S SUPPLEMENT

S.1 Graph Kernel

S.1.1 Graph Definitions and Notation A graph $G = (V, E)$ consists of two sets V and E. The notation $V(G)$ and $E(G)$ is used when G is not the only graph considered. The elements of V are called *vertices* and the elements of E are called *edges*. Each edge has a set of two elements in V associated with it, which are called its *endpoints*, which we denote by concatenating the vertices variables, e.g. we represent the edge between the vertices u and v with uv. An edge is said to *join* its endpoints. A vertex v is *adjacent* to a vertex u if they are joined by an edge. An edge and a vertex on that edge are called *incident*. The *degree* of a vertex is number of edges incident to it. A *multi-edge* is a collection of two or more edges having identical endpoints. A *self-loop* is an edge that joins a single endpoint to itself. A *simple graph* is a graph that has no self-loops nor multi-edges. In this work we consider only simple graphs. A graph is *complete* if every pair of vertices is joined by an edge. A graph is *rooted* when we distinguish one of its vertices, called *root*; we denote a rooted graph G with root vertex v with G^v. A *walk* in a graph G is a sequence of vertices $W = v_0, v_1, \ldots, v_n$ such that for $j = 1, \ldots, n$, the vertices v_{j-1} and v_j are adjacent. The *length* of a walk is the number of edges (counting repetitions). A *path* is a walk such that no vertex is repeated, except at most the initial (v_0) and the final (v_n) vertex (in this case it is called a *cycle*). The *distance* between two vertices, denoted $\mathcal{D}(u, v)$, is the length of the shortest path between them. A graph is *connected* if between each pair of vertices there exist a walk. In this work we consider only connected graphs. We denote the class of simple connected graphs with $\mathcal{G}$. The *neighborhood* of a vertex v is the set of vertices that are adjacent to v and is indicated with $N(v)$. The *neighborhood* of radius r of a vertex v is the set of vertices at a distance less than or equal to r from v and is denoted by $N_r(v)$. In a graph G, the *induced-subgraph* on a set of vertices $W = \{w_1, \ldots, w_k\}$ is a graph that has W as its vertex set and it contains every edge of G whose endpoints are in W. A subgraph H is a *spanning* subgraph of a graph G if $V(H) = V(G)$. The *neighborhood subgraph* of radius r of vertex v is the subgraph induced by the neighborhood of radius r of v and is denoted by $\mathcal{N}_r^v$. A *labeled graph* is a graph whose vertices and/or edges are labeled, possibly with repetitions, using symbols from a finite alphabet. We denote the function that maps the vertex/edge to the label symbol as $\mathcal{L}$. Two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic*, which we denote by $G_1 \simeq G_2$, if there is a bijection $\phi : V_1 \to V_2$, such that for any two vertices $u, v \in V_1$, there is an edge uv if and only if there is an edge $\phi(u)\phi(v)$ in G_2. An isomorphism is a structure-preserving bijection. Two labeled graphs are isomorphic if there is an isomorphism that preserves also the label information, i.e. $\mathcal{L}(\phi(v)) = \mathcal{L}(v)$. An *isomorphism invariant* or *graph invariant* is a graph property that is identical for two isomorphic graphs (e.g. the number of vertices and/or edges). A *certificate for isomorphism* is an isomorphism invariant that is identical for two graphs if and only if they are isomorphic.

S.1.2 Kernel Definition and Notation Given a set X and a function $K : X \times X \to \mathbb{R}$, we say that K is a *kernel on* $X \times X$ if K is symmetric, i.e. if for any x and $y \in X$ $K(x, y) = K(y, x)$, and if K is *positive-semidefinite*, i.e. if for any $N \geq 1$ and any $x_1, \ldots, x_N \in X$, the matrix K defined by $K_{ij} = K(x_i, x_j)$ is positive-semidefinite, that is $\sum_{ij} c_i c_j K_{ij} \geq 0$ for all $c_1, \ldots, c_N \in \mathbb{R}$ or equivalently if all its eigenvalues are nonnegative. It is easy to see that if each $x \in X$ can be represented as $\phi(x) = \{\phi_n(x)\}_{n \geq 1}$ such that K is the ordinary l_2 dot product $K(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_n \phi_n(x)\phi_n(y)$ then K is a kernel. The converse is also true under reasonable assumptions (which are almost always verified) on X and K, that is, a given kernel K can be represented as $K(x, y) = \langle \phi(x), \phi(y) \rangle$ for some choice of ϕ. In particular it holds for any kernel K over $X \times X$ where X is a countable set. The vector space induced by ϕ is called the *feature space*. Note that it follows from the definition of positive-semidefinite that the *zero-extension* of a kernel is a valid kernel, that is, if $S \subseteq X$ and K is a kernel on $S \times S$ then K may be extended to be a kernel on $X \times X$ by defining $K(x, y) = 0$ if x or y is not in S. It is easy to show that kernels are closed under summation, i.e. a sum of kernels is a valid kernel.

Let now $x \in X$ be a *composite structure* such that we can define $x_1, \ldots, x_D$ as its parts[5]. Each part is such that $x_d \in X_d$ for $d = 1, \ldots, D$ with $D \geq 1$ where each X_d is a countable set. Let R be the relation defined on the set $X_1 \times \ldots \times X_D \times X$, such that $R(x_1, \ldots, x_D, x)$ is true iff $x_1, \ldots, x_D$ are the parts of x. We denote with $R^{-1}(x)$ the inverse relation that yields the parts of x, that is $R^{-1}(x) = \{x_1, \ldots, x_D : R(x_1, \ldots, x_D, x)\}$. In Haussler (1999) it is demonstrated that, if there exist a kernel K_d over $X_d \times X_d$ for each $d = 1, \ldots, D$, and if two instances $x, y \in X$ can be decomposed in $x_1, \ldots, x_d$ and $y_1, \ldots, y_d$, then the following generalized convolution:

$$K(x, y) = \sum_{\substack{x_1, \ldots, x_m \in R^{-1}(x) \\ y_1, \ldots, y_m \in R^{-1}(y)}} \prod_{m=1}^{M} K_m(x_m, y_m)$$

is a valid kernel called a *convolution* or *decomposition* kernel[6]. In words: a decomposition kernel is a sum (over all possible ways to decompose a structured instance) of the product of valid kernels over the parts of the instance.

