &\quad = \begin{cases} \mathsf{Distance}_{[X_i, X_j]}(x_i, x_j) & \text{if } y_i = r \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

To quickly bound the complexity (but improving over applying Proposition 4 directly), we follow Section3.5. For the purpose of a conservative worst case complexity analysis, let us even add Y_i variables for the nonreticulation nodes. Consequently, we derive a tree decomposition by decomposing the input network(!) with treewidth w, which allows us to infer a tree decomposition of our feature network with $w_X = w_Y = w$ (since the input network is equivalent to the dependency graph after collapsing the nodes of X_i and Y_i for each i). On that basis, by Cor. 1, we obtain $\mathcal{O}(|\Sigma|^{w+1} 2^w (|V| + |E|))$ for solving softwired network parsimony.

Discussion.

Scornavacca and Weller [5] present algorithms for hardwired, softwired, and parental network parsimony with respective complexities of $\mathcal{O}(|\Sigma|^{w+1}|E|)$, $\mathcal{O}(|\Sigma|^w(3^w|\Sigma||V| + |E|)$, and $\mathcal{O}(6^{w|\Sigma|}4^{w\log(c)}|E|)$ (after obtaining the tree decomposition). In the hardwired case, we obtain the exact same complexity out-of-the-box.

In the case of *softwired complexity* for the special case of binary networks, we even obtain a complexity with a better treewidth dependence. To show this, given $|E| < 2|V|$ under the assumption of binary networks, one simplifies our result to $\mathcal{O}(|\Sigma|^{w+1}2^w|V|)$ and theirs to $\mathcal{O}(|\Sigma|^{w+1}3^w|V| + |\Sigma|^w 2|V|) = \mathcal{O}(|\Sigma|^{w+1}3^w|V|)$.

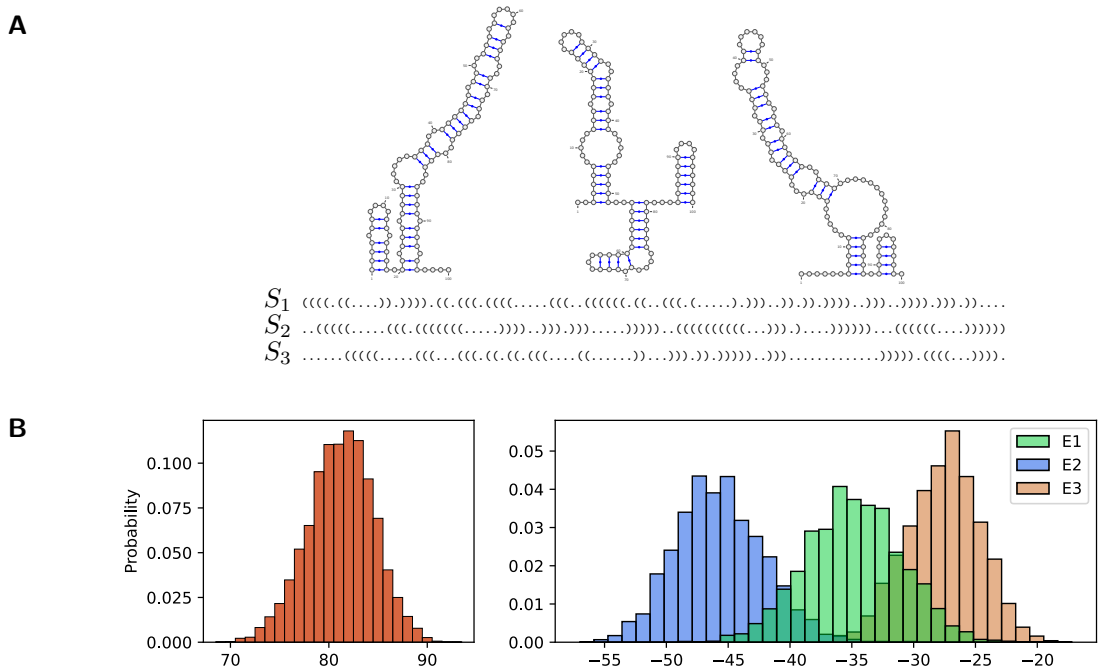

Fig. 5 RNA multitarget design. **A)** Three target RNA secondary structures of length 100 as 2D plots (by VARNA [38]) and dot-bracket strings; taken from a multitarget design benchmark set [39]. **B)** Histograms of the features GC content (left), and the Turner energies (kcal/mol) of the three targets (right) in 5000 sequences sampled from the multitarget design model $\mathcal{N}_{\text{design}}$ at weight -5 for every feature. One can observe that 1) equal weights lead to different mean energies for the targets; 2) strong control of the GC weight is required to avoid extreme GC content for stable designs. To automate the calibration of weights (and target specific feature value combinations), we suggest multidimensional Boltzmann sampling in Section 5.4.

We refer to our accompanying notebook for the case of *parental parsimony*. There, we provide a model that induces an efficient solution whenever the treewidth of the feature network remains bounded. In this case, the feature network simply consists of the input network, augmented by ternary constraints at reticulation nodes. Consequently, while obtaining an FPT algorithm even for this complex parsimony problem, we cannot directly compare its complexity to [5].

4.2 RNA design

Designing biomolecules for specific biotechnological or medical applications is typically an interdisciplinary endeavor combining experimentation and computational design. On the computational side this calls for flexible, extensible systems that can express and efficiently cope with various constraints and objectives—a paradigmatic playing field for our framework (see our treatment in [28]). A challenging, computationally hard subproblem in this area is the design of RNA sequences that fold into multiple target structures. The Infrared framework generalizes the FPT algorithm of our earlier work RNARedprint [16]—a method to generate RNA sequences w, words over $\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}$, with very specific properties towards multiple structures and specific GC content (i.e. the amount of G and C characters; denoted $\#\mathtt{GC}(w)$). Here, we model the core problem of RNARedprint directly as feature network, which allows its implementation in Infrared.

Definition 12 (RNA secondary structure)**.** A **secondary structure** S of length n is a set of **base pairs**, i.e. pairs (i, j) of sequence positions, $1 \leq i < j \leq n$. Secondary structures are required to be **free of base triplets**, i.e. every base $1 \leq i \leq n$ is involved in at most one base pair. A secondary structure S is called **crossing** iff there are pairs $(i, j), (k, l) \in S$, such that $i < k < j < l$; otherwise, it is **noncrossing**.

Multitarget design sampling.

Given one or multiple noncrossing RNA secondary structures as *targets* (Fig. 5A), we consider the problem of controlled sampling of designs (i.e. RNA sequences) from a Boltzmann distribution governed by the thermodynamic energies of the targets and the GC content, whose respective influence is controlled by weights (Fig. 5B).

Problem 4 (Multitarget RNA sequence sampling)**.** Given are k target structures, i.e. noncrossing secondary structures $S_1, \ldots, S_k$ of length n, together with weights $\alpha_1, \ldots, \alpha_k$ and $\alpha_{\mathtt{GC}}$. We ask for r RNA sequences of length n such that for each sequence s

$$\mathbb{P}(s) \propto \exp\left(\alpha_{\mathtt{GC}} \cdot \#\mathtt{GC}(s)\right) \cdot \prod_{\ell=1}^{k} \exp\left(\alpha_\ell \cdot E(s, S_\ell)\right).$$

Constraints and functions.

In common energy models of RNAs, such as the nearest neighbor model [40], all base pairs must be **canonical**, i.e. in

$$\mathcal{B} = \{(\mathtt{A},\mathtt{U}), (\mathtt{C},\mathtt{G}), (\mathtt{G},\mathtt{C}), (\mathtt{G},\mathtt{U}), (\mathtt{U},\mathtt{A}), (\mathtt{U},\mathtt{G})\}.$$

Otherwise, the energy $E(s, S)$ is infinite. This imposes hard constraints on the solutions of our design problem; in [16], we proved that these constraints make even the counting of valid solutions (with implications on controlled sampling) #P-hard.

In our model, in line with [16], we express a relatively simple energy function $E(s, S)$, namely

$$E_{bp}(s, S) = \sum_{(i,j)\in S} \mathsf{BPEnergy}(s_i, s_j)$$

where $\mathsf{BPEnergy} : \mathcal{B} \to \mathbb{R}$ is a function assigning values to single base pairs. Note that we empirically demonstrated the direct use of this simple energy model for design sampling [16] (apart from being extensible to more accurate models). This is in remarkable contrast to structure prediction, which for relevant accuracy relies on models that assign energies to stabilizing and destabilizing loops [40]. Figure 5B shows that sampling based on the simple base pair model can produce controllable concentrated distributions with regard to Turner energies [41]. This effect is studied in more depth in [16].

Feature network for design.

We express Problem 4 as a feature network and use Infrared to solve it. The FN $\mathcal{N}_{\text{design}}$ is composed of

- $\mathcal{X}_{\text{design}} = \{X_1, \ldots, X_n\}$;
- $\mathcal{D}_{\text{design}} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}^n$;
- $\mathcal{C}_{\text{design}} = \{\mathsf{BPCompl}_{[i,j]} \mid (i,j) \in \bigcup_\ell S_\ell\}$;
- $\mathcal{F}_{\text{design}} = \{F_{gc}, F_1, \ldots, F_k\}$ with features $F_{gc} = \{\mathsf{GC}_{[i]} \mid i \in [1,n]\}$ and $F_\ell = \{\mathsf{BPEnergy}_{[i,j]} \mid (i,j) \in S_\ell\}$ $(1 \le \ell \le k)$.

