BMC Genomics

RESEARCH Open Access

Identifying gene clusters by discovering common intervals in indeterminate strings

Daniel Doerr[1,2*], Jens Stoye[1,2], Sebastian Böcker[3], Katharina Jahn[1,2,4]

From Twelfth Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics
Cold Spring Harbor, NY, USA. 19-22 October 2014

Abstract

Background: Comparative analyses of chromosomal gene orders are successfully used to predict gene clusters in bacterial and fungal genomes. Present models for detecting sets of co-localized genes in chromosomal sequences require prior knowledge of gene family assignments of genes in the dataset of interest. These families are often computationally predicted on the basis of sequence similarity or higher order features of gene products. Errors introduced in this process amplify in subsequent gene order analyses and thus may deteriorate gene cluster prediction.

Results: In this work, we present a new dynamic model and efficient computational approaches for gene cluster prediction suitable in scenarios ranging from traditional gene family-based gene cluster prediction, via multiple conflicting gene family annotations, to gene family-free analysis, in which gene clusters are predicted solely on the basis of a pairwise similarity measure of the genes of different genomes. We evaluate our gene family-free model against a gene family-based model on a dataset of 93 bacterial genomes.

Conclusions: Our model is able to detect gene clusters that would be also detected with well-established gene family-based approaches. Moreover, we show that it is able to detect conserved regions which are missed by gene family-based methods due to wrong or deficient gene family assignments.

Background

Gene clusters are sets of functionally associated genes in prokaryotic and fungal genomes that are located close to each other over a longer period of evolutionary time, despite the genome undergoing significant rearrangements. Consequently, gene clusters may be found in several related species by means of comparative gene order analysis. Over the past years several such methods have been proposed and subsequently improved in their sensitivity. Initial gene cluster models considered only completely conserved genomic segments that retain gene order and orientation [1,2]. Later models still required gene clusters to be contiguous and complete but dropped the requirement for co-linearity [3-5]. The most powerful class of approaches can handle imperfect conservation of gene content by allowing to some extent either inserted [6-8] or both inserted and deleted genes [9-11].

All above methods require prior knowledge of homology relations between genes, using either a one-to-one mapping between the gene sets of different genomes [3,6,5], or a general partitioning into gene families [4,7-11]. In the latter, a genome is modeled as a set of sequences over the alphabet of gene families, where each sequence corresponds to a particular chromosome of the organism.

Most commonly, gene families are predicted computationally on the basis of sequence similarity. Various databases exist that offer information of precomputed gene families [12-14]. Furthermore, several software tools are freely available that allow for direct computation of gene family assignments in a dataset of interest [15-17]. Typically, these approaches assume that gene families naturally

* Correspondence: daniel.doerr@cebitec.uni-bielefeld.de
[1]Genome Informatics, Faculty of Technology, Bielefeld University, Bielefeld, Germany
Full list of author information is available at the end of the article

cluster into densely connected subgraphs in the gene similarity network. However, multi-domain proteins can have strong ties not only to their own family but also to other families they share a domain with. Some of these proteins may not be at all traceable back to a single gene family. While some recent approaches can deal with the ambiguities caused by multi-domain proteins [18,19], it is still a major challenge to define cut-offs in the clustering process that at the same time eliminate spurious edges and keep gene families at a reasonable granularity[20,21].

In this paper, we present a new dynamic model and efficient computational approaches for gene cluster prediction suitable in scenarios ranging from traditional gene family-based gene cluster prediction, via multiple conflicting gene family annotations, to gene family-free analysis, in which gene clusters are predicted solely on the basis of a pairwise similarity measure between the genes of different genomes. We do this by introducing the concept of *common intervals* to *indeterminate strings*, which are a class of strings that can have more than one character at every position. We then extend this concept to allow for a limited number of insertions and deletions. Furthermore, we present algorithms that solve related discovery problems of finding all *weak common intervals* and *approximate weak common intervals* in indeterminate strings. Finally, we propose a new method for gene family-free discovery of gene clusters based on (approximate) weak common intervals in indeterminate strings.

Methods

Definitions

Indeterminate strings, also known as *degenerate strings* are formally defined as [22]:

Definition 1 (indeterminate string) *For a given finite alphabet* $\sum$*, let* $\mathcal{P}(\sum)$*be the power set of* $\sum$*. An* indeterminate string *is a sequence of* character sets*, which are elements of* $\mathcal{P}(\sum)\backslash(\emptyset)$.

In other words, for an indeterminate string S with n index positions must hold that for every i, $1 \leq i \leq n$, $S[i] \subseteq \sum$ and $S[i] \neq \emptyset$, where $S[i]$ denotes the character set associated with the i-th position in S. In the special case where every position of indeterminate string S holds a singleton set, S is equivalent to an ordinary string. We denote the *length* of an indeterminate string S with n index positions by $|S| \equiv n$ and its *cardinality*, i.e. the number of *all* elements in S, by $\| S \| \equiv \sum_{i=1}^{n} |S[i]|$. Two positions a and b, $1 \leq a \leq b \leq |S|$, induce the indeterminate *substring* $S[a,b] \equiv S[a]\ S[a+1] \dots S[b]$. To distinguish intervals in different indeterminate strings, we indicate the affiliation of an interval $[i,j]$ to indeterminate string S by the subscript notation $[i,j]_S$.

Example 1 $S = \{a,d,g\}\{c\}\{a,d\}\{e,f\}\{b\}\{c,g\}$*is an indeterminate string of length* $|S| = 6$*and cardinality* $||S|| = 11$*over alphabet* $\sum = \{a,b,c,d,e,f,g\}$*. The third element of S is given by character set* $S[3] = \{a,b\}$*. Interval* $[2,4]$*induces the substring* $S[2,4] = \{c\}\{a,d\}\{e,f\}$.