S.1.3 The Neighborhood Subgraph Pairwise Distance Kernel Given the notation introduced in the previous sections, in the following we define the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) as an instance of a decomposition kernel.

We define the relation $R_{r,d}(A^v, B^u, G)$ between two rooted graphs A^v, B^u and a graph G to be true iff both A^v and B^u are in $\{\mathcal{N}_r^v : v \in V(G)\}$, where we require that A^v (B^u) be isomorphic to some $\mathcal{N}_r$ to verify the set inclusion, and that $\mathcal{D}(u, v) = d$. In words: the relation $R_{r,d}$ selects all pairs of neighborhood graphs of radius r whose roots are at distance d in a given graph G.

We define $\kappa_{r,d}$ over $\mathcal{G} \times \mathcal{G}$ as the decomposition kernel on the relation $R_{r,d}$, that is:

$$\kappa_{r,d}(G, G') = \sum_{\substack{A_v, B_u \in R_{r,d}^{-1}(G) \\ A'_{v'}, B'_{u'} \in R_{r,d}^{-1}(G')}} \delta(A_v, A'_{v'})\delta(B_u, B'_{u'})$$

[5] Note that the set of parts needs not be a partition for the composite structure, i.e. the parts may "overlap".

[6] To be precise, the valid kernel is the zero-extension of K to $X \times X$ since $R^{-1}(x)$ is not guaranteed to yield a non-empty set for all $x \in X$.

where the *exact matching kernel* $\delta(x, y)$ is 1 if $x \simeq y$ (i.e. if the graph x is isomorphic to y) and 0 otherwise. In words: $\kappa_{r,d}$ counts the number of identical pairs of neighboring graphs of radius r at distance d between two graphs.

The Neighborhood Subgraph Pairwise Distance Kernel is finally defined as:

$$K(G, G') = \sum_r \sum_d \kappa_{r,d}(G, G').$$

For efficiency reasons however, in this work we consider the zero-extension of K obtained by imposing an upper bound on the radius and the distance parameter: $K_{r^*,d^*}(G, G') = \sum_{r=0}^{r^*} \sum_{d=0}^{d^*} \kappa_{r,d}(G, G')$, that is, we are limiting NSPDK to the sum of the $\kappa_{r,d}$ kernels for all increasing values of the radius (distance) parameter up to a maximum given value r^* (d^*). Furthermore we consider a normalized version of $\kappa_{r,d}$, that is: $\hat{\kappa}_{r,d}(G, G') = \frac{\kappa_{r,d}(G,G')}{\sqrt{\kappa_{r,d}(G,G)\kappa_{r,d}(G',G')}}$, to ensure that relations of all orders are equally weighted regardless of the size of the induced part sets[7].

Finally, it is easy to show that the Neighborhood Subgraph Pairwise Distance Kernel is a valid kernel as: 1) it is built as a decomposition kernel over the countable space of all pairs of neighborhood subgraphs of graphs of finite size; 2) the kernel over parts (the exact matching kernel) is a valid kernel; 3) the zero-extension to bounded values for the radius and distance parameters preserves the kernel property; and 4) so does the normalization step.

S.1.4 Graph Invariant The NSPDK includes an exact matching kernel over two graphs which is equivalent to solving the graph isomorphism problem (ISO). Since the existence of (deterministic) polynomial algorithms for ISO is still an open problem, we have to resort to one of two strategies: 1) limit the class of graphs under consideration and solve ISO exactly; or 2) give an approximate (fast) solution of ISO on general graphs. Here we opt for the latter solution since we are mainly concerned with application domains where the number of graphs to be processed are in the range of tens to hundreds of thousands and application specific pre-processing might alter the class of the input graphs (making them non-outer planar for example).

In this work we implement the exact matching kernel $\delta(G_h, G'_{h'})$ in two steps: 1) we compute a fast graph invariant encoding for G_h and $G'_{h'}$ via a label function $\mathcal{L}^g : \mathcal{G}_h \to \Sigma^*$, where $\mathcal{G}_h$ is the set of rooted graphs and Σ^* is the set of strings over a finite alphabet Σ; 2) we make use of a hash function $H : \Sigma^* \to \mathbb{N}$ to confront $H(\mathcal{L}^g(G_h))$ and $H(\mathcal{L}^g(G'_{h'}))$. In words: we produce an efficient string encoding of graphs from which we obtain a unique identifier via a hashing function from strings to natural numbers. In this way the isomorphism test between two graphs is reduced to a fast numerical identity test. Note that we cannot hope to exhibit an efficient certificate for isomorphism in this way, but only an efficient graph invariant at most, i.e. there will be cases where two non-isomorphic graphs are assigned the same identifier.

The graph encoding $\mathcal{L}^g(G_h)$ that we propose is best described by introducing new label functions for vertices and edges, denoted $\mathcal{L}^n$ and $\mathcal{L}^e$ respectively. $\mathcal{L}^n(v)$ assigns to vertex v the concatenation of the lexicographically sorted listed of distance-label pairs $\langle \mathcal{D}(v, u), \mathcal{L}(u) \rangle$ for all $u \in G_h$. Since G_h is a rooted graph we can exploit the knowledge about the identity of the root vertex h and include, for each vertex v, the additional information of the distance from the root node $\mathcal{D}(v, h)$. $\mathcal{L}^e(uv)$ assigns to edge uv the label $\langle \mathcal{L}^n(u), \mathcal{L}^n(v), \mathcal{L}(uv) \rangle$. $\mathcal{L}^g(G_h)$ assigns to the rooted graph G_h the concatenation of the lexicographically sorted list of $\mathcal{L}^e(uv)$ for all $uv \in E(G_h)$. In words: we relabel each vertex with a string that encodes the vertex distance from all other labeled vertices (plus the distance from the root vertex); the graph encoding is obtained as the sorted edge list, where each edge is annotated with the endpoints' new labels.