The constraint $\mathsf{BPCompl}_{[i,j]}(x_i, x_j)$ is True if $(x_i, x_j) \in \mathcal{B}$; it ensures that (i,j) is a canonical base pair in the design w. The network functions $\mathsf{BPEnergy}_{[i,j]}$ and $\mathsf{GC}_{[i]}$ decompose the global properties, energy and GC content, into their local contributions from base pairs or bases. To evaluate the assignment, feature F_{gc} has a weight of $\alpha_{\mathtt{GC}}$ and each feature F_ℓ has α_ℓ for $\ell \in [1,k]$.

Efficient solving in Infrared.

To randomly generate r designs, sampled exactly from the defined distribution of Problem 2, we encode $\mathcal{N}_{\text{design}}$ as an object of the class `infrared.Model` and pass it to Infrared's sampler `infrared.Sampler`, which is then asked r-times to return a sample. The efficiency of sampling depends exponentially on the complexity of the graph $G_{\text{design}} = (\{1, \ldots, n\}, \bigcup_{\ell=1}^{k} S_\ell)$, which combines all the dependencies between sequence positions due to the target structures.

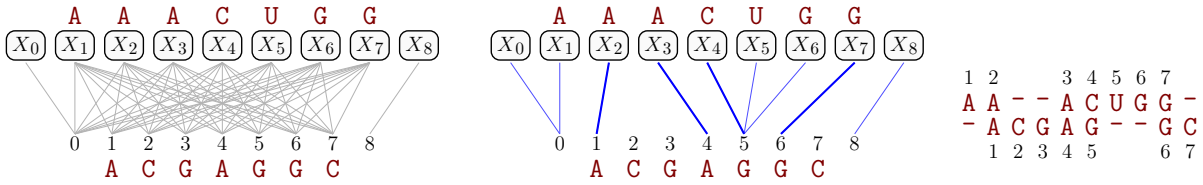

Fig. 6 Modeling the sequence alignment of `AAACUGG` and `ACGACGC`. From left to right, we illustrate the alignment model $\mathcal{N}_{\text{ali}}$; a valid assignment; the corresponding alignment.

Corollary 2. Infrared's engine solves Problem 4 in $\mathcal{O}((n+m)\cdot 4^w) + r\cdot 4(n+m))$ time and $\mathcal{O}(n\cdot 4^w)$ space, where $m = n + 2\sum_{\ell=1}^{k} |S_\ell|$, i.e. the total number of functions and constraints, and w is the treewidth of G_{design}.

Discussion.

Multitarget design well showcases the advantages due to a declarative system. Thus, we quickly developed RNARedprint v2 with extended functionality and improved performance compared to our original C++ implementation of RNARedprint [16]. Notably, based on the presented model, this reimplementation has identical computational complexity (Cor. 2).

As expected, the declarative modeling framework in Python strongly facilitated the reimplementation and extension. The performance improvements (Fig. 11A) can be attributed to Infrared's systematic Python/C++ hybrid design, which allowed us to better optimize its generic computational engine.

Our Jupyter notebook for multitarget design shows that RNARedprint's targeted sampling functionality can be coded in less than 100 lines of Python. Due to Infrared, this code is extensible and adaptable and makes the functionality well accessible for integration in larger Python workflows, for example, design involving negative design criteria that complements exact sampling with heuristic optimization (see [28]). Finally, the Infrared implementation can serve as a basis and "rapid prototyping" experimentation platform for future extensions and ideas on multitarget design.

4.3 Sequence alignment

Expressing sequence alignment, one of the most prominent problems of bioinformatics, in our framework enables solving various more expressive, even highly complex types of alignment by extending the model. To give an example, we later (Sec. 5.2) discuss the extension to pseudoknotted RNA structure alignment, close to LicoRNA [3]. We start by modeling the elementary problem, which has well-known efficient solutions [42, 43] by classic dynamic programming. The extension of this first model from linear to affine gap cost is discussed in Section 5.1.

Definition 13 (Sequence alignment)**.** A **sequence alignment** $\mathcal{A}$ of two **sequences** a and b (both are words over Σ) is a sequence of pairs (aka **alignment columns**) composed of $(\Sigma\cup\{-\})^2 \setminus \{(-,-)\}$ such that removing $-$ from the words composed of the first (resp. second) letter of all pairs yields a (resp. b). Let (i,j) be a pair in the alignment. We say (i,j) is a match if i equals to j, an insertion if i is `-`, a deletion if j is `-`, and a mismatch otherwise.

For simplicity, we begin our discussion with **linear gap cost** scoring models, where the score of an alignment $\mathcal{A}$ is defined by gap cost γ and an **elementwise score** $\sigma : \Sigma^2 \to \mathbb{R}$, as

$$\text{score}(\mathcal{A}) = \sum_{i:A_i \text{ match}} \sigma(A_i) + \#\text{gaps}(\mathcal{A})\gamma,$$

where $\#\text{gaps}(\mathcal{A})$ denotes the number of insertions and deletions in $\mathcal{A}$.

Consider two RNA sequences AAACUGG and ACGACGC ($\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$). Assuming similarity scores 2 for matching, and uniformly -1 for insertion and deletion, their alignment

$$\begin{array}{c} \mathtt{A}_1\mathtt{A}_2 - - \mathtt{A}_3\mathtt{C}_4\mathtt{U}_5\mathtt{G}_6\mathtt{G}_7 - \\ - \mathtt{A}_1\mathtt{C}_2\mathtt{G}_3\mathtt{A}_4\mathtt{G}_5 - - \mathtt{G}_6\mathtt{C}_7 \end{array}$$

has a score of $6 - 3 - 3 = 0$ due to three matches, three insertions, and three deletions.

The **alignment problem** takes two sequences, denoted a of length n and b of length m, and an elementwise score σ. Assuming that σ defines a similarity, it asks for maximizing the score($\mathcal{A}$) over all alignments and an optimal alignment $\mathcal{A}^*$.

Modeling alignment.

We model this problem by introducing one variable X_i per position i of the first sequence, whose values indicate their alignment to positions in the second sequence. Before stating our model, we need to resolve a significant issue with this idea. If we express assignments (match/mismatch) between positions i of a and j of b directly as assignment of j to X_i ($x_i = j$) then how do we express deletions of i? Naively introducing a *special* value for deletion, e.g. $\perp := m + 1$, makes it *difficult* to express the **noncrossing condition** on assignments, namely the positions j of b can be assigned to positions i of a in increasing order ($i > i'$ implies $j > j'$). More precisely, naive encoding introduces inequality-like constraints between all pairs of variables X_i and $X_{i'}$ ($1 \leq i < i' \leq n$).

Instead, following [3, 20], we model the deletion of a position i by assigning the same value to X_i and X_{i-1}. This keeps the assigned values in increasing order and allows a unique representation of alignments by assignments. To further facilitate modeling, we introduce *sentinel* variables $X_0 = 0$ and $X_{n+1} = m + 1$. As illustrated in Figure 6, our example alignment is then encoded by the assignment

i	0	1	2	3	4	5	6	7	8
x_i	0	0	1	4	5	5	5	6	8

.

With this idea (illustrated in Fig. 6), the feature network $\mathcal{N}_{\text{ali}}$ is formalized by

- $\mathcal{X}_{\text{ali}} = \{X_0, \ldots, X_{n+1}\}$;
- $\mathcal{D}_{\text{ali}} = \{0\} \times \{0, \ldots, m\}^n \times \{m + 1\}$;
- $\mathcal{C}_{\text{ali}} = \{\mathsf{Leq}_{[X_{i-1}, X_i]} \mid i \in [2, n]\}$;
- $\mathcal{F}_{\text{ali}} = \{F_{\text{match}}, F_{\text{insertion}}, F_{\text{deletion}}\}$ with
  - $F_{\text{match}} = \{\mathsf{Match}_{[X_i]} \mid i \in [1, n]\}$;
  - $F_{\text{deletion}} = \{\mathsf{Deletion}_{[X_{i-1}, X_i]} \mid i \in [1, n]\}$.
  - $F_{\text{insertion}} = \{\mathsf{Insertion}_{[X_{i-1}, X_i]} \mid i \in [1, n + 1]\}$;

The constraint $\mathsf{Leq}_{[X_{i-1}, X_i]} : (x_{i-1}, x_i) \mapsto (x_{i-1} \leq x_i)$ ensures an increasing order of the values in the assignment. The network functions express the alignment score:

$$\mathsf{Match}_{[X_{i-1}, X_i]}(x_{i-1}, x_i) = \begin{cases} \sigma(a[i], b[x_i]) & x_{i-1} < x_i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{Deletion}_{[X_{i-1}, X_i]}(x_{i-1}, x_i) = \begin{cases} \gamma & x_{i-1} = x_i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{Insertion}_{[X_{i-1},X_i]}(x_{i-1},x_i) = \begin{cases}\gamma(x_i - x_{i-1} - 1) & x_{i-1} \neq x_i \\ 0 & \text{otherwise.}\end{cases}$$

Efficient solving.