In this work, we generalize the concept of common intervals, which were initially introduced on permutations [23] and subsequently extended to strings [24]. The idea behind common intervals is to compare strings, or rather substrings, based on their character sets. The character set of an ordinary string S is defined as $\mathcal{C}(S) \equiv \{S[i] \mid 1 \leq i \leq |S|\}$. The equivalent concept on indeterminate strings is the following:

Definition 2 (character set) *The* character set *of an indeterminate string S is defined by* $\mathcal{C}(S) \equiv \bigcup_{i=1}^{n} S[i]$.

In two ordinary strings S and T over a finite alphabet Σ, two intervals, $[i, j]$ in S and $[k, l]$ in T, are called *common intervals* if $\mathcal{C}(S[i,j]) = \mathcal{C}(T[k,l])$). The analogon for indeterminate strings is:

Definition 3 (strict common intervals) *Given two indeterminate strings* S*and* T*, two intervals,* $[i,j]$*in* S*and* T*in* T*, are said to be* strict common intervals *if and only if their character sets* $\mathcal{C}(S[i,j])$*and* $\mathcal{C}(T[k,l])$*are equal.*

A weaker definition based on the intersection relation between character sets is:

Definition 4 (weak common intervals) *Given two indeterminate strings* S*and* T*, two intervals,* $[i,j]$*in* S*and* T*in* T*, are* weak common intervals *with* common character set $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$*if for each* x*,* $i \leq x \leq j$*, it holds that* $C \cap S[x] \neq \emptyset$*and for each* y*,* $k \leq y \leq l$*, it holds that* $C \cap T[y] \neq \emptyset$.

In all our use cases, in particular when dealing with conflicting gene family assignments as well as gene family-free gene cluster detection, the concept of weak common intervals appears to be more appropriate. Thus, in the following, we restrict ourselves to the study of weak common intervals.

Furthermore, continuing a previous line of research initially proposed by Schmidt and Stoye in [4], we further extend weak common intervals by allowing a limited number of insertions and deletions:

Definition 5 (approximate weak common intervals) *Given two indeterminate strings* S*and* T*and a threshold* $\delta \in \mathbb{N}_0$*, two intervals,* $[i,j]$*in* S*and* $[k,l]$*in* T*, are* approximate weak common intervals *with* common character set $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$*if the number of* positions *with no intersection with* C *is limited by* δ*, i.e.* $|\{x \mid i \leq x \leq j : S[x] \cap C = \emptyset\}| + |\{y \mid k \leq y \leq l : T[y] \cap C = \emptyset\}| \leq \delta$*. These positions are called* indels.

Generally, algorithms for discovering common intervals of ordinary strings only report pairs of intervals that both are *maximal*, whereby *maximality* is defined as follows: An interval $[i,j]$ in string X is called *maximal* if its immediate left and right neighboring characters, $X[i-1]$ and $X[j+1]$ (if such exist), are not contained in $\mathcal{C}(X[i,j])$. In other words, interval $[i,j]$ cannot be extended to its left or right without expanding the character set of the interval.

In terms of weak common intervals, we introduce the following property derived from [11]:

Definition 6 ($\mathcal{C}$-closed) *Given an indeterminate string S, an interval $[i,j]$, and a character set $C \subseteq \sum$, interval Cis C*-closed *if* $S[i]$, $S[i] \subseteq C$, *and if* $i = 1$*or* $S[i-1] \cap C = \emptyset$, *and if* $j = n$*or* $S[j+1] \cap C = \emptyset$.

A reasonable balance between omitting expedient and including apparently redundant weak common intervals is found by the subset of those that are *mutually-closed*, as defined as follows:

Definition 7 (mutually-closed) *Given a pair of intervals* $([i,j]_S, [k,l]_T)$*of indeterminate strings* S and T, $[i,j]_S$*and* $[i,j]_S$*are* mutually-closed *if* $([i,j]_S, [k,l]_T)$*is* $\mathcal{C}(T[k,l])$ *-closed and* $[i,j]_S\mathcal{C}(S[i,j])$*-closed.*

We consequently restrict the enumeration of weak common intervals and approximate weak common intervals to those that are mutually-closed.

Combinatorial complexity. The maximal number of mutually-closed weak common intervals of two indeterminate strings S and T of length n and m, respectively, is bounded by nm. This result follows from the fact that the number of intervals $[k, l]$ in T that are mutually-closed weak common intervals with any interval with fixed left bound i in S is bounded by m. Likewise, the maximal number of mutually-closed approximate weak common intervals of indeterminate strings S and T is bounded by $(\delta + 1)^2 nm$.

Gene family-free analysis. In absence of gene family assignments, each gene in the dataset is represented by a unique *character*, linearly ordered along a *chromosomal string*. Therefore, the n characters of a chromosomal string can be identified by their integer *index set* $\{1, 2, \ldots, n\}$. Relating characters of one chromosomal string to characters of another, we presume that we are given a symmetric *similarity measure* $\sigma_{AB} : A \times B \to \mathbb{R}_{\geq 0}$ for any two index sets A and B.