We finally resort to a Merkle-Damgård construction based hashing function for variable-length data to map the graph encoding string to a 32-bit integer.

S.1.5 Kernel Algorithmic Complexity The time complexity of the NSPDK depends on two key procedures: 1) the extraction of all pairs of neighborhood graphs $\mathcal{N}_r^v$ at distance $d = 0, \ldots, d^*$, and 2) the computation of the graph invariant for those subgraphs. The first procedure can be efficiently implemented by factoring it into a) the extraction of $\mathcal{N}_r^v$ for all $v \in V(G)$ and b) the computation of distances between pairs of vertices whose pairwise distance is less than d^*. For this latter step we can repeat a breadth-first (BF) visit up to distance d^* for each vertex in $O(|V(G)||E(G)|)$. Note that, on graphs with bounded (low) degree, the complexity is more realistically modeled as a linear function of $|V(G)|$ since a small d^* implies, in practice, that each bounded BF visit can be performed in constant time. The complexity of point a) is linear in the number of edges in the neighborhood (constant in practice for small r). Finally, the complexity of point 2) (the computation of the graph invariant for neighborhood graphs) can be analyzed in terms of i) the computation of the string encoding $\mathcal{L}^g(G_h)$ and ii) the computation of the hash function $H(\mathcal{L}^g(G_h))$. Part i) is dominated by the computation of all pairwise distances in $O(|V(G_h)||E(G_h)|)$ and the sorting of the relabeled edges, which has complexity $O(|V(G_h)||E(G_h)| \log |E(G_h)|)$ since edges are relabeled with strings containing the distance information of the endpoints from all other vertices. The hash function complexity (part ii)) is linear in the size of the string. We conclude that the overall complexity $O(|V(G)||V(G_h)||E(G_h)| \log |E(G_h)|)$ is dominated by the repeated computation of the graph invariant for each vertex of the graph. Since this is a constant time procedure for small values of d^* and r^*, we conclude that the NSPDK complexity is in practice linear in the size of the graph.

Note finally that, to reduce space complexity, we do not manage the hash collisions, as this would force the algorithm to keep in memory all the encoding key - hashed value pairs.

S.2 Efficient Neighborhood graph extraction using Locality Sensitive Hashing

As datasets size increases, algorithms that directly make use of pairwise distance or similarity information become infeasible as they inevitably exhibit a quadratic complexity. The key idea then is to formulate the clustering problem in terms of approximate nearest neighbors queries which can be answered efficiently (sub-linearly). That is, given a set of n instances $P = \{p_1, \ldots, p_n\}$ in a metric space X with a distance function d, a neighborhood query is a

[7] As the number of neighborhood graphs grows exponentially with the radius, large (infrequent) subgraphs tend to dominate the kernel value with negative effects on the generalization performance of predictive systems.

procedure that returns the instance in P closest to a query instance $q \in X$. The *nearest neighbor search problem* is formulated as a dataset pre-processing that allows nearest neighbors queries to be answered efficiently. The key idea is to relax the requirements, ask for $\epsilon-$approximate nearest neighbor queries, and use *locality-sensitive hashing* techniques. The $\epsilon-$approximate nearest neighbor query returns an instance p for a given query q such that $\forall p' \in P, d(p,q) \leq (1+\epsilon)d(p',q)$. A locality-sensitive hash function is a hash function such that the probability of collision is higher for objects that are close to each other than for those that are far apart. As locality-sensitive hash function we choose the min-hash function Broder (1997) as it approximates the natural similarity notion defined by the Jaccard index. However these techniques require instances to be represented as sparse *binary* vectors rather than sparse *real* vectors. We therefore binarize all instances from $\mathbb{R}^m \mapsto \{0,1\}^m$ setting to 1 all non-null components. Let $x, z \in \{0,1\}^m$ be two instances; the Jaccard similarity between the two instances is defined as $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, i.e., the ratio of the number of features that the instances have in common over the overall number of features. We build a min-hash function starting from a set of random hash functions $f_i : \mathbb{N} \mapsto \mathbb{N}$, i.e., functions that map integers randomly (but consistently) to integers; in our case the domain/co-domain represent feature indicators. These functions must be independent and satisfy: $\forall x_j \neq x_k, f_i(x_j) \neq f_i(x_k)$, and $\forall x_j \neq x_k, P(f_i(x_j) \leq f_i(x_k)) = \frac{1}{2}$. The min-hash function derived from f_i is defined as $h_i(x) = \arg\min_{x_j \in x} f_i(x_j)$, i.e., the min-hash returns the first feature indicator under a random permutation of the features order. A rather surprising (and useful) fact is that a min-hash collision is an unbiased estimator of the Jaccard similarity:

$$P(h_i(x) = h_i(z)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$$

i.e. the probability to select as the minimum feature indicator a non-null feature that belongs to both x and z is exactly the fraction of features that x and z have in common over the total number of non-null features of x and z. In order to decrease the (high) variance of this estimate one can take N independent min-hash functions and compute the number n of times that $h_i(x) = h_i(z)$. The estimated value n/N is the average of N different 0-1 random variables, which evaluates to one when $h_i(x) = h_i(z)$ and zero in all other cases. The average of these unbiased estimators of $s(x,z)$ is also an unbiased estimator, with an expected error bounded by $O(1/\sqrt{N})$ [8], or, equivalently, for any constant $\gamma > 0$ we can compute a constant $N = O(1/\gamma^2)$ such that the expected error of the estimate is at most γ. For example, with 400 hash functions the estimate of $s(x,z)$ would have an expected error $\leq .05$.