Infrared's general solving mechanism computes alignments based on this model in $\mathcal{O}(n \cdot m^2)$ time, dominating the $\mathcal{O}(nm)$ time for the traceback, and $\mathcal{O}(nm)$ space following Propositions 4 and 5 (treewidth 1; n variables with domains of size m; $\mathcal{O}(n)$ functions, each evaluated in constant time).

Note that while this automatic solution is efficient, it is still more costly than the known dynamic programming alignment algorithms by a linear factor. (In more detail, it does not profit from the linear cost of insertion; one could, within the same complexity, encode nonlinear insertion cost by modifying the functions Insertion.) This issue has been discussed and solved before for the case of sequence alignment based on the presented model [3, 20]; essentially it can be solved by applying DP to process single bags. Resolving this issue in broader generality is an open problem, whereas in principle the known specific solutions for sequence alignment can be implemented in the framework.

In practice, this issue is strongly alleviated by *banding strategies* that limit the domain size to $\mu \ll m$; this reduces the complexity to $\mathcal{O}(n\mu^2)$ time (and $\mathcal{O}(n\mu)$ space).

5 Model extensions and advanced topics

5.1 Sequence alignment with affine gap cost

For more realistic alignments, the cost of consecutive runs of insertions and deletions (aka **gaps**) is scored in a nonlinear fashion; e.g. k consecutive insertions are evolutionarily *less costly* than k independent insertions. This motivates redefining the score of an alignment $\mathcal{A}$:

$$\text{score}'(\mathcal{A}) = \sum_{i:A_i \text{ match}} \sigma(A_i) + \text{gapcost}(\mathcal{A}),$$

where generally $\text{gapcost}(\mathcal{A}) = \sum_{\text{gap of length } \ell \text{ in } \mathcal{A}} g(\ell)$. For $g(\ell) := \gamma\ell$, this score degenerates to the case of linear gap cost. The most prominent case is **affine gap cost**, where $g(\ell) := \beta + \gamma\ell$, distinguishing gap opening β from gap extension cost γ.

Underlining the asymmetry of $\mathcal{N}_{\text{ali}}$, we could extend the model to arbitrary cost of insertions by redefining $\mathsf{Insertion}_{[X_{i-1},X_i]}(x_{i-1},x_i) := g(x_i - x_{i-1} - 1)$; however, modeling affine cost for deletions cannot be expressed in a direct modification of $\mathsf{Deletion}_{[X_{i-1},X_i]}(x_{i-1},x_i)$ since we lack information to distinguish gap opening and extension.

One can envision at least two possible fixes. First, we can replace the binary deletion network functions with ternary functions that depend on X_{i-2}, X_{i-1}, X_i. This extension comes at the price of increasing the tree width by 1 (and thus the complexity by a further factor of m.) Second, we can introduce additional Boolean variables Y_i to reflect the matching state at position i: Y_i is assigned to $y_i = 1$ if i is matched; $y_i = 0$, if i is deleted. In turn, the deletion function can be modified to depend on X_{i-1}, X_i and Y_{i-1}:

$$\mathsf{Deletion}_{[X_{i-1},X_i,Y_{i-1}]}(x_{i-1},x_i,y_{i-1}) = \begin{cases}\beta + \gamma & x_{i-1} = x_i \text{ and } y_{i-1} = 0 \\ \beta & x_{i-1} = x_i \text{ and } y_{i-1} = 1 \\ 0 & \text{otherwise.}\end{cases}$$

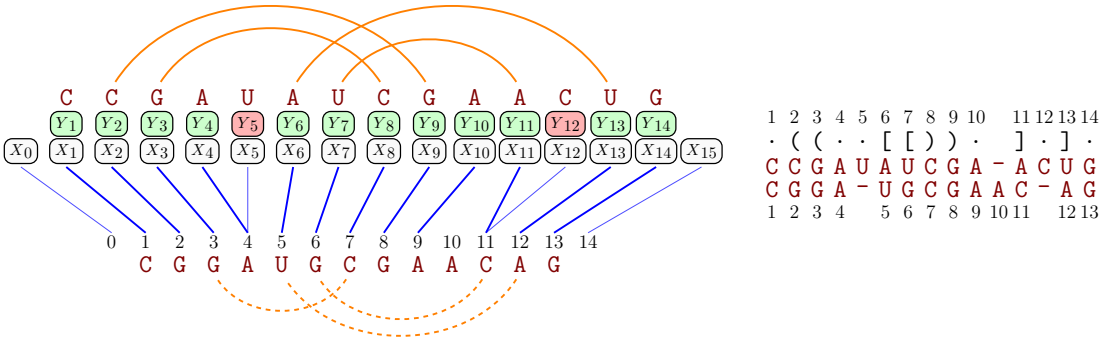

Fig. 7 Modeling sequence structure alignment. Example of a valid assignment and corresponding alignment with a pseudoknotted structure. The model contains one network function BPMatch per input base pair (arcs on top). These functions contribute τ for matches to canonical bases (dashed arcs).

Complexity.

For the first idea, we derive a time complexity of $\mathcal{O}(nm^3)$ (Prop. 4), since the treewidth is 2. In the second model, adding Boolean variables (and ternary constraints to relate them to the X variables) technically increases the treewidth, but since the variables Y_i have a domain size of 2, in contrast to the linear domain size of the variables X_i, their effect on the complexity is much lower (in this case, even constant in sequence length).

Here, the direct application of Proposition 4 would strongly overestimate; instead we follow the argumentation of Section 3.5. The introduced Y_i variables each correspond to the X_i variable of the same index. Collapsing the nodes of these corresponding variables in the dependency graph, let us us decompose it with width 1. Thus, we bound the time complexity by $\mathcal{O}(n \cdot m^2 2^2)$; see also our discussion of the linear case. The Y variables thus contribute a constant factor of 4, comparable to the overhead of Gotoh's algorithm [43] over linear gap cost alignment (approximately factor 3). Thus, the second model improves the first one by a linear factor—intuitively, it allows sharing Boolean variables between bags instead of variables of linear domains.

5.2 From sequence alignment to pseudoknot sequence-structure alignment

We will develop Infrared models for RNA alignment, where the first RNA is annotated by a potentially crossing secondary structure. We build on the sequence previously described alignment model $\mathcal{N}_{ali}$. Recall our definition of RNA secondary structure from Section 4.2; here, we will explicitly consider *general* secondary structures, where base pairs can cross and thereby form arbitrary pseudoknots. This means that we are solving the essentially same optimization problem as LicoRNA [3]. While LicoRNA implements hand-crafted, specialized dynamic programming algorithms, Infrared automatically derives closely related algorithms from a network model, typically from less than 100 lines of Python code. These algorithms solve the pseudoknotted RNA alignment problem efficiently for the same fixed treewidth parameter.

Given are two RNA sequences a and b of respective length n and m, additionally a general (i.e. not necessarily noncrossing, potentially pseudoknotted) RNA secondary structure S of length n; S is also called **arc-annotation** of a.

We are interested in optimizing a type of alignment score that takes the structural relations due to the arc annotation into account; see Figure 7. To demonstrate the principle, we extend the sequence alignment score of the previous section by an **arc match bonus** τ. Let us thus define our **sequence structure alignment score** by

$$\text{score}_S(\mathcal{A}) = \text{score}(\mathcal{A})$$

$$+ \sum_{(i,j)\in S} \begin{cases} \tau & \mathcal{A} \text{ matches } i \text{ to } i' \\ & \text{and } j \text{ to } j';\ (b[i'], b[j']) \in \mathcal{B} \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{B}$ is the set of canonical base pairs (Sec. 4.2).

Problem 5 (General sequence-structure alignment)**.** Given sequences a, and b annotated by S, the **sequence structure alignment problem** asks for a sequence alignment of a and b (Def. 13) that optimizes the sequence structure alignment score $\text{score}_S(\mathcal{A})$.

Our feature network model $\mathcal{N}_{\text{sali}}$ directly builds on $\mathcal{N}_{\text{ali}}$, extending it by network functions to encode the structure component of the score. As discussed in the previous subsection (for the purpose of modeling affine gap cost), we introduce Boolean variables Y_i to indicate the match of position i in a since they let us express the arc match bonus more efficiently. We obtain

- $\mathcal{X}_{\text{sali}} = \mathcal{X}_{\text{ali}} \cup \{Y_1, \ldots, Y_n\}$;
- $\mathcal{D}_{\text{sali}}$ extends $\mathcal{D}_{\text{ali}}$ by Boolean domains $\{0,1\}$ for all Y_i;
- $\mathcal{C}_{\text{sali}} = \mathcal{C}_{\text{ali}} \cup \mathcal{C}_{\text{relXY}}$;
- $\mathcal{F}_{\text{sali}} = \mathcal{F}_{\text{ali}} \cup \{F_{\text{bpmatch}}\}$;

where $\mathcal{C}_{\text{relXY}}$ is a set of constraints that relate the variables Y_i, X_{i-1} and X_i, such that $y_i = 1 \iff x_{i-1} < x_i$ (for all $1 \le i \le n$) and $F_{\text{bpmatch}} = \{\mathsf{BPMatch}_{[X_i,X_j,Y_i,Y_j]} \mid (i,j) \in S\}$

$$\mathsf{BPMatch}_{[X_i,X_j,Y_i,Y_j]}(x_i, x_j, y_i, y_j) = \begin{cases} \tau & y_i = 1, y_j = 1, \text{ and } (b[x_i], b[x_j]) \in \mathcal{B} \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathsf{BPMatch}$ (for an arc $(i,j) \in S$) cannot be defined in dependency of only X_i and X_j, since $(b[x_i], b[x_j]) \in \mathcal{B}$ could hold in cases where i or j are deleted.