In gene family-free gene cluster analysis we aim at finding pairs of intervals in two chromosomal strings, whose characters are similar. We can reduce this problem to finding (approximate) weak common intervals between two indeterminate strings. To this end, we construct an *index mapping* B_A:

$$B_A[y] = \begin{cases} \{x \mid x \in A : \sigma_{AB}(x,y) > 0\} & \text{if any } x \in A \text{ exists s.t. } \sigma_{AB}(x,y) > 0 \\ \{\infty\} & \text{otherwise.} \end{cases}$$

Thus, B_A is an indeterminate string over alphabet $\{1, 2, \ldots, |A|, \infty\}$. Let $I_A = \{1\}\{2\}\cdots\{|A|\}$ represent the indeterminate string of A, a position in I_A shares a character with a position in B_A if and only if the similarity of the two corresponding characters is non-zero. Thus, finding intervals in chromosomal strings A and B, whose characters are similar, is equivalent to finding (approximate) weak common intervals of indeterminate strings I_A and B_A. Note that the set of (approximate) weak common intervals of I_A and B_A is identical to the one of I_B and A_B. The (approximate) weak common intervals differ in size and, most substantially, in the similarities between characters within the interval pairs. Therefore, we introduce a simple scoring scheme by which the solution space can be arranged into a landscape of highs and lows of (approximate) weak common intervals, ranked by taking into account the number and the similarities of the contained characters. We define a score function μ_{xy} over an index x in index set X and an interval $[a, b]_Y$ in index set Y as

$$\mu_{XY}(x, [a,b]_Y) = \begin{cases} \dfrac{\max\limits_{y \in [a,b]_Y} \sigma_{XY}(x,y)}{\max\limits_{z \in Y} \sigma_{XY}(x,z))} & \text{if } \max\limits_{z \in Y} \sigma_{XY}(x,z) > 0 \\ 0 & \text{otherwise} \end{cases}$$

so that μ_{xy} takes values between 0 and 1, being 1 if the gene with highest similarity to x lies within interval $[a, b]_Y$. The overall score of two interval pairs $([i, j]_A, [k, l]_B)$ is then

$$score(([i,j]_A, [k,l]_B)) = \sum_{x=i}^{j} \mu_{AB}(x, [k,l]_B) + \sum_{y=k}^{l} \mu_{BA}(y, [i,j]_A)$$

We now describe three algorithms to compute all mutually-closed weak common intervals and all mutually-closed approximate weak common intervals with at most δ indels in two indeterminate strings. Note that mutually-closed weak common intervals are a special subclass of mutually-closed approximate weak common intervals for $\delta = 0$.

In the following, we consider two indeterminate strings S of length n and T of length m.

Discovering weak common intervals

We now describe the algorithm ***W**eak **C**ommon **I**ntervals on **I**ndeterminate **S**trings* (WCII) as presented in Figure 1. It solves the following problem:

Problem 1 *Given two indeterminate strings $S$and T, discover all mutually-closed weak common intervals of S and T.*

To tackle this problem we make use of the following constructs:

Definition 8 (index string) *Given an indeterminate string $S$of length n, $I_S \equiv \{1\}\{2\}\cdots\{n\}$ denotes the* index string *of S.*

Definition 9 (index mapping) *Given two indeterminate strings $S$and T of lengths n and m respectively, the* index mapping *of S onto T is given by* $(T_S[y])_{y=1,\ldots,m}$, *where*

$$T_S[y] = \begin{cases} \{x \mid x = 1, \ldots, n : S[x] \cap T[y] \neq \emptyset\} & \text{if } T[y] \cap \mathcal{C}(S) \neq \emptyset \\ \{\infty\} & \text{otherwise.} \end{cases}$$

```
Input: Two indeterminate strings $S$ and $T$
Output: All mutually-closed weak common intervals of $S$ and $T$
 1: Construct indeterminate string $T_S$ and POS
 2: for $i \leftarrow 1$ to $|S|$ do
      // initialize LIST with positions in $T_S$ that contain character $i$
 3:   Initialize empty list LIST
 4:   for each element $y$ in POS$[i]$ do
        // tuple storing current position $y$ in path of rank-nearest successors, and current path bounds
 5:     add $(y, [y, y])$ to LIST
 6:   end for
      // main loop
 7:   $j \leftarrow i$
 8:   while LIST is not empty do
 9:     PREVIOUS $\leftarrow [\infty, \infty]$
10:     Initialize empty list LIST$'$
11:     for each element $(y, [p_k, p_l])$ in LIST do
12:       $[k, l] \leftarrow$ min-rank interval of character $j$ around position $y$ in $T_S$.
13:       if $i - 1 \notin \mathcal{C}(T_S[k, l])$ then
            // test if there are interval pairs to output
14:         $y' \leftarrow$ rank-nearest successor of $y$ if such exists, otherwise $\infty$
15:         if $[p_k, p_l] \subseteq [k, l]$ and $[k, l] \neq$ PREVIOUS and $y' \notin [k, l]$ then
16:           output $([i, j], [k, l])$
17:           PREVIOUS $\leftarrow [k, l]$
18:         end if
            // compute the next level
19:         if $y' \neq \infty$ and $y$ is the leftmost index with shortest min-rank distance to $y'$ among
            positions of the list having the same rank-nearest successor $y'$ then
              // update path bounds
20:           $p'_k \leftarrow \min(p_k, y')$
21:           $p'_l \leftarrow \max(p_l, y')$
22:           add $(y', [p'_k, p'_l])$ to LIST$'$
23:         end if
24:       end if
25:     end for
26:     LIST $\leftarrow$ LIST$'$
        // Increase right bound $j$
27:     $j \leftarrow j + 1$
28:   end while
29: end for
```

Figure 1 WCII algorithm. WCII adapts the search strategy of Didier's Algorithm [24] for common intervals in strings to the computation of weak common intervals in indeterminate strings.

Note that index strings and index mappings are again indeterminate strings. The key idea of WCII arises from the following observation:

Observation 1 *Given two indeterminate strings $S$and $T$with index string $I_S$and index mapping T_S, two intervals Sin $S$and $[k, l]$in $T$are weak common intervals if and only if $[i, j]_{I_S}$and $[k, l]_{T_S}$are weak common intervals.*

This equivalence holds because any two positions, x in S and y in T intersect if and only if $I_S[x]$ and $T_S[y]$ intersect. Since it holds that $I_S[x] = \{x\}$ for all $x = 1, \ldots, n$, we simplify the notation of single character set $I_S[x]$ to just x and character set $\mathcal{C}(I_S[i, j])$ to just $[i, j]$. Note that character $c \in \mathcal{C}(I_S[i, j])$ serves subsequently both as character $c \in [i, j]$ as well as index in I_S.