We collect the results of the entire set of min-hash functions in an *instance sketch* as the tuple $\langle h_1(x), \ldots, h_N(x) \rangle$. In order to obtain an efficient neighbor search procedure, we build an inverse index that returns all instances with the same min-hash value in $O(1)$. More precisely, given the $i-$th hash function and a value $\bar{h} = h_i(x)$, we collect the set of instances $Z_i(\bar{h}) = \{z \in P : h_i(z) = \bar{h}\}$. The approximate neighbourhood Z of an instance x is then induced from the multi-set $Z = \{Z_i\}_{i=1}^N$. Note that when γ (or equivalently N) is fixed, the complexity to build a single signature is constant and therefore the complexity for building the index is linear in the size of the dataset. To improve the quality of the returned neighbors we consider only the most frequent elements in Z and sort them according to their `NSPDK` similarity to x. The $k-$neighborhood $N_k(x)$ is finally the set of the k-closest elements. If the size of Z is small and independent of the dataset size $|P|$, these steps can be performed in constant time.

[8] The relation can be obtained by standard Chernoff bounds for sums of 0-1 random variables.

S.3 `BlockClust` Parameters Optimization

In order to assess the best parameter settings for each tool used in the pipeline, attribute discretization and selection, we applied `BlockClust` on a specific data set with different parameter settings and different attribute combinations. We call each attribute combination as a *configuration*. For each configuration and parameter setting we measure the performance of the clustering and chose the best configuration and parameter settings for the usage of the `BlockClust`. In order to measure performance for the known ncRNA families we had to look at the annotations, hence it is supervised learning.

S.3.1 Mapping. We removed adapters and linkers from all raw reads using `fastx-clipper`[9] and applied `segemehl` Hoffmann *et al.* (2009) to align the clipped reads to the human genome (we reported only best scoring hits and required a minimum mapping accuracy of 85%). `segemehl` can efficiently deal mismatches and indels, it is independent of the underlying sequencing platforms and handles reads of different lengths. To correct for multiple mappings, we normalized the read counts n of each tag by the number of mappings k in the reference genome. Thus, the *tag expressions* n/k are assigned to each tag.

We relied on supervised learning to set up `BlockClust` and to find optimal attribute combinations and the best parameter values for the external tools run by our pipeline (`blockbuster` and `NSPDK`). Among others, this comprises the following major steps: partitioning of labeled input data; attribute generation, encoding, discretization and selection; parameter optimization; clustering.

S.3.2 Random partitioning. We randomly partitioned each benchmark dataset into three sub sets: train ($\sim 35\%$), validation ($\sim 35\%$), and test set ($\sim 30\%$). The training and validation set is used to benchmark our approach, e.g. during attribute selection and parameter optimization. The independent test set is used to obtain a final performance estimate on the fully trained model. Note, that these random splits were done on the level of reads to ensure unbiased learning. They are independent of any subsequent `blockbuster` or `BlockClust` call, no blocks or block groups have been assigned yet. Thus, instead of randomly distributing reads among train, validation and test set, we relied on the concept of "*read stretches*" for an unbiased partitioning of the data. We define a "*read stretches*" as a series of sorted reads separated by a maximum distance d. With the exception of a few ribosomal RNAs, most of the classic short ncRNAs are not longer than 500 nt. Thus, we set d to 500 and split the data on the level of read stretches, ensuring that most of the subsequently computed read profiles cover full ncRNA genes.

[9] `http://hannonlab.cshl.edu/fastx_toolkit`

Table S1. Overview on the samples used to benchmark `BlockClust`. Last three columns correspond to the number of blockgroups found by `blockbuster`, number of block groups filtered by length and expression level annotation and number of block groups retain at the end after intersecting with annotation.

GEO accession	Organism	Tissue/Cell line	Seq. machine	#Reads	#Tags	#BGs	#Filtered	#Known
GSE16368/GSM450239	Human	H1 cell line	Illumina GAII	16830686	618590	2404	755	629
GSE31069/GSM769509	Human	MCF-7 cytoplasmic	Illumina GAII	15493265	571470	2503	687	508
GSE31069/GSM769510	Human	MCF-7 total cell	Illumina GAII	14670735	519579	2168	586	474
GSE31069/GSM769511	Human	MCF-7 cytoplasmic	Illumina GAII	9237490	380461	1957	603	466
GSE31069/GSM769512	Human	MCF-7 total cell	Illumina GAII	8689337	320205	1770	552	458
GSE26545/GSM652847	Human	Cortex of brain	Illumina GAII	8241330	416757	1253	414	378
GSE26545/GSM652851	Human	Gyrus of the brain	Illumina GAII	6486498	490336	1232	464	410
GSE18012/GSM450597	Human	Gyrus of the brain (2 days)	Illumina GAII	6754470	231368	752	215	190
GSE18012/GSM450598	Human	Gyrus of the brain (34 days)	Illumina GAII	7299034	343234	1019	321	296
GSE18012/GSM450603	Human	Gyrus of the brain (98 years)	Illumina GAII	5763414	184097	760	238	224
GSE18012/GSM450605	Human	Gyrus of the brain (14 years)	Illumina GAII	8538940	729571	1554	524	482
GSE31037/GSM768988	Human	Skin	Illumina GAIIx	15579483	616913	2678	880	729
GSE31037/GSM769007	Human	Skin	Illumina GAIIx	21217688	360220	2534	863	750
GSE26545/GSM652849	Chimp	Cortex of brain	Illumina GAII	7776308	387720	1254	458	290
GSE26545/GSM652853	Chimp	Gyrus of the brain	Illumina GAII	7240683	512620	1272	413	247
GSE36639/GSM897819	Mouse	NIH 3T12 cells	Illumina GA	1843676	99306	625	247	223
GSE36639/GSM897820	Mouse	NIH 3T12 cells	Illumina GA	5694227	182587	1246	429	350
GSE36639/GSM897821	Mouse	NIH 3T12 cells	Illumina GA	8526798	213149	1520	461	370
GSE36639/GSM897822	Mouse	NIH 3T12 cells	Illumina GA	5682600	200515	1263	437	349
GSE36639/GSM897823	Mouse	NIH 3T12 cells	Illumina GA	6521133	232861	1264	449	336
GSE38702/GSM947965	Mouse	Testis	Illumina HiSeq 2K	23783785	2592368	8530	3144	133
GSE38702/GSM947966	Mouse	Uterus	Illumina GA	15876066	526206	1083	472	339
GSE11624/GSM272651	Fly	S2 & KC cells	Illumina GA	746043	163952	790	186	177
GSE11624/GSM286601	Fly	male heads	Illumina GA	621971	120124	308	161	159
GSE11624/GSM286602	Fly	male body	Illumina GA	980097	287958	1008	383	367
GSE40015/GSM983642	Fly	Female body	Illumina GA	1943622	1067609	3760	475	313
GSE40015/GSM983641	Fly	Female body	Illumina GA	487729	378940	1236	312	280
GSE17153/GSM427301	Worm	One cell embryo	Illumina GA II	3742851	427651	722	73	61
GSE17153/GSM427346	Worm	Mixed embryos	Illumina GA II	2965597	200072	658	246	229
GSE25738/GSM632205	Plant	seedlings	Illumina GA	11772773	1686361	10672	3514	364
GSE25738/GSM632207	Plant	seedlings	Illumina GA	11955547	2019497	7901	2265	220
GSE36934/GSM906549	Plant	leaves	Illumina GA IIx	6946527	1562959	12209	3813	260