Complexity.

As in the analysis of Section 5.1, we collapse each pair of nodes of variables X_i and Y_i (of the same index i) in the dependency graph. The result is isomorphic to the *structure graph* of RNA a, consisting of its nucleotides as nodes, and edges due to its backbone and base pairs. For the treewidth w of this graph, we derive $\mathcal{O}(n2^{w+1}m^{w+1})$ time complexity by Corollary 1.

Whereas in our models for network parsimony or RNA design the domain size is constantly bounded, here it depends on the input size. Consequently, solving of this RNA alignment problem is not in parameterized complexity class **FPT**, but **XP** (Sec. 3.6).

Discussion.

The presented model extension yields an automatically derived solution to the pseudoknot sequence-structure alignment problem with parameterized complexity in the treewidth. Compared to LicoRNA, our algorithms depend on the exact same fixed parameter. Note that, in the current implementation, Infrared's complexity stays behind by a linear factor (in practice reduced to the band-width) due to the same reason as we discussed for sequence alignment before. This is contrasted by general benefits due to the declarative implementation in Infrared (Jupyter notebook). For example, the code is well maintainable, extensible by further constraints and evaluation criteria, and can profit from future developments and optimization of the Infrared system.

5.3 Finite state automata

A common side condition when designing RNA or DNA sequences is to avoid or enforce certain sequence motifs. For example, one could be interested to avoid stop codons anywhere in the designed sequence (or avoid restriction sites, enforce binding sites...). Such conditions can be generalized in

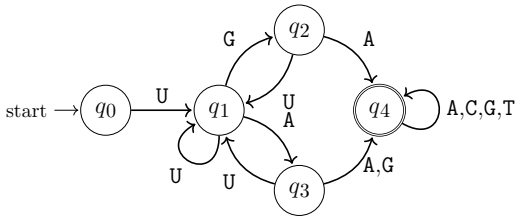

Fig. 8 Sketch of the 5-state Deterministic finite "Aho-Corasick" automaton accepting the three stop codons `UGA`, `UUA`, `UUG`. We do not draw back-transitions to q_0, which occur implicitly for all not explicitly shown cases (i.e. `A`,`C`,`G` in q_0; `C` in q_1, `C`,`G` in q_2; and `C` in q_3). To *forbid*, instead of accept, all of the three stop codons, we complement the language by making all states but q_4 accepting.

terms of regular languages, accepted by deterministic finite state automata (DFA; Fig. 8). DFAs have been introduced to sequence design before [44] to perform such tasks efficiently for a set of sequence motifs. We show that finite state automata can be emulated in network models. Remarkably, this allows us to efficiently handle such requirements even in combination with other design objectives; e.g. the automaton model of this section could be merged with our model for multitarget RNA design $\mathcal{N}_{\text{design}}$ (Sec. 4.2).

The model is a good example for the use of several types of variables, as we are going to introduce, for every sequence position, one variable to model the nucleotide and one to model the automaton state.

Definition 14 (Deterministic Finite Automaton)**.** A **Deterministic Finite Automaton (DFA)** is a 5-tuple $(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_F)$ with

- Finite set of states $\mathcal{Q}$;
- Finite set of symbols Σ;
- Transition function $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$;
- Initial state $q_0 \in \mathcal{Q}$;
- Set of final, accepting states $\mathcal{Q}_F \subset \mathcal{Q}$.

A word $w = a \dots w_n$ of length n is accepted by a DFA if there exists a sequence of states $q = \{q_0, \dots, q_n\} \subset \mathcal{Q}^{n+1}$ starting with initial state q_0 such that $q_n \in \mathcal{Q}_F$ and $\delta(q_{i-1}, w_i) = q_i$ for all $i \in [1, n]$.

By modeling a DFA as a network model, we can use Infrared to sample accepted words. We consider two types of variables, one for the word w and the other for the state sequence q. Given a DFA, the accepted word sampling problem is formalized by the feature network $\mathcal{N}_{\text{DFA}}$ as follows:

- $\mathcal{X}_{\text{DFA}} = \{X_1, \dots, X_n\} \cup \{Y_0, \dots, Y_n\}$;
- $\mathcal{D}_{\text{DFA}} = \Sigma^n \times \{q_0\} \times \mathcal{Q}^{n-1} \times \mathcal{Q}_F$;
- $\mathcal{C}_{\text{DFA}} = \{\mathsf{Transition}_{[X_i, Y_{i-1}, Y_i]} \mid i \in [1, n]\}$;
- $\mathcal{F}_{\text{DFA}} = \{\}$.

The constraint $\mathsf{Transition}_{[X_i, Y_{i-1}, Y_i]} : (x_i, y_{i-1}, y_i) \mapsto (y_i = \delta(y_{i-1}, x_i))$ encodes the DFA transition function. This ensures that, in each sampled assignment, $y_0 \dots y_n$ is the state sequence of the word $x_1 \dots x_n$, which is accepted by DFA as the domain of Y_n is the set of final states $\mathcal{Q}_F$.

Complexity.

Again, we collapse the variables X_i and Y_i for the same i in the dependency graph; let w be the treewidth of the collapsed graph. We obtain time complexity $\mathcal{O}(4^{w+1}|\mathcal{Q}|^{w+1}n)$ by Cor. 1, where $w = 1$ for the pure automaton model (without extensions).

When forbidding/enforcing motifs in other design settings, e.g. single or multitarget RNA design, the treewidth typically increases since the automaton model causes dependencies between variables of consecutive positions i and $i+1$, while e.g. RNA design defines dependencies between nonconsecutive positions i and j for each target base pair (i, j). Based on this analysis, we achieve efficiency equivalent

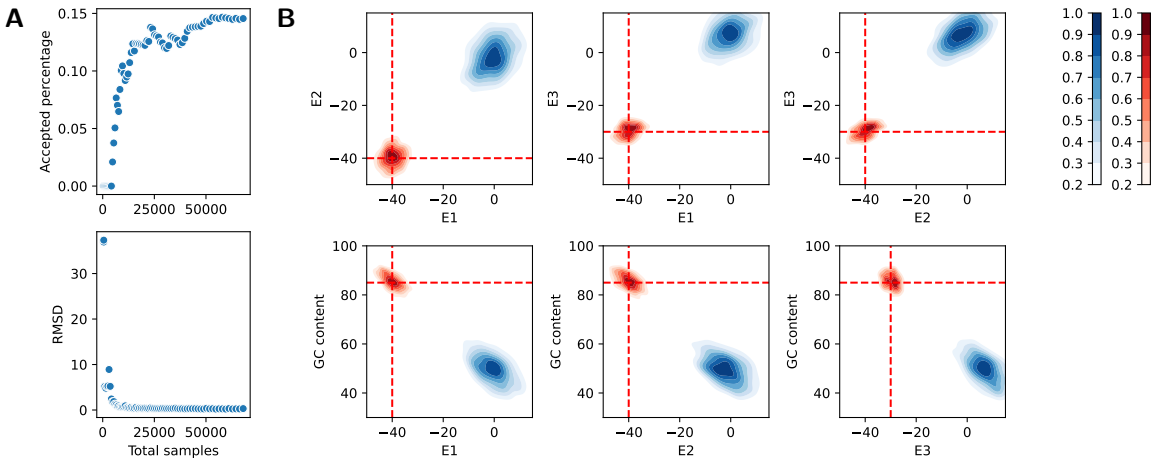

Fig. 9 Multidimensional Boltzmann sampling applied to RNA design. For the example of Figure 5, we target GC content 85% and respective energies E1=-40, E2=-40, E3=-30 for the target structures (with tolerances of 5% GC content and 0.5 kcal/mol energy). Infrared's multidimensional Boltzmann sampling (MDBS) strategy starts from uniform sampling (weights 0 for every feature). It iteratively generates Boltzmann samples and updates the weights to move the (estimated) expectation closer to the targets. **A)** Accepted samples as well as root mean square distance (RMSD) to the targets during this procedure, which considered over 70,000 total samples to generate 100 targeted samples. **B)** Kernel density estimate plots: distributions of features for uniform sampling (blue) and sampling at the end of the MDBS run (red), where distributions are shifted to the targets (dashed red lines).

to that of the hand-crafted algorithm [44]. Since the domain size depends on the input size, specifically the number of states, this is another example of solving by Infrared in **XP** (Sec. 3.6).