WCII is an adaptation of Didier's Algorithm [24] of enumerating maximal common intervals in ordinary strings. Didier's strategy can be described as follows: The algorithm iterates over all positions i in S as possible left interval bounds. In each iteration all mutually-closed weak common interval pairs are reported that share the same left bound i in I_S. For each possible right bound $j \geq i$, the algorithm iterates through the set of positions in T_S that contain j in their character set. To this end, we make use of an array POS, where POS[j], $1 \leq j \leq n$, is a sorted list of positions in T_S containing character j. Each position $y \in$ POS[j] is associated with an interval $[k, l]_{T_S}$, $k \leq y \leq l$, called the *min-rank interval* of character j for position y. It is the largest interval around y for which every position in $[k, l]_{T_S}$ contains at least one character in $[i, j]$. Obviously, $[k, l]_{T_S}$ is $[i, j]$-closed. It remains to be checked if $[i, j]_{I_S}$ is closed w.r.t. $\mathcal{C}(T_S[k, l])$ and that every position in $[i, j]_{I_S}$ and $[k, l]_{T_S}$ contains a character from $C = \mathcal{C}(I_S[i, j]) \cap \mathcal{C}(T_S[k, l])$. To show the latter, it is sufficient to show that $[i, j] \subseteq \mathcal{C}(T_S[k, l])$, because the character set of each position in I_S corresponds to the single element set of its index. The details of both tests are explained below, after relevant data structures are introduced. If both conditions are satisfied, a mutually-closed weak common interval pair is found and subsequently reported.

Like in Didier's Algorithm, we employ two tricks that improve the performance: precomputing *min-rank intervals* and following paths of *rank-nearest successors*.

Precomputing min-rank intervals. In order to identify min-rank intervals, it is sufficient to observe the smallest character $c \geq i$ in each position. To this end, we make use of the following construct:

Definition 10 (*i*-reduced string) *Given index mapping* T_S, $(T_S^i[y])_{y=1,\ldots,m}$*is the* i-reduced string of T_S of the ith iteration, *where* $T_S^i[y] = \min(\{c \mid c \in T_S[y] \cup \{\infty\} : c \geq i\})$.

Min-rank intervals in T_S^i are identical to *rank intervals* as initially defined by Dider *et al.* [24]. Interestingly, rank intervals in T_S^i correspond directly to min-rank intervals in T_S:

Lemma 1 *The set of min-rank intervals in* T_S*is identical to the set of rank intervals in* T_S^i.

Proof: Didier *et al.*[24] show that rank intervals in a string are nested and that their number is bounded by the length of the string.

Observe that for any position y in T_S^i the rank interval of character $j = T_S^i[y]$ is identical to the min-rank interval of j at position y in T_S. Let y be a position in T_S and $j \in T_S[y]$ such that $j > T_S^i[y]$. Further, let $[k, l]_{T_S}$ be the min-rank interval of j at $T_S[y]$, $j' = \max(\{c \mid c \in \mathcal{C}(T_S^i[k, l]) : c \leq j\})$, and $[k', l']_{T_S}$ be the min-rank interval of j' at its corresponding position in T_S. Because $j' \leq j$ it consequently holds that $[k', l']_{T_S} \subseteq [k, l]_{T_S}$. Now, according to the definition of min-rank intervals, $T_S^i[k' - 1] > j'$, if such position exists. Since j', is the largest character in $T_S^i[k, l]$ that is smaller than or equal to j, it must also hold that $T_S^i[k' - 1] > j$. The same argument holds for $T_S^i[l' + 1]$ if such position exists, therefore $[k, l]_{T_S} = [k', l']_{T_S}$ is the min-rank interval of both characters j' and j. We conclude that all min-rank intervals for any character in T_S at iteration i are contained in the set of rank intervals of T_S^i. □

Consequently, all min-rank intervals in T_S in the ith iteration (i.e. for a fixed left bound i in I_S) can be precomputed in $\mathcal{O}(m)$ time using the algorithm given by Didier *et al.* [24]. They are stored in table INT. For a currently processed character j at position y in T_S, INT [y] contains its corresponding min-rank interval. Unlike Didier's Algorithm, INT must be updated after each iteration such that all positions in INT accessed in the following (j + 1)th iteration contain the corresponding min-rank intervals of character j + 1. Details of the update step can be found in Additional file 1 Section 1.1.

Following paths of rank-nearest successors. The second trick in the algorithm consists in increasing the right bound j in I_S while walking through positions and characters of T_S. Thereby the algorithm jumps from a current position y that contains character j to its *rank-nearest successor*, which is the position y' containing character j + 1 with the smallest *min-rank distance* to y as defined as follows:

Definition 11 (min-rank distance) *The* min-rank distance *of any two positions* k *and* l *in indeterminate string* T_S *for the ith iteration is given by:*

$$d_{T_S}^i(k, l) \equiv \max(\{T_S^i[p] \mid k \leq p \leq l\})$$

If several co-optimal positions are available, the tie is broken by choosing the leftmost one as rank-nearest successor. In case no position with character j + 1 exists, or the smallest min-rank distance is '∞', j has no successor. For the ith iteration, all rank-nearest successors are precomputed and stored in table SUCC which is explained in more detail in Additional file 1 Section 1.2.