After partitioning the read stretches to train,test and validaiton sets, we compute block groups using `blockbuster` on each set individually. A block group generalizes the expression profile of a ncRNA.

S.3.3 Annotation. We assigned a specific ncRNA class label to each block group using ncRNA annotation from different sources. We considered all human ncRNAs from the Rfam v10.1 (Gardner *et al.*, 2011) and Ensembl release-72 (Flicek *et al.*, 2012) databases. In addition, we downloaded miRNAs from miRBase v19 (Griffiths-Jones *et al.*, 2006), tRNAs from gtRNAdb (Chan and Lowe, 2009). See Table 2 for details.

For a reliable annotation, we filtered for block groups consisting of $\geq$ 2 blocks, a minimum expression of 50, a length between 50 and 200 nt, and a reciprocal overlap of at least 70% for each block group and its associated ncRNA. Block groups overlapping more than one ncRNA are likely to exhibit blurred read profiles and have been discarded. In line, if multiple block groups are found at a single ncRNA, we kept the block group with the largest overlap and ignored all others. Furthermore we tested for overlap of block groups mRNAs or pseudogenes from Ensembl database release-72 (Flicek *et al.*, 2012). We discarded the block groups with some significant overlap with mRNAs, as we consider reliable ncRNAs only.

After annotating the block groups with known ncRNAs we combined all train data sets of the 4 libraries together to get a single train set. Analogously done for validation and test data sets.

S.4 Attribute selection.

In order to identify the characteristic attributes of block groups we analyzed different sets of attribute: 5 were specific to block groups, 5 modeled blocks, and 2 were intended to capture the relation between blocks, see Supplementary Table S3.

As characteristic attributes of block groups we analyzed entropies of tag starts, tag ends, tag lengths. We define *entropy of tag starts* as follows: let q_i denote the fraction of tags in a given block group starting at position i. The *entropy of tag starts* is then defined as $-\sum_i q_i \log_2 q_i$. Analogously, we defined the *entropy of tag ends* and *entropy of tag lengths*. In addition to these entropies, *median*

Table S2. Overview on the ncRNA classes and annotation sources used to develop and benchmark `BlockClust`. All numbers refer to version hg19 of the human genome.. Database versions are as follows: Ensembl v72, Rfam v11.0, miRBase v20.

ncRNA family	Database	No. of ncRNAs
tRNA	gtRNADB, Rfam, Ensembl	625+904+22
miRNA	miRBase, Rfam, Ensembl	1871+1232+3215
snoRNA C/D box	Rfam, Ensembl	511+748
snoRNA H/ACA box	Rfam, Ensembl	440+312
rRNA	Rfam, Ensembl	608+508
snRNA	Rfam, Ensembl	2023+1404
Y_RNA	Rfam, Ensembl	893+821

of tag expressions and *tag expressions in first quantile* reveal the distribution of expression levels of tags within the block group. For blocks also we computed the *entropy of tag lengths* and in addition *entropy of the tag expressions*. In block level the *number of multi mapped tags* is an important attribute to consider. For miRNAs and C/D box snoRNAs it is too low compared to tRNAs (see Supplementary Figure S1. This attribute is related to *tag expressions in first quantile* of block group. With increasing number of multi mapped tags, the tag expressions divides by number of times it mapped (see section S.3.1). Hence low expression in first quantile for tRNA, rRNA and snRNs. *Length of the block* it self and the *minimum tag length*, i.e., the shortest tag length within a block are also considered as block specific attributes. For each pair of two adjacent blocks, we computed their pairwise *block contiguity*. This single attribute represents the percentage overlap or percentage distance between two consecutive blocks, resp. We calculate total edge length between adjacent blocks, i.e. the number of bases spawned by both adjacent blocks including the gap between them. Then we calculate the distance or overlap between the those two blocks. To distinguish overlap and distance we use positive values for overlap and negative values for distance between blocks. We define *block contiguity* as the ratio of block overlap or distance to the total edge length. Let two adjacent blocks B_1 and B_2 with start positions s_{B_1}, s_{B_2} and end positions e_{B_1}, e_{B_2} respectively. The *block contiguity* for two adjacent blocks is defined as $(e_{B_1} - s_{B_2})/(e_{B_2} - s_{B_1})$.

For each block we computed median of tag expressions and take the difference of these medians for adjacent blocks as a attribute. Let a *block height* be the highest tag expression within a Supplementary Table S 3 gives an overview of selected attributes in learning phase.