The complexity due to the automaton should be compared to simpler ideas to enforce/forbid motifs of maximum size k. More naively, one could introduce such requirements by k-ary constraints on each run of k consecutive variables $X_i, \ldots, X_{i+k-1}$ (for all $1 \leq i \leq n-k+1$). This idea results in $\mathcal{O}(4^k n)$ without additional constraints. Automata thus offer a favorable trade-off between domain size and treewidth/exponent (as advocated in [44]). Infrared supports adapting the strategy to the concrete problem.

5.4 Multidimensional Boltzmann sampling

Recall that Section 4.2 demonstrated random generation of solutions by sampling from the Boltzmann distribution defined by multiple features and weights. The histograms from Figure 5 show the feature distributions resulting from large negative weights for all features in a multitarget RNA design example. In the example, this allows us to produce designs with low target structure energies and GC content.

Substantially extending the level of control, Infrared supports the random generation of objects with narrowly defined target feature values based on multidimensional Boltzmann sampling (MDBS) [15]. This technique was successfully demonstrated before in RNA design: generating sequences with defined dinucleotide frequencies [45], targeting GC content in single target RNA design by IncaRNAtion [46] and generating RNA designs with specific energies of multiple structures and specific GC content by RNARedprint [16].

Problem statement.

Concretely, given a network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$, we look at the problem of randomly generating valid assignments x that satisfy constraints

$$|F(x) - \tau_F| \leq \delta_F,$$

for given target values τ_F and tolerances δ_F for all (or a subset) of the features $F \in \mathcal{F}$. Let us call such assignments (τ, δ)**-admissible**.

Algorithm 4: Multidim. Boltzmann sampling

Input : Feature network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$; targets τ and tolerances δ; initial weights α; integers K, k, parameters $\gamma > 0$, $\Gamma \geq 1$

Output: K 'targeted' (τ, δ)-admissible samples

$\mathcal{S} \leftarrow \emptyset$;

while True **do**

  compute partition functions for $\mathcal{N}$ at weights α;

  $\mu \leftarrow \vec{0}$;

  for $1 \dots k$ **do**

    $s \leftarrow$ sample from $\mathcal{N}$ at α;

    $\mu_F \leftarrow \mu_F + F(s)/k$ **for all** $F \in \mathcal{F}$;

    if $|F(s) - \tau_F| \leq \delta_F$ **for all** $F \in \mathcal{F}$ **then**

      $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$;

      return $\mathcal{S}$ **if** $|\mathcal{S}| \geq K$

  $\alpha_F = \alpha_F + \gamma \cdot (\tau_F - \mu_F)$ **for all** $F \in \mathcal{F}$;

  $r \leftarrow \sqrt{|\mathcal{F}|^{-1} \sum_{F \in \mathcal{F}} (\mu_F - \tau_F)^2}$;

  if r *didn't improve (over previous iteration)* **then**

    $\gamma \leftarrow \gamma / \Gamma$;

    $k \leftarrow k \cdot \Gamma$

MDBS strategy and algorithm.

As shown in [15] the problem can be solved effectively, under certain assumption even with provable efficiency, by MDBS. This strategy combines rejection sampling, which accepts only (τ, δ)-admissible samples, with a learning strategy to maximize its efficiency.

Here, we observe that rejection sampling is most effective, when the targeted values τ_F coincide with the means of the sampled distributions; moreover, these means depend on the feature weights α.

Therefore, Infrared's MDBS algorithm (Algorithm 4), starting from initial weights α (by default, $\alpha = \vec{0}$), iteratively generates k-many samples per round. In every iteration, it tweaks the weights α aiming to shift the sampling means closer to the targets; the update step-size is controlled by the tweaking factor γ. The procedure is repeated until K-many (τ, δ)-admissible samples are generated. To stabilize this heuristic strategy, Infrared additionally implements an annealing scheme based on improvement of the root mean square deviation (RMSD) to the targets and controlled by the cooling factor Γ.

MDBS for RNA design.

Figure 9 illustrates this MDBS strategy for the example of Figure 5 and specific energy and GC content targets. Showing typical behavior, the strategy improves the RMSD while generating admissible and nonadmissible samples. In this way, it increases the efficiency of generating admissible assignments (Fig. 9A). Figure 9B shows how MDBS shifts the multivariate distribution toward the targets (here starting from uniform sampling).

Targeting by proxy.

The multitarget design example showcases an interesting extension of the standard MDBS mechanism. Namely, in this case, we distinguish base pair energy from Turner energy. To target the latter, we use base pair energies as proxies, since they allow much more efficient sampling (and are sufficiently correlated to Turner energies; compare [16]). To shift the distributions during the MDBS algorithm, we thus estimate the means of the *Turner energies*; then, based on their difference from the target Turner energies, we update the weights of the corresponding *base pair energy* feature. Our Infrared implementation supports 'targeting by proxy' in a generalized way (using a second kind of feature F whose evaluations $F(x)$ are defined explicitly, instead of being induced by their network functions).

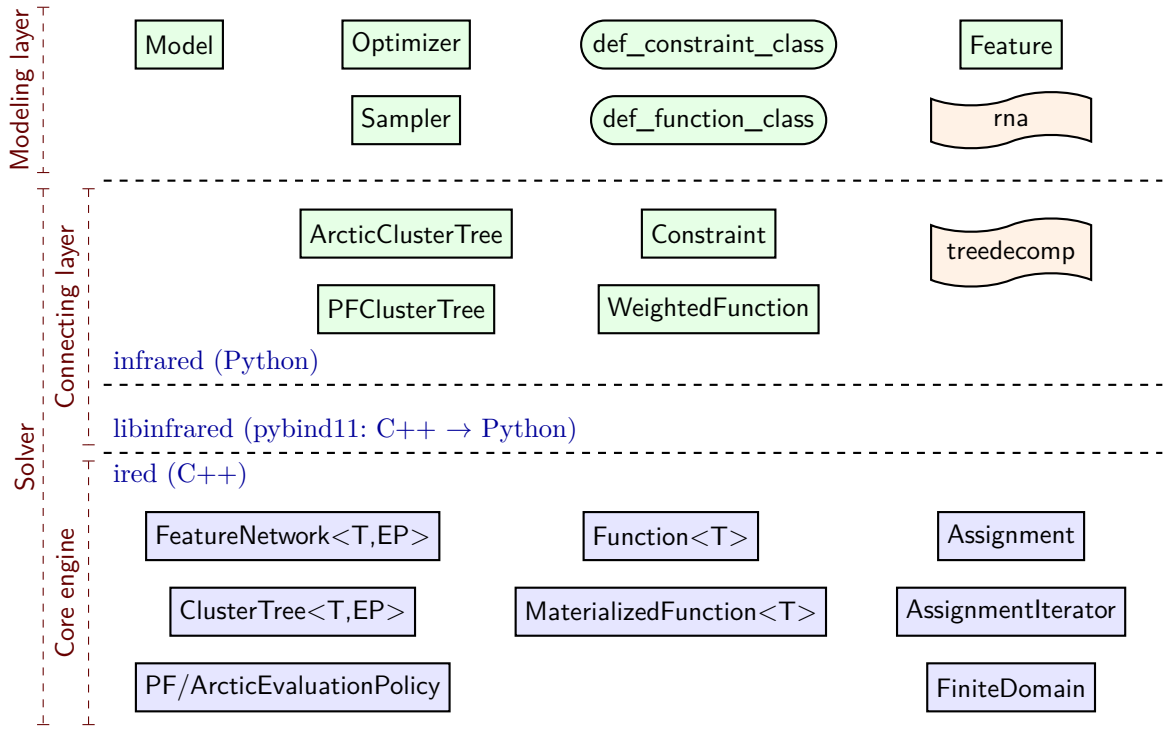

Fig. 10 Overview of the Infrared architecture. The C++ **core engine** is connected to a high-end **modeling layer** in Python by a hybrid **connecting layer**. The C++ core implements the computational engine to evaluate forward and traceback algorithms on cluster trees as generic code (e.g. supporting different algebras as evaluation policies; `PF` for partition function, `Arctic` for maximization). This optimized low-level layer is exposed to Python using `pybind11`; the core functionality is moreover extended (by tree decomposition, using module `treedecomp`, population of the cluster tree...) on the Python side to implement the full computational engine. Finally, the high-level modeling interface of Infrared offers functionality to model and solve feature networks, perform sampling targeting specific features (multidimensional Boltzmann sampling), define custom constrain and function types. Moreover, we include the module `rna` for RNA-specific functionality.

Available code examples.

In supplemental online material (Jupyter notebooks), we show the code to produce the samples and plots for Figure 9; as a further example, we demonstrate effective random sequence generation targeting all 16 dinucleotide frequencies of a SAM riboswitch (RF00162; from *S. thermophilum*), while maintaining compatibility with its pseudoknotted RNA structure.

6 Implementation

The Infrared software supports declarative modeling of problems as feature networks and treewidth-dependent efficient solving through a high-level Python interface. Figure 10 sketches its architecture. For solving, the software relies on optimized implementations of the presented algorithms in C++. The low-level C++ engine is glued to a high-level 'modeling' layer by a pybind11-based C++/Python interface. Thus, Infrared maintains a small algorithmic core in C++ (for high performance), while extending functionality in Python (for increased flexibility). For example, the C++ **core engine** solves cluster trees and focuses on *necessary* functionality, while the construction of the cluster tree from a model, as well as "high-level" functionality such as the declarative composition of models, are implemented in Python.