Connecting characters larger than or equal to i at their corresponding positions in T_S with their rank-nearest successors through directed edges results in a forest of rooted trees. Nodes (across all trees) sharing the same character are said to reside on the same *level*. In lines 8-28 of Figure 1, the algorithm traverses along paths through this forest in a bottom-up procedure, from one level to the next, starting at those leaves with character i. Besides the currently visited nodes of the level, the algorithm keeps track of the *path bounds*, which are the outermost positions in T_S a path has visited thus far. The currently visited nodes of the paths and their corresponding path bounds are stored in a list labeled LIST. Only after all nodes of the same level j are processed, the algorithm follows all current paths to nodes of the next level $j + 1$, thereby ensuring that each character in T_S is processed at most once. To this end, for all positions containing character j that have the same rank-nearest successor y', the algorithm discontinues the paths of all but the leftmost one with shortest min-rank distance to y' (line 19). Traversing along paths of rank-nearest successors in WCII differs from Didier's Algorithm by the fact that a position in T_S may be visited by the same path several times on different levels.

For any given min-rank interval $[k, l]_{T_S}$ there cannot be more than one weak common interval partner in I_S starting at position i. Therefore it is sufficient to track at least one path in each min-rank interval to find all mutually-maximal intervals of I_S and T_S. Positions in POS are sorted, thus paths leading to the same weak common interval pair appear adjacent to each other in LIST and the common interval pair is reported only for the first (lines 15-17).

For each node in LIST, associated with character j and position y, the algorithm checks if the min-rank interval $[k, l]_{T_S}$ of j encloses the path bounds up to position y (see condition in line 15). If validated, a weak common interval pair has been found, given by $([i, j]_{I_S}, [k, l]_{T_S})$. To ensure mutual closedness, the interval pair is only reported if $i - 1$ is not contained in the character set $\mathcal{C}(T_S|k, l|)$ and the successor of y is not within the current bounds of its path (see conditions in lines 13 and 15). Checking for the former can be achieved in $\mathcal{O}(1)$ time after $\mathcal{O}(m)$ time preprocessing by performing a range minimum query on an array of size $\mathcal{O}(m)$ where each position containing character $i - 1$ is assigned 0 and 1 otherwise.

The overall complexity of the algorithm can be summarized as follows: Each position in I_S is regarded exactly once as left bound i for all weak common intervals that are reported in one iteration. Once T_S^i is computed for $i = 1$ it can be up-dated using array POS, taking overall $\mathcal{O}(||T_S||)$ time for all left bounds $i = 1, \ldots, n$. Further, for each left bound the algorithm performs $\mathcal{O}(m)$ steps to precompute all min-rank intervals and $\mathcal{O}(||T_S||)$ steps to precompute all rank-nearest successors. The subsequent bottom-up procedure and the reporting of weak common intervals requires again $\mathcal{O}(||T_S||)$ time. Therefore we have:

Theorem 1 *Given two indeterminate strings $\mathcal{S}$and $\mathcal{T}$, Algorithm WCII finds all pairs of mutually-closed weak common intervals of $\mathcal{S}$and $\mathcal{T}$in $\mathcal{O}(n||T_S||)$ time.*

Discovering approximate weak common intervals

We now present the algorithm ***A**pproximate **W**eak **C**ommon **I**ntervals on **I**ndeterminate Strings* (AWCII) as presented in Figure 2, thus line numbers mentioned in this subsection refer to Figure 2. AWCII solves the following problem:

Problem 2 *Given two indeterminate strings $\mathcal{S}$and $\mathcal{T}$and indel threshold $\delta \in \mathbb{N}_0$, discover all mutually-closed approximate weak common intervals of $\mathcal{S}$and $\mathcal{T}$with no more than δ* indels.

Following a strategy similar to WCII, AWCII solves Problem 2 for index mappings I_S and T_S, instead of S and T. As before, in each iteration the algorithm maintains a fixed left bound i in I_S. For each character $j \in [i, n]$ all positions y in T_S are processed that contain character j (lines 5-25). Thereby character j at position y in T_S can be associated with several different intervals around y that are candidates of mutually-closed approximate weak common interval partners for interval $[i, j]_{I_S}$. Only intervals surrounding one (or several) positions y can be mutually-closed. However, for an interval $[k, l]_{T_S}$ containing indels, it no longer holds that the minrank distance of any two positions within the interval is always smaller than the min-rank distance from any position inside to any position outside the interval. As a result, neither precomputed min-rank intervals nor following paths of ranknearest successors can be used for improving the algorithm's performance. Instead we pursue a different approach, thereby making AWCII an adaptation of the RGC algorithm of Jahn [11].

For each $d_k = 1,..., \delta$ (lines 7-23) AWCII identifies the leftmost position k in T_S such that at most d_k indels are contained in interval $[k, y]_{T_S}$ and $T_S[k] \cap [i, j] \neq \emptyset$. Let $d'_k \leq d_k$ be the observed number of indels in $[k, l]_{T_S}$ (see line 9), the algorithm then finds for each $d_l = 1,..., \delta - d'_k$ (lines 14-21) the rightmost position l such that again $T_S[l] \cap [i, j] \neq \emptyset$ and the number of indels in $[y, l]_{T_S}$ does not exceed d_l. Each (k, l) of the at most $(\delta + 1)^2$ combinations of leftmost and rightmost positions gives rise to a candidate pair of mutually-closed approximate weak common intervals $([i, j]_{I_S}, [k, l]_{T_S})$. For each candidate pair it is checked that the number of characters in $[i, j]$ not contained in $\mathcal{C}(T_S[k, l])$ plus the already consumed number of indels in $[k, l]_{T_S}$ does not exceed δ. Finally, it is tested if $[i, j]_{I_S}$ is $\mathcal{C}(T_S[k, l])$-closed. If both conditions are satisfied, a mutually-closed approximate weak common interval pair is found and is subsequently reported (line 18).