Table S3. Attribute selection. Overview of the selected attributes for the graph encoding.

Category	Attribute
block group	entropy of tag starts
block group	entropy of tag ends
block group	entropy of tag lengths
block group	median of tag expressions
block group	tag expression levels in first quantile
block	number of multi mapped tags
block	entropy of tag lengths
block	entropy of tag expressions
block	minimum tag length
block	block length
pairs of blocks	contiguity
pairs of blocks	difference in median tag expressions

Table S4. Clustering performance of `BlockClust` on `Benchmark Data`. The AUC of block group similarities indicate that `BlockClust` is robust across these diverge data sets. Note that the training was done on human data sets and performs fairly well on fly, worm and plant.

GEO accession	miRNA		tRNA		CD-box		HACA-box		rRNA		snRNA		YRNA		Average	
	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC
GSE16368/GSM450239	226	0.899	208	0.843	95	0.719	14	0.803	38	0.836	14	0.679	31	0.754	629	0.835
GSE31069/GSM769509	170	0.926	218	0.774	29	0.827	7	0.866	52	0.776	18	0.596	13	0.592	508	0.819
GSE31069/GSM769510	164	0.899	190	0.816	67	0.772	12	0.813	24	0.884	5	0.501	11	0.639	474	0.835
GSE31069/GSM769511	134	0.925	222	0.778	33	0.795	0	0	47	0.766	19	0.545	10	0.559	466	0.806
GSE31069/GSM769512	148	0.907	186	0.822	77	0.779	7	0.754	25	0.797	5	0.652	8	0.841	458	0.839
GSE26545/GSM652847	166	0.888	127	0.702	43	0.675	3	0.719	2	0.991	2	0.698	35	0.667	378	0.779
GSE26545/GSM652851	164	0.905	154	0.702	39	0.639	4	0.785	3	0.862	12	0.800	34	0.679	410	0.780
GSE18012/GSM450597	146	0.850	22	0.628	14	0.590	1	1.000	1	1.000	1	1.000	5	0.910	190	0.809
GSE18012/GSM450598	178	0.916	78	0.776	19	0.641	2	0.918	2	0.881	1	1.000	16	0.729	296	0.851
GSE18012/GSM450603	157	0.899	51	0.744	7	0.767	1	1.000	2	0.898	2	0.990	4	0.976	224	0.862
GSE18012/GSM450605	189	0.911	150	0.714	46	0.630	9	0.727	3	0.690	40	0.839	44	0.705	482	0.793
GSE31037/GSM768988	182	0.932	243	0.748	117	0.830	42	0.911	89	0.768	41	0.659	10	0.609	729	0.813
GSE31037/GSM769007	207	0.905	245	0.774	128	0.829	40	0.892	76	0.769	33	0.645	16	0.629	750	0.817
Human	2231	0.905	2094	0.770	714	0.759	142	0.859	364	0.789	193	0.690	237	0.693	5994	0.817
GSE26545/GSM652849	149	0.951	130	0.723	6	0.859	0	0	0	0	1	1.000	4	0.734	290	0.844
GSE26545/GSM652853	145	0.931	92	0.695	5	0.773	0	0	0	0	2	0.989	3	0.784	247	0.839
Chimp	294	0.941	222	0.711	11	0.820	0	0	0	0	3	0.993	7	0.755	537	0.842
GSE36639/GSM897819	133	0.951	90	0.699	0	0	0	0	0	0	0	0	0	0	223	0.849
GSE36639/GSM897820	153	0.964	144	0.697	0	0	0	0	53	0.966	0	0	0	0	350	0.854
GSE36639/GSM897821	162	0.962	149	0.716	0	0	0	0	59	0.948	0	0	0	0	370	0.861
GSE36639/GSM897822	146	0.973	150	0.702	0	0	0	0	53	0.926	0	0	0	0	349	0.849
GSE36639/GSM897823	131	0.940	142	0.700	0	0	0	0	63	0.879	0	0	0	0	336	0.827
GSE38702/GSM947965	56	0.865	77	0.752	0	0	0	0	0	0	0	0	0	0	133	0.800
GSE38702/GSM947966	156	0.932	133	0.762	0	0	0	0	50	0.945	0	0	0	0	339	0.868
Mouse	937	0.949	885	0.716	0	0	0	0	278	0.931	0	0	0	0	2100	0.848
GSE11624/GSM272651	48	0.983	124	0.903	0	0	0	0	0	0	5	0.649	0	0	177	0.917
GSE11624/GSM286601	69	0.970	90	0.787	0	0	0	0	0	0	0	0	0	0	159	0.866
GSE11624/GSM286602	65	0.992	193	0.754	0	0	0	0	98	0.986	11	0.742	0	0	367	0.858
GSE40015/GSM983641	64	0.967	231	0.937	4	0.909	0	0	0	0	14	0.639	0	0	313	0.929
GSE40015/GSM983642	59	0.991	213	0.960	2	0.996	0	0	0	0	6	0.862	0	0	280	0.965
Fly	305	0.980	851	0.880	6	0.938	0	0	98	0.986	36	0.709	0	0	1296	0.907
GSE17153/GSM427301	11	0.930	15	0.701	0	0	0	0	15	0.982	5	0.632	0	0	61	0.801
GSE17153/GSM427346	32	0.982	142	0.776	0	0	0	0	15	0.986	0	0	0	0	229	0.768
Worm	43	0.969	157	0.769	0	0	0	0	30	0.984	5	0.632	0	0	290	0.775
GSE25738/GSM632205	20	0.892	341	0.875	0	0	0	0	0	0	1	1.000	0	0	364	0.876
GSE25738/GSM632207	17	0.886	201	0.910	0	0	0	0	0	0	2	0.844	0	0	220	0.907
GSE36934/GSM906549	22	0.885	237	0.884	0	0	0	0	0	0	0	0	0	0	260	0.885
Plant	59	0.888	779	0.887	0	0	0	0	0	0	3	0.896	0	0	844	0.887
All	3869	0.925	4988	0.795	731	0.762	142	0.859	770	0.873	240	0.698	244	0.694	11061	0.839

Table S5. Classification performance of `BlockClust` on `Benchmark Data`. `BlockClust` was applied on a total of 32 independent data sets from 6 different species and several tissues and cell lines. Despite of some poor recall values for CD-box snoRNAs and tRNAs, `BlockClust` performed well on these diverse data sets.