The C++ code is templated to generically support different function types and evaluation algebras, which keeps the code compact and maintainable; moreover it prepares future extensions of the system. For fast processing of bags, the core engine relies on fast backtracking enumeration of partial assignments (class `AssignmentIterator`), where constraints and functions are evaluated as early

as possible (to avoid unnecessary and redundant computation). Bag processing evaluates constraints (`Function<bool>`) and functions (`Function<double>`) and, in the forward phase, computes messages, stored in memory as objects of `MaterializedFunction<double>`.

For completion of the Infrared solver, the **connecting layer** exposes the C++ functionality to Python, specializing the templates to optimization (arctic policy) and partition function over real-valued features. Moreover, it extends the core by Python wrapper classes that 'know' how to construct cluster trees from feature networks. To prepare the definition of function and constraint classes, it wraps the Boolean and real-valued C++ function classes. Generating tree decompositions is delegated to the module `treedecomp`. While we provide interfaces to different external tree decomposers and support customization, the implemented default strategy applies a randomized min-fill-in heuristic [13] and returns a minimum width tree decomposition from iterated calls.

Finally, the **modeling interface** layer enables a declarative style of defining feature networks as objects of the class `Model` through Python code. Adding variables, constraints, and functions supports naturally specifying, but also extending and merging feature networks. From models one can construct solvers to perform different tasks, including optimization, Boltzmann sampling at specified feature weights or targeted sampling; the latter relies on a multidimensional Boltzmann strategy that learns weights to effectively target specific feature values.

Definition of application-specific constraints and functions.

Infrared supports the definition of constraints and functions in Python by special concise syntax via the respective functions `def_constraint_class` and `def_function_class`. Examples of their use were given before for defining the constraint `NotEquals` and network function `Card` for the introductory graph coloring model. To create Python classes of constraints or network functions, the user calls these functions with the class name and two Python functions. The first function (`init`) has two roles. It defines the arguments of the constructor and returns the scope (or dependency list) of the constraint or function. The second function (`value`) defines how the constraint/function is evaluated at specific values of the dependency variables. Using arguments of the same name, information can be passed from initialization to evaluation; e.g. this enables constraint/function type arguments or auxiliary data structures. For clear semantics (while allowing optimizations), we require referential transparency, i.e. the result of the value function must not vary with anything but its arguments.

Precomputation.

The core engine precomputes constraints and network functions when they are added to the cluster tree. For this purpose, they are evaluated for all partial assignments and the results are tabulated (`MaterializedFunction`). This simple mechanism supports the convenient specification of constraints and functions in Python, while resulting in fast computation times in practice (by significantly reducing the overhead due to the Python computation). From a theoretical perspective, this mechanism preserves the worst case time complexity, since k-ary constraints and functions impose a bound on the treewitdh $w \geq k-1$. The strategy requires additional space in $\mathcal{O}(d^k \mathtt{m})$, for maximum domain size d and $\mathtt{m}$ constraints and functions. Space and time consumption due to the precomputation are thus dominated by the solving complexity if $w > k-1$. In this way, the strategy is optimized for the typical performance-critical cases. Future implementations can speed up the precomputation by possibly lazy caching mechanisms without changing semantics and the interface.

7 Discussion

The Infrared framework was motivated by the success of related technology in solving complex bioinformatics problems; most directly, by our work on multitarget RNA design [16]. Thus, the system started out as a library that generalized the fixed-parameter tractable (FPT) sampling algorithm of RNARedprint and its multidimensional Boltzmann sampling strategy. Since then it has been developed into a broadly applicable framework, supporting convenient declarative modeling of problems with multiple features, where models can be solved by a generic treewidth-based algorithm using different

algebras. This text was written to supply the reader with a comprehensive discussion of the techniques combined in the Infrared system (cf. our text [28] focusing on coding of design problems in Infrared; lacking in-depth discussion of methods).

Since the system's first application for concisely reimplementing RNARedprint with improved functionality and performance, it has proven to be a very useful tool for further algorithmic developments in bioinformatics [17, 19, 47]. Other previous work [3, 20, 44, 46] could have directly profited from the Infrared framework. For several previous algorithms [3, 17, 44, 46], we presented feature network models and discussed their solving complexity. For these examples, the system yields essentially identical algorithms; with the exception of alignment, where Infrared lacks a problem-specific optimization (Secs. 4.3 and 5.2), which we plan to add in the future. From the well-researched field of network parsimony, we present further examples where the Infrared's solving complexity is on par with state-of-the-art algorithms [5] (even improving softwired parsimony on binary networks).

Utiltity for prototyping and practical applications.

In summary, these previous works witness the suitability of Infrared for prototyping novel algorithmic ideas; moreover, their benchmark results show the practical utility of the system to solve relevant problem instances. Of particular interest, we show that—for many practically relevant problem instances—the treewidth is sufficiently low to enable effective solving by Infrared.

In addition, we wanted to learn about Infrared's practical performance in relation to optimized problem-specific code in a high-performance computing language (Fig. 11A). Taking a unique opportunity, we compared our original C++ implementation of multitarget design [16, 48], RNARedprint v1, to our Infrared-based reimplementation RNARedprint v2. We chose the benchmark set "RNAfold", e.g. used in [16, 49], comprising 400 design instances of 3–6 structures, which were generated to pose ambitious challenges with treewidths up to 16. This experiment was performed on an Intel Core i7-4770 CPU with 32 GB memory.

Moreover, we studied the practical applicability of treewidth-based network parsimony algorithms (Sec. 4.1) on a set of nonartificial phylogenetic networks [50] compiled from the literature (see https://phylnet.univ-mlv.fr/recophync/networkDraw.php). This data set is typically used as a reference set for the comparison and evaluation of various algorithms on phylogenetic networks. For our purposes, we determined the treewidths of the networks in dependency on their size and the number of reticulation nodes (Fig. 11B). Here treewidth directly provides information about the solving efficiency of hardwired and softwired parsimony problems (Sec. 4.1). The low to moderate treewidths on these instances hint at permissible performance in many real-word scenarios.

Characteristics and application range.

By modeling a series of concrete bioinformatics problems, we showed that Infrared is broadly applicable, going well beyond the selected examples. As discussed before, this extends to applicability *in practice*, where Infrared can efficiently solve relevant instances of expressible problems. Although feature networks are virtually universal, such that they do not limit the system's expressivity, Infrared's solver relies on a very specific mechanism, where efficiency strictly depends on the treewidth of the problem instance. Arguably, this is a prerequisite for the very characteristic properties of the framework. In contrast to heuristic methods, our tree decomposition-based solving strategy leads to predictable worst-case complexity guarantees for exact optimization *and* sampling. Notably, exact controlled sampling rules out many heuristic pruning-type solving strategies, since it requires exact computation of partition functions.

Nevertheless, the dependency on treewidth necessarily limits the *practically solvable* problems and instances. In practice, such problems (explicitly or implicitly) have some graph structure. Examples are graph coloring or multitarget RNA design, which both are NP hard, but efficiently solvable for specific instances, whose graphs are sufficiently close to trees.

Infrared was designed to handle such tree decomposable problems (and their low-treewidth instances) well, but its general solving mechanism offers the flexibility to go beyond. For example, the framework supports strategies that limit the treewidth of considered instances (e.g. we controlled

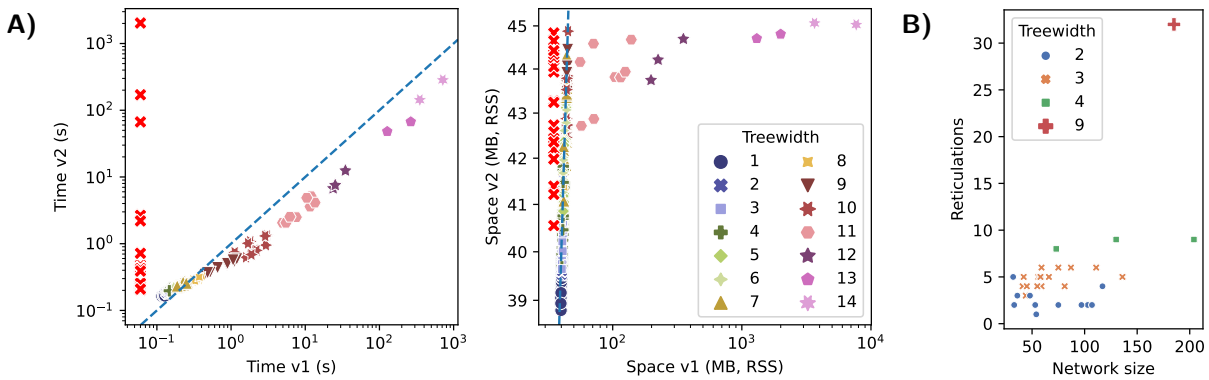

Fig. 11 **A)** Benchmark comparison of the Infrared-based RNARedPrint v2 to the original C++ implementation "v1". Time is measured as user time; space, as maximum resident set size (RSS). We run the tools on the RNAfold benchmark set [16]. We let both tools generate 1000 samples at fixed weights; note that time and space are strongly dominated by the precomputation phase. To directly compare the implementations of the core algorithms, we run both tools on identical tree decompositions, although Infrared's default tree decomposer improves for several instances (including the most expensive one). One observes that the RNARedPrint v2 improves in space and time over the original implementation. Only for very short runtimes, at low treewidth, the C++ implementation has a slight edge, presumably due to less overhead. Both implementation show almost no noticable space increase at low tree widths; however the space requirements of the original implementation increase dramatically for treewidths larger than 10. Due to its extreme space requirement, we didn't solve the single instance of treewidth 16 with RNARedPrint v1; in other cases, it failed due to a bug. For those instances, we indicate only the performance of version 2 (red crosses). **B)** We used Infrared to compute the treewidths for a set of various phylogenetic networks that were collected from recent studies [50]. Using the Infrared network parsimony model, we count the number of reticulation nodes in the networks and calculate their treewidth. It can be seen that the treewidth rather correlates with the number of reticulation nodes than with network size (number of nodes and edges). Our study on 'real-world' phylogenetic networks suggests that treewidths are often low in practice; consequently Infrared can effectively compute network parsimony by solving the presented models.