Runtime improvements are achieved by precomputing right and left bounds of candidate intervals $[k, l]_{T_S}$ for each character j in T_S. These bounds are computed within $\mathcal{O}((\delta + 1)||T_S||)$ time for a fixed left bound i in I_S and stored in tables L and R respectively. Further, for each such candidate interval $[k, l]_{T_S}$ the number of characters that are within $[i, j]$ can also be precomputed. This number is used to determine δ_S in line 16. The construction of the corresponding table, called RANGE-CONTENT, is achieved within $\mathcal{O}((\delta + 1)^2||T_S||)$ time for

Input: Indeterminate strings S, T, indel threshold δ.
Output: All mutually-closed approximate weak common intervals of S and T with at most δ indels

```
Construct indeterminate string $T_S$ and POS
for $i \leftarrow 1$ to $|S|$ do
   for $j \leftarrow i$ to $|S|$ do
      PREVIOUS $\leftarrow -\infty$
      for each element $y$ in POS$[j]$ do
         $d_k \leftarrow \delta$
         while $d_k \geq 0$ do
            $k \leftarrow$ leftmost position in $T_S$ s.t. $|\{p \mid k \leq p \leq y : T_S[p] \cap [i,j] = \emptyset\}| \leq d_k$
            // $d'_k$ corresponds to the observed number of indels in $[k,y]_{T_S}$
            $d'_k \leftarrow |\{p \mid k \leq p \leq y : T_S[p] \cap [i,j] = \emptyset\}|$
            if PREVIOUS $\geq k$ then
               break
            end if
            $d_l \leftarrow \delta - d_k$
            while $d_l \geq 0$ do
               $l \leftarrow$ rightmost position in $T_S$ s.t. $|\{p \mid y \leq p \leq l : T_S[y] \cap [i,j] = \emptyset\}| \leq d_l$
               $d_S \leftarrow j - i - |\mathcal{C}(T_S[k,l]) \cap [i,j]| + 1$
               if $d_S + d_k + d_l \leq \delta$ and $\{i-1, j+1\} \cap \mathcal{C}(T_S[k,l]) = \emptyset$ and $i \in \mathcal{C}(T_S[k,l])$ then
                  output $([i,j],[k,l])$
               end if
               $d_l \leftarrow |\{p \mid y \leq p \leq l : T_S[y] \cap [i,j] = \emptyset\}| - 1$
            end while
            $d_k \leftarrow d'_k - 1$
         end while
         PREVIOUS $\leftarrow y$
      end for
   end for
end for
```

Figure 2 AWCII algorithm. AWCII is a search algorithm for approximate weak common intervals in indeterminate strings. It is an adaptation of RGC [11], an algorithm for computing approximate common intervals in strings.

a fixed left bound i. The details of constructing tables L, R, and RANGECONTENT can be found in Additional file 1 Section 2. Note that RANGECONTENT differs significantly from the data structure NUM used in RGC to count characters in intervals.

In terms of overall runtime, for each fixed bound i in I_S the algorithm spends $\mathcal{O}((\delta+1)^2||T_S||)$ time on computation of the above mentioned auxiliary tables. Thereafter, AWCII requires $\mathcal{O}((\delta+1)^2||T_S||)$ time to iterate through all combinations of candidate intervals in L and R and to identify approximate weak common intervals.

Testing for $\mathcal{C}(T_S[k,l])$-closedness of interval $[i,j]_{I_S}$ can be achieved in $\mathcal{O}(1)$ time by precomputing a table for all candidate intervals in T_S of the ith iteration, where each entry indicates if a character $i - 1$ or $j + 1$ is contained in the corresponding candidate interval. Such a table can be constructed within $\mathcal{O}((\delta+1)\cdot||T_S||)$ time using again a simple sweep algorithm. We conclude with the following theorem:

Theorem 2 *Given two indeterminate strings $\mathcal{S}$and $\mathcal{T}$and indel threshold $\delta \in \mathbb{N}_0$, algorithm AWCII computes all pairs of mutually-closed approximate weak common intervals of $\mathcal{S}$and $\mathcal{T}$in $\mathcal{O}((\delta+1)^2 \cdot n^2||T_S||)$time.*

A runtime heuristic for discovering approximate weak common intervals

Our third algorithm, ACSI (see Figure 3) represents a runtime heuristic that solves Problem 2 exactly and in practice outperforms both WCII and AWCII in gene family-free analysis by orders of magnitude.

Just as the two algorithms before, ACSI operates on index string I_S and index mapping T_S instead of indeterminate strings S and T directly. For every fixed interval $[i, j]$ in I_S, ACSI identifies mutually-closed approximate