GEO accession	miRNA			tRNA			snoRNA C/D-box		
fold	#	PPV	Recall	#	PPV	Recall	#	PPV	Recall
GSE16368/GSM450239	226	0.887	0.832	208	0.814	0.822	95	0.592	0.337
GSE31069/GSM769509	170	0.888	0.882	218	0.821	0.821	29	0.526	0.345
GSE31069/GSM769510	164	0.885	0.890	190	0.950	0.795	67	0.743	0.388
GSE31069/GSM769511	134	0.883	0.903	222	0.829	0.806	33	0.647	0.333
GSE31069/GSM769512	148	0.878	0.872	186	0.903	0.747	77	0.795	0.403
GSE26545/GSM652847	166	0.875	0.928	127	0.831	0.504	43	0.700	0.326
GSE26545/GSM652851	164	0.885	0.848	154	0.770	0.500	39	0.636	0.359
GSE18012/GSM450597	146	0.946	0.959	22	0.800	0.545	14	0.600	0.214
GSE18012/GSM450598	178	0.955	0.944	78	0.786	0.564	19	0.375	0.158
GSE18012/GSM450603	157	0.980	0.943	51	0.806	0.490	7	0.286	0.286
GSE18012/GSM450605	189	0.898	0.884	150	0.638	0.587	46	0.421	0.174
GSE31037/GSM768988	182	0.945	0.940	243	0.633	0.732	117	0.886	0.265
GSE31037/GSM769007	207	0.954	0.894	245	0.651	0.792	128	0.732	0.234
GSE26545/GSM652849	149	0.969	0.846	130	0.985	0.508	6	0.545	1.000
GSE26545/GSM652853	145	0.977	0.862	92	0.881	0.402	5	0.167	0.400
GSE36639/GSM897819	133	0.961	0.940	90	1.000	0.433	0	0.000	0.000
GSE36639/GSM897820	153	0.985	0.837	144	0.971	0.701	0	0.000	0.000
GSE36639/GSM897821	162	0.986	0.876	149	0.882	0.705	0	0.000	0.000
GSE36639/GSM897822	146	0.992	0.877	150	0.940	0.627	0	0.000	0.000
GSE36639/GSM897823	131	0.975	0.893	142	0.844	0.570	0	0.000	0.000
GSE38702/GSM947965	56	1.000	0.804	77	0.965	0.714		0.000	0.000
GSE38702/GSM947966	156	1.000	0.808	133	0.808	0.444	0	0.000	0.000
GSE11624/GSM272651	48	0.977	0.875	124	0.968	0.726	0	0.000	0.000
GSE11624/GSM286601	69	1.000	0.768	90	1.000	0.611	0	0.000	0.000
GSE11624/GSM286602	65	0.977	0.661	193	0.787	0.782	0	0.000	0.000
GSE40015/GSM983641	64	1.000	0.781	231	0.950	0.831	4	0.000	0.000
GSE40015/GSM983642	59	1.000	0.898	213	0.972	0.831	2	0.000	0.000
GSE17153/GSM427301	11	0.714	0.909	15	0.136	0.200	0	0.000	0.000
GSE17153/GSM427346	32	0.806	0.906	142	0.828	0.747	0	0.000	0.000
GSE25738/GSM632205	20	0.941	0.800	341	1.000	0.595	0	0.000	0.000
GSE25738/GSM632207	17	1.000	0.647	201	0.985	0.647	0	0.000	0.000
GSE36934/GSM906549	22	0.944	0.773	237	1.000	0.641	0	0.000	0.000

Table S6. Clustering performance of `BlockClust` on 10 random test splits of `Development Data` measured by the average per instance AUC ROC.

fold	miRNA		tRNA		C/D-box		H/ACA-box		rRNA		snRNA		YRNA		Average	
	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC	#	AUC
1	156	0.891	166	0.728	89	0.729	7	0.849	12	0.770	7	0.653	7	0.711	444	0.789
2	157	0.893	180	0.734	64	0.720	5	0.800	28	0.832	9	0.668	9	0.722	452	0.802
3	171	0.884	158	0.772	81	0.750	5	0.795	26	0.942	6	0.666	10	0.646	457	0.822
4	160	0.908	173	0.757	87	0.744	3	0.970	28	0.877	6	0.650	6	0.719	463	0.812
5	181	0.890	174	0.703	78	0.761	4	0.942	16	0.845	9	0.580	6	0.799	468	0.798
6	169	0.906	174	0.728	74	0.695	1	1.000	21	0.840	5	0.650	9	0.624	453	0.795
7	169	0.905	175	0.712	88	0.754	3	0.808	17	0.893	6	0.641	7	0.645	466	0.797
8	166	0.897	174	0.771	68	0.698	8	0.767	24	0.914	10	0.591	3	0.682	453	0.816
9	176	0.910	183	0.735	75	0.731	5	0.808	12	0.899	8	0.621	9	0.589	468	0.802
10	172	0.881	177	0.768	76	0.711	1	1.000	13	0.864	7	0.688	9	0.758	455	0.812

Table S7. Clustering performance of `BlockClust` on 10 random test splits of `Development Data` measured on `MCL` clustering precision.