the maximum treewidth in our negative design approach RNAPOND [17]) or reduce the treewidth in controlled ways (e.g. TreeDiet [18]). For this class of problems, we identify potential for future improvements of the Infrared solver, which could allow instances to be solved significantly faster or with better complexity. For example, we discussed the complexity of alignment; additionally, consistency-based methods can yield significant improvements over the current evaluation strategy by cluster tree elimination.

Delimitation.

The system is, however, not designed and is even unsuitable for problems that cannot be modeled as a decomposable feature network. This comprises many constraint satisfaction (CSP) or constraint optimization problems (COP) that would typically be solved by general constraint solvers (e.g. the constraint programming system Gecode [26]), SAT solving (e.g. MiniSAT [51]), or solving of integer linear programs (e.g. CPLEX [52]). In many classical CSP examples (e.g. n-Queens, Sudoku), models induce a complete (or almost complete) dependency graph. Here, Infrared does not implement strategies to heuristically cope with the exponential worst case complexity (e.g. constraint propagation or branch and bound); its solving strategy would therefore be essentially brute force. Obviously Infrared is thus not a general solver for CSP/COP, even if its declarative modeling paradigm and interface remind of such systems.

While Infrared proposes a novel form of automated solving of declarative problem specifications by dynamic programming (DP), bioinformatics already has a long tradition of combining declarative methods and dynamic programming in the form of Algebraic Dynamic Programming (ADP) [53].

Despite the common DP backdrop, Infrared and ADP pursue different, even orthogonal goals: instead of deriving a DP algorithm from a declarative problem description, ADP implementations such as GAPC [54] or ADPFusion [55, 56] aim to support the implementation of DP algorithms through algebraic abstraction.

8 Conclusions

We presented a framework for rapidly developing applications that make use of efficient exact optimization and sampling techniques by declaratively specifying problems in Python. As such, the framework provides flexible access to recent advanced algorithmic techniques—while specifying problems resembles common constraint modeling systems.

The system allows modeling problems as feature networks, which we introduced as a form of weighted constrained networks that support several features. The main advantages and characteristics of the framework stem from combining expressive modeling with automated combinatorial solving strategies that support exact optimization and weighted sampling. In particular, exact sampling, which requires complete combinatorial algorithms, can be used in innovative ways. For example, it allows generating decoys and background models in complex settings or targeting multiple features in its extension to multidimensional Boltzmann sampling.

As elaborated, these tasks are performed by generic solving algorithms based on tree decompositions of the models. Being parameterized by the treewidth, this strategy profits from the often moderately low treewidth of many typical problems in bioinformatics.

We underline the broad range of possible applications, by our discussion of diverse application examples and their implementation (online documentation). Demonstrating the concise reimplementation of previous bioinformatics methods, these applications serve as reference coding examples and also show the practical relevance of the framework.

Crucially, the system makes such methods accessible through a declarative interface in Python. Since this strongly facilitates their flexible use, the system promotes future applications of these techniques. Increasing flexibility, the system supports extension and refinement of existing models as well as their composition, e.g. sequence design targeting structure RNA *and* forbidding specific sequence motifs.

Future work.

We plan to further optimize the Infrared solver due to consistency methods and/or forward checking (in single bag processing). In specific cases, such techniques even improve complexity bounds over the currently implemented CTE-like evaluation mechanism. Moreover, we want to adapt the linear-factor speedup over standard evaluation for alignment problems (LicoRNA)—generalizations of this technique pose interesting research questions. As another path of optimization, we will implement improved tree decomposition adapted to our solver.

Furthermore, the architecture of the framework enables additional solvers; for example, such solvers can compute Pareto-optima or perform nonredundant sampling [57] based on feature network models. Moreover, it will be interesting to explore the use of evaluation further algebras, which can be used with the generic evaluation algorithms of Infrared's C++ core engine.

Finally, the system highlights benefits due to tree decomposition of problems that enable general solving by efficient combinatorial methods. While such methods can be tedious to implement from scratch, we demonstrated their use through a declarative modeling interface. In future work, we envision developing and making related methods accessible in a similar way. One exciting line would extend our work on tree decomposition-based automatic generation of dynamic programming schemes [19].

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

Author's contributions. HTY and SW mainly wrote the methods and application examples; SW wrote the introduction, discussion and conclusion. SW implemented the Infrared framework and the accompanying application examples. HTY contributed crucially to software deployment and documentation and computed results. SJB wrote the network parsimony section and computed results. BM wrote the discussion of theoretical complexities and background on tree decompositions and parameterized complexity. YP and SW initiated the Infrared project. YP made indispensable contributions across the manuscript, improved the presentation and contributed details of multidimensional Boltzmann sampling. All authors read and approved the final manuscript.

Funding. This work is supported by the French national research agency ANR: grant ANR-21-CE45-0034-01 "INSSANE", as well as the Austrian Science Fund (FWF), grant no. I 4520. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 10102967.

References

[1] Wachsmuth, M., Domin, G., Lorenz, R., Serfling, R., Findeiß, S., Stadler, P.F., Mörl, M.: Design criteria for synthetic riboswitches acting on transcription. RNA Biol. **12**(2), 221–231 (2015). doi:10.1080/15476286.2015.1017235. 25826571

[2] Wu, M.J., Andreasson, J.O.L., Kladwang, W., Greenleaf, W., Das, R.: Automated design of diverse stand-alone riboswitches. ACS Synth. Biol. **8**(8), 1838–1846 (2019). doi:10.1021/acssynbio.9b00142

[3] Rinaudo, P., Ponty, Y., Barth, D., Denise, A.: Tree decomposition and parameterized algorithms for rna structure-sequence alignment including tertiary interactions and pseudoknots. In: Algorithms in Bioinformatics, pp. 149–164. Springer, Berlin, Germany (2012). doi:10.1007/978-3-642-33122-0_12

[4] Jabbari, H., Wark, I., Montemagno, C., Will, S.: Knotty: efficient and accurate prediction of complex rna pseudoknot structures. Bioinformatics **34**(22), 3849–3856 (2018). doi:10.1093/bioinformatics/bty420

[5] Scornavacca, C., Weller, M.: Treewidth-based algorithms for the small parsimony problem on networks. Algorithms for Molecular Biology **17**(1), 15 (2022). doi:10.1186/s13015-022-00216-w

[6] Katoch, S., Chauhan, S.S., Kumar, V.: A review on genetic algorithm: past, present, and future. Multimed. Tools Appl. **80**(5), 8091–8126 (2021). doi:10.1007/s11042-020-10139-6

[7] Miklós, I., Paige, T.B., Ligeti, P.: Efficient sampling of transpositions and inverted transpositions for bayesian MCMC. In: Algorithms in Bioinformatics, pp. 174–185. Springer, Berlin, Germany (2006). doi:10.1007/11851561_17

[8] Neumann, J., Lin, Y.T., Mallela, A., Miller, E.F., Colvin, J., Duprat, A.T., Chen, Y., Hlavacek, W.S., Posner, R.G.: Implementation of a practical Markov chain Monte Carlo sampling algorithm in PyBioNetFit. Bioinformatics **38**(6), 1770–1772 (2022). doi:10.1093/bioinformatics/btac004

[9] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York, NY, USA (1999). https://link.springer.com/book/10.1007/978-1-4612-0515-9

[10] Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science, Walthm, MA, USA (2006)

[11] Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco, CA (2003). doi:10.1016/B978-1-55860-890-0.X5000-2

[12] Dechter, R.: Reasoning with Probabilistic and Deterministic Graphical Models. Springer, Cham, Switzerland (2019). doi:10.1007/978-3-031-01583-0

[13] Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations i. upper bounds. Information and Computation **208**(3), 259–275 (2010). doi:10.1016/j.ic.2009.03.008

[14] Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Cham, Switzerland (2015).

https://link.springer.com/book/10.1007/978-3-319-21275-3

[15] Bodini, O., Ponty, Y.: Multi-dimensional Boltzmann sampling of languages. In: Proceedings of the 21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA'10), pp. 49–64. DMTCS Proceedings, Vienna, Austria (2010)