Input: Indeterminate strings S, T, indel threshold δ.
Output: All mutually-closed approximate weak common intervals of S and T with at most δ indels

```
1: Construct indeterminate string $T_S$ and Pos
2: for $i \leftarrow 1$ to $|S|$ do
3:   for each element $y$ in Pos$[i]$ do
4:     for $j \leftarrow i$ to $|S|$ do
5:       EXTEND$(\delta, i, j, y, y)$
6:     end for
7:   end for
8: end for

9: procedure EXTEND$(d, i, j, k, l)$
10:   Increase interval $[k, l]_{T_S}$ to both sides until $[i, j] \cap T_S[k-1] = [i, j] \cap T_S[l+1] = \emptyset$
11:   if $i - 1 \in \mathcal{C}(T_S[k, l])$ or $j + 1 \in \mathcal{C}(T_S[k, l])$ then
12:     return
13:   end if
      // test if there are interval pairs to output
14:   if $j - i + 1 - |\mathcal{C}(T_S[k, l]) \cap [i, j])| \leq d$ and $i \in \mathcal{C}(T_S[k, l])$ then
15:     output $([i, j]_S, [k, l]_T)$
16:   end if
      // extend to both directions until all indels d are consumed
17:   $k' \leftarrow$ highest index strictly smaller $k - 1$ such that $\mathcal{C}(T_S[k', k-1]) \cap [i, j] \neq \emptyset$
18:   if $k - k' - 1 \leq d$ and $\{i-1, j+1\} \cap \mathcal{C}(T_S[k', k-1]) = \emptyset$ then
19:     EXTEND$(d + k' + 1 - k, i, j, k', l)$
20:   end if
21:   $l' \leftarrow$ lowest index strictly larger $l + 1$ such that $\mathcal{C}(T_S[l+1, l']) \cap [i, j] \neq \emptyset$
22:   if $l' - l - 1 \leq d$ and $\{i-1, j+1\} \cap \mathcal{C}(T_S[l+1, l']) = \emptyset$ then
23:     EXTEND$(d + l + 1 - l', i, j, k, l')$
24:   end if
25: end procedure
```

Figure 3 ACSI algorithm. ACSI is a runtime heuristic that computes all approximate weak common intervals in indeterminate strings.

weak common interval partners $[k, l]$ in T_S. To this end, it iterates through elements of POS$[i]$, i.e. positions in T_S that contain character i (lines 3-7 of Figure 3). For each such position $y \in$ POS$[i]$ the algorithm calls a recursive procedure, denoted EXTEND (line 5). This recursive procedure requires 5 parameters, corresponding to fixed bounds $[i, j]_{I_S}$, the currently processed interval $[k, l]$ in T_S, and the current number of allowed indels, d. In the initial call, interval $[k, l]_{T_S}$ is set to $[y, y]_{T_S}$ and $d = \delta$. EXTEND then increases interval $[k, l]_{T_S}$ to both sides until $[i, j] \cap T_S[k - 1] = \emptyset$ and $[i, j] \cap T_S[l + 1] = \emptyset$ (line 10). If in this process the algorithm observes characters $i - 1$ or $j + 1$ in $\mathcal{C}(T_S[k, l])$, EXTEND returns to the previous call (lines 11-13). Otherwise, it verifies if $([i, j]_{I_S}, [k, l]_{T_S})$ is a mutually-closed approximate weak common interval pair by testing if the number of characters in $[i, j]$ that are missing in $\mathcal{C}(T_S[k, l])$ is less than or equal to the current d and if $i \in \mathcal{C}(T_S[k, l])$ (line 14). The interval pair is reported if both conditions are validated. EXTEND then increases $[k, l]_{T_S}$ to the left, thereby consuming indel positions as long as their overall number remains less than or equal to the current d (line 17). If a position $k' < k - 1$ has been found such that $[i, j] \cap T[k'] \neq \emptyset$, EXTEND is called recursively with parameter values $[i, j]_{I_S}$, $[k', l]_{T_S}$, and the remaining number of allowed indels, given by $d + k' + 1 - k$ (lines 18-20). This step is also symmetrically executed for the right side of $[k, l]_{T_S}$ (lines 21-24).

The actual heuristic speed-up of the algorithm is achieved by calling procedure EXTEND in line 5 not for all intervals $[i, j]$ in I_S but only for those that have chances of success for being a weak common intervals pair with an interval $[k, l]$ around a position $y \in$ POS $[i]$. Thus, the neighborhood around position y is scanned for suitable values of j prior to the execution of EXTEND. The details are described in Additional file 1 Section 3.

Results and discussion

In the following, we highlight the benefit of our dynamic model in comparison with present approaches. Although conflicting gene family assignments are extremely common in practice, this scenario is difficult to evaluate. Assuming the existence of an ultimately true gene family assignment, conflicts arise by incorrect gene family assignments. Therefore an evaluation would inevitably result in benchmarking gene family prediction tools, rather than scrutinizing our model.

Instead, we decided to evaluate our gene family-free model against the traditional gene family-based approach. To this end, we chose a genomic dataset of bacterial genomes that has been used in a prior gene cluster study [8] and was originally obtained from [25]. The dataset features 133 chromosomal sequences, of which we removed all sequences originating from plasmids.

In practice ACSI outperforms both WCII and AWCII as shown by Figure 4. Thus, in all subsequent results, we used ACSI to compute mutually-closed (approximate) weak common intervals.

Gene family-based dataset. Genes in this dataset are annotated with COG (Clusters of Orthologous Groups) identifiers [12] which are used to establish homology relationships between genes. The set of genes in the dataset was revised by the latest available gene information under the accession numbers of the respective genomes at NCBI. To this end, genes that are meanwhile marked as pseudo genes were removed from the dataset. No genes were added, since COG annotations of new genes are not available. We further omitted all genomes from subsequent analyses of which more than 10 pseudo genes were removed in this process. 93 genomes remained, comprising on average 2726 genes (minimum/ maximum number of genes: 784/8317).

Gene family-free dataset. Pairwise similarities between genes in the dataset were obtained using the relative reciprocal BLAST score (RRBS) [26]. Genes were compared on the basis of their encoding protein sequence using BLASTP+ [27] with an e-value threshold of 0.1 and disabled composition-based score adjustments.

For evaluation purposes, we produced different degrees of pruned gene similarity sets by filtering spurious gene similarities. For this, we employed an undirected variant of the stringency criterion (parameterized by $f \in [0, 1]$) for gene similarities proposed in [28], which is described in more detail in Additional file 1 Section 4.1.