fold	miRNA		tRNA		C/D-box		rRNA		Average	
	#clusters	Precision	#clusters	Precision	#clusters	Precision	#clusters	Precision	#clusters	Precision
1	7	0.833	20	0.827	10	0.614	1	1.000	38	0.777
2	8	0.830	22	0.794	6	0.662	2	1.000	38	0.792
3	10	0.824	18	0.861	8	0.898	2	1.000	38	0.852
4	7	0.881	20	0.896	13	0.680	2	1.000	42	0.832
5	12	0.846	19	0.819	10	0.597	1	0.750	42	0.772
6	8	0.910	24	0.834	9	0.608	1	1.000	42	0.804
7	7	0.822	18	0.831	8	0.739	1	1.000	36	0.813
8	11	0.851	11	0.875	3	0.628	2	0.961	27	0.845
9	20	0.841	8	0.833	3	0.711	2	0.833	34	0.823
10	13	0.912	12	0.808	4	0.761	1	1.000	20	0.853

Table S8. Classification performance of `BlockClust` on 10 random test splits of the `Development Data`. For miRNAs and tRNAs we achieved a mean precision about 0.9 and for C/D-box snoRNAs about 0.87. The recall for these three classes varies. The highest mean recall obtained for miRNAs about 0.89 and for tRNAs it is 0.8. C/D-box snoRNAs show lowest mean recall about 0.47.

	miRNA			tRNA			snoRNA C/D-box		
fold	#	PPV	Recall	#	PPV	Recall	#	PPV	Recall
1	156	0.887	0.853	166	0.873	0.873	89	0.868	0.371
2	157	0.937	0.847	180	0.869	0.778	64	0.931	0.422
3	171	0.874	0.930	158	0.881	0.892	81	0.911	0.506
4	160	0.898	0.881	173	0.944	0.775	87	0.796	0.449
5	181	0.883	0.917	174	0.879	0.839	78	0.836	0.654
6	169	0.852	0.917	174	0.848	0.897	74	0.879	0.392
7	169	0.936	0.776	175	0.888	0.771	88	0.971	0.375
8	166	0.896	0.9334	174	0.937	0.598	68	0.833	0.515
9	176	0.917	0.875	183	0.943	0.721	75	0.819	0.480
10	172	0.924	0.924	177	0.925	0.830	76	0.852	0.606

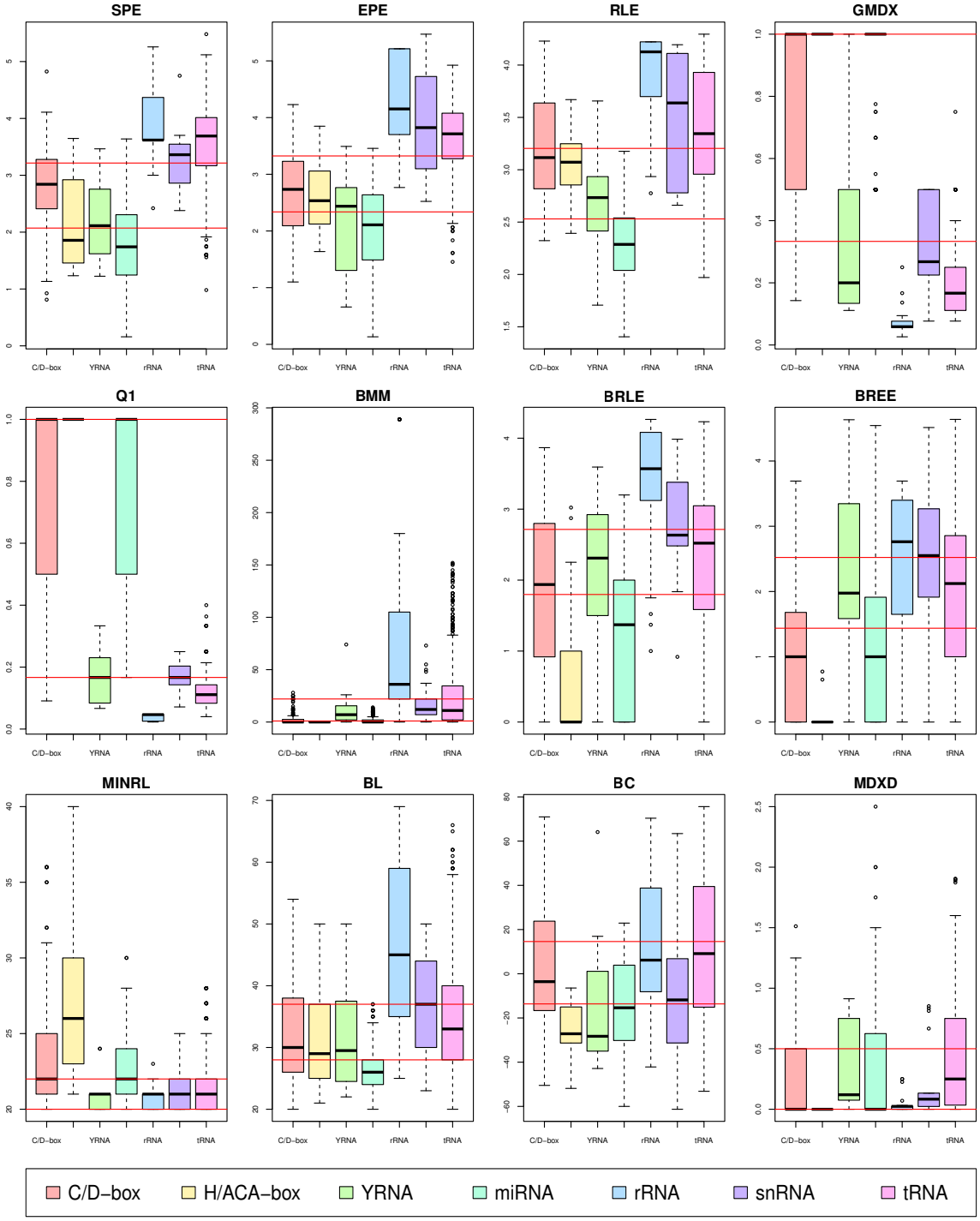

Fig. S1: Attribute value boxplots. The selected discretization levels are depicted with red lines.