[16] Hammer, S., Wang, W., Will, S., Ponty, Y.: Fixed-parameter tractable sampling for RNA design with multiple target structures. BMC Bioinf. **20**(1), 1–13 (2019). doi:10.1186/s12859-019-2784-7

[17] Yao, H.-T., Waldispühl, J., Ponty, Y., Will, S.: Taming disruptive base pairs to reconcile positive and negative structural design of rna. In: Research in Computational Molecular Biology - 25th Annual International Conference, RECOMB 2021. Lecture Notes in Computer Science. Springer, Padova (2021)

[18] Marchand, B., Ponty, Y., Bulteau, L.: Tree diet: reducing the treewidth to unlock fpt algorithms in RNA bioinformatics. Algorithms Mol. Biol. **17**(1), 1–17 (2022). doi:10.1186/s13015-022-00213-z

[19] Marchand, B., Will, S., Berkemer, S.J., Bulteau, L., Ponty, Y.: Automated design of dynamic programming schemes for RNA folding with pseudoknots. In: Boucher, C., Rahmann, S. (eds.) 22nd International Workshop on Algorithms in Bioinformatics (WABI 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 242, pp. 7–1724. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022). doi:10.4230/LIPIcs.WABI.2022.7. https://drops.dagstuhl.de/opus/volltexte/2022/17041

[20] Will, S., Busch, A., Backofen, R.: Efficient sequence alignment with side-constraints by cluster tree elimination. Constraints **13**(1), 110–129 (2008). doi:10.1007/s10601-007-9032-x

[21] McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers **29**(6-7), 1105–1119 (1990). doi:10.1002/bip.360290621

[22] Ding, Y., Lawrence, C.E.: A statistical sampling algorithm for RNA secondary structure prediction. Nucleic acids research **31**, 7280–7301 (2003). doi:10.1093/nar/gkg938

[23] Hastings, W.K.: Monte carlo sampling methods using markov chains and their applications. Biometrika **57**(1), 97–109 (1970). doi:10.1093/biomet/57.1.97

[24] Roshan, U., Livesay, D.R.: Probalign: multiple sequence alignment using partition function posterior probabilities. Bioinformatics **22**(22), 2715–2721 (2006). doi:10.1093/bioinformatics/btl472

[25] Will, S., Joshi, T., Hofacker, I.L., Stadler, P.F., Backofen, R.: LocARNA-P: Accurate boundary prediction and improved detection of structural RNAs. RNA **18**(5), 900–14 (2012). doi:10.1261/rna.029041.111

[26] Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with Gecode. Schulte, Christian and Tack, Guido and Lagerkvist, Mikael **1** (2010)

[27] Allouche, D., Bessiere, C., Boizumault, P., de Givry, S., Gutierrez, P., Lee, J.H.M., Leung, K.L., Loudni, S., Métivier, J.-P., Schiex, T., Wu, Y.: Tractability-preserving transformations of global cost functions. Artif. Intell. **238**, 166–189 (2016). doi:10.1016/j.artint.2016.06.005

[28] Yao, H.-T., Ponty, Y., Will, S.: Developing complex RNA design applications in the Infrared framework, (2022). preprint; to be published. https://hal.science/hal-03711828

[29] Bodlaender, H.L., Koster, A.M.: Combinatorial optimization on graphs of bounded treewidth. The Computer Journal **51**(3), 255–269 (2008)

[30] Dechter, R.: Tractable Structures for Constraint Satisfaction Problems. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence, vol. 2, pp. 209–244. Elsevier, Walthm, MA, USA (2006). doi:10.1016/S1574-6526(06)80011-8

[31] Bachoore, E., Bodlaender, H.L.: Weighted treewidth algorithmic techniques and results. In: Tokuyama, T. (ed.) International Symposium on Algorithms and Computation (ISAAC 2007), pp. 893–903. Springer, Berlin, Heidelberg (2007). doi:10.1007/978-3-540-77120-3_77

[32] Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic Discrete Methods **8**(2), 277–284 (1987). doi:10.1137/0608024

[33] Tamaki, H.: Positive-instance driven dynamic programming for treewidth. Journal of Combinatorial Optimization **37**(4), 1283–1311 (2019)

[34] Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing, pp. 226–234 (1993)

[35] Doolittle, W.F.: Phylogenetic classification and the universal tree. Science **284**(5423), 2124–2128 (1999). doi:10.1126/science.284.5423.2124

[36] Fitch, W.M.: Toward defining the course of evolution: Minimum change for a specific tree topology. Syst. Biol. **20**(4), 406–416 (1971). doi:10.1093/sysbio/20.4.406

[37] Sankoff, D., Rousseau, P.: Locating the vertices of a steiner tree in an arbitrary metric space. Math. Program. **9**(1), 240–246 (1975). doi:10.1007/BF01681346

[38] Darty, K., Denise, A., Ponty, Y.: VARNA: Interactive drawing and editing of the RNA secondary structure. Bioinformatics **25**(15), 1974 (2009)

[39] Taneda, A.: Multi-objective optimization for RNA design with multiple target secondary structures. BMC Bioinf. **16**, 280 (2015). doi:10.1186/s12859-015-0706-x. 26335276

[40] Mathews, D.H., Sabina, J., Zuker, M., Turner, D.H.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. Journal of Molecular Biology **288**(5), 911–940 (1999). doi:10.1006/jmbi.1999.2700

[41] Turner, D.H., Mathews, D.H.: NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. Nucleic acids research **38**(suppl_1), 280–282 (2010)

[42] 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**(3), 443–453 (1970). doi:10.1016/0022-2836(70)90057-4

[43] Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**(3), 705–708 (1982). doi:10.1016/0022-2836(82)90398-9. 7166760

[44] Zhou, Y., Ponty, Y., Vialette, S., Waldispuhl, J., Zhang, Y., Denise, A.: Flexible RNA design under structure and sequence constraints using formal languages. In: BCB'13: Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics, pp. 229–238. Association for Computing Machinery, New York, NY, USA (2013). doi:10.1145/2506583.2506623

[45] Zhang, Y., Ponty, Y., Blanchette, M., Lécuyer, E., Waldispühl, J.: SPARCS: a web server to analyze (un)structured regions in coding RNA sequences. Nucleic Acids Res. **41**(W1), 480–485

(2013). doi:10.1093/nar/gkt461

[46] Reinharz, V., Ponty, Y., Waldispühl, J.: A weighted sampling algorithm for the design of RNA sequences with targeted secondary structure and nucleotide distribution. Bioinformatics **29**(13), 308–315 (2013). doi:10.1093/bioinformatics/btt217. https://academic.oup.com/bioinformatics/article-pdf/29/13/i308/18534655/btt217.pdf

[47] Boury, T., Ponty, Y., Reinharz, V.: Automatic exploration of the natural variability of RNA non-canonical geometric patterns with a parameterized sampling technique. In: 23nd International Workshop on Algorithms in Bioinformatics (WABI 2023). Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://hal.science/hal-04094288

[48] Ponty, Y., Hammer, S., Yao, H.-T., Will, S.: Advanced design of structural RNAs using RNARedPrint. RNA Bioinformatics, p. . Springer, ??? (2020)

[49] Hammer, S., Tschiatschek, B., Flamm, C., Hofacker, I.L., Findeiß, S.: RNAblueprint: flexible multiple target nucleic acid sequence design. Bioinformatics **33**(18), 2850–2858 (2017). doi:10.1093/bioinformatics/btx263

[50] Gambette, P., Gunawan, A.D., Labarre, A., Vialette, S., Zhang, L.: Solving the tree containment problem in linear time for nearly stable phylogenetic networks. Discrete Applied Mathematics **246**, 62–79 (2018)

[51] Eén, N., Sörensson, N.: An extensible SAT-solver. In: Theory and Applications of Satisfiability Testing, pp. 502–518. Springer, Berlin, Germany (2004). doi:10.1007/978-3-540-24605-3_37

[52] Cplex, I.I.: V12. 1: User's manual for CPLEX. International Business Machines Corporation **46**(53), 157 (2009)

[53] Giegerich, R., Meyer, C., Steffen, P.: A discipline of dynamic programming over sequence data. Sci. Comput. Programming **51**(3), 215–263 (2004). doi:10.1016/j.scico.2003.12.005

[54] Sauthoff, G., Möhl, M., Janssen, S., Giegerich, R.: Bellman's GAP—a language and compiler for dynamic programming in sequence analysis. Bioinformatics **29**(5), 551–560 (2013). doi:10.1093/bioinformatics/btt022

[55] 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**(12), 1124–1136 (2013). doi:10.1002/bip.22337

[56] Riechert, M., Höner zu Siederdissen, C., Stadler, P.F.: Algebraic dynamic programming for multiple context-free grammars. Theoret. Comput. Sci. **639**, 91–109 (2016). doi:10.1016/j.tcs.2016.05.032

[57] Michálik, J., Touzet, H., Ponty, Y.: Efficient approximations of RNA kinetics landscape using non-redundant sampling. Bioinformatics (Oxford, England) **33**, 283–292 (2017). doi:10.1093/bioinformatics/btx269

Additional Files

Additional file 1 — Correctness proofs

The file contains proofs of Propositions 1, 2 and 3, which establish the correctness of the optimization and sampling algorithms.