To evaluate the gene family-free model, we ran an implementation of ACSI for $\delta = 0$ on the unpruned gene similarity graph of our dataset and compared the 4015841 interval pairs with respect to the contained COG identifiers. We discarded all pairs for which at least one interval contained less than two genes with a COG identifier. In the remaining 1194036 interval pairs, we observed that the similarity in the set of COG identifiers depends strongly on the intervals' score (Table 1). Among the clusters with a score greater or equal 10, 95% have the same set of identifiers in both intervals. While this number decreases for smaller scores, still a quarter of the interval pairs with a score lower than 1 do not differ in their COG identifiers. This shows that our approach is able to detect gene clusters that would also be detected with well-established gene family based approaches.

This is not a surprise, as weak common intervals are in fact a generalization of the classic common intervals model: A run of ACSI on a dataset where similarity scores are only set between members with the same COG identifiers finds the exact same set of clusters as the common intervals based approach.

To evaluate the predictive power of our approach, we compare the output of our program to gene clusters predicted by the *reference gene cluster algorithm* (RGC) [11]. While this algorithm is capable of multiple genome comparison and the detection of faint conservation patterns, we use it in this context for pairwise genome comparison to detect interval pairs (I_1,I_2) whose gene sets have a symmetric set distance of at most 2. It has been previously observed that the generalization to approximate conservation underlying the reference gene cluster approach is not only a way to find imperfectly

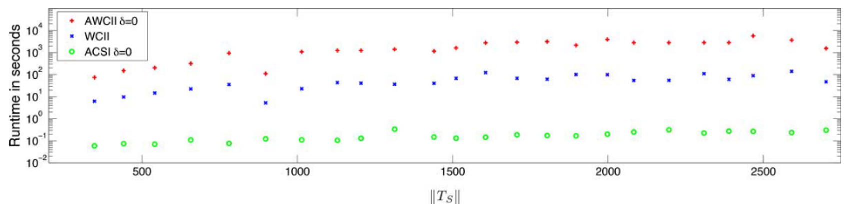

Figure 4 Runtimes of presented algorithms in practice. Running times of ACSI and AWCII with $\delta = 0$ and WCII, measured in a sample of 24 arbitrarily chosen pairwise comparisons of genomes that are contained in the studied dataset. All algorithms produced identical output (as expected). Running times are plotted against the number of pairwise gene similarities (equivalent to the size of $||T_S||$) contained in the pairwise comparison.

Table 1 Statistics of overlaps between the COG identifier sets of pairs of weak common intervals.

	score										
overlap in %	**<1**	**[1 − 2]**	**[2 − 3**	**[3 − 4]**	**[4 − 5]**	**[5 − 6]**	**[6 − 7]**	**[7 − 8]**	**[8 − 9]**	**[9 − 10]**	**≥ 10**
100	28.1	22.0	46.7	78.4	90.2	75.6	84.6	63.2	86.5	78.4	95.0
[80 − 100[	0.0	0.0	0.1	0.2	0.4	1.8	2.0	10.4	8.2	18.5	4.9
[60 − 80[	1.7	1.7	2.7	2.1	2.7	13.6	8.4	17.4	4.0	2.6	0.2
[40 − 60[	12.0	14.7	18.5	9.9	2.4	2.5	2.1	5.0	0.7	0.3	0.0
[20 − 40[	0.1	0.1	0.3	0.7	1.1	3.4	1.5	1.8	0.1	0.2	0.0
[0 − 20[	58.1	61.4	31.8	8.8	3.2	3.1	1.4	2.7	0.6	0.2	0.0
Total	30002	239077	289450	253643	199372	49254	58889	17952	23603	4568	28226

Columns stand for bins of weak common interval scores, rows for bins of overlap sizes of the COG identifier sets of weak common interval pairs. Values are given in percent with respect to the total number of pairs per score bin given in the last row.

conserved clusters, but also a means to add robustness against errors in gene family assignment. For example, an interval pair may appear to have a set distance of two because besides the shared genes, there is one gene unique to I_1 and one gene unique to I_2. However a closer inspection of the genes reveals that these genes are in fact homologs that were not recognized in the preceding partitioning of genes into homology families. We ran RGC on all pairs of the 93 genomes setting parameters $\delta = 2$ (maximal tolerated symmetric set distance) and $s = 3$ (the minimum cluster size). The program returned among others 192900 "single-mismatch clusters", i.e. clusters that have exactly one extra gene in each interval. In 47453 (24.60%) of the single-mismatch clusters, we observe a similarity score between the two extra genes in our BLAST dataset. ACSI found 89.84% of the single-mismatch clusters and for 75.24% the extra genes turned out to be pairwise best hits. Moreover we observe that in 18143 among the single-mismatch gene clusters predicted by RGC the two extra genes have exactly the same annotation string. (Annotations containing the word "hypothetical" were ignored.) ACSI finds 90.19% of these clusters. Surprisingly, 4.59% of the single-mismatch clusters in which the two extra genes had best hits to each other were not found by ACSI. This is because for one or more of the other genes in the cluster our BLAST results did not return any similarity score to a gene in the other interval. Apparently the elements of a cluster of orthologous groups can be very faintly related in terms of sequence similarity.

Comparison with RegulonDB data. Among other information about transcriptional regulation, RegulonDB [29] provides a list of operon locations in *Escherichia coli* K12. While the majority of operons in RegulonDB are computationally predicted, some are also experimentally confirmed. From 2649 operons reported in RegulonDB, 846 span two or more genes. We mapped these operons to the annotation of the *E. coli* K12 genome in our data set. However, 104 operons contain genes that are not annotated in our dataset and thus were omitted from subsequent analysis. The remaining 742 operons span between 2 and 16 genes, 71.83% of which span 2 or 3 genes. The number of detected gene clusters depends strongly on the degree of evolutionary relatedness between the *E. coli* K12 genome and other genomes in the dataset. While ACSI and RGC predicted many occurrences in other close related γ-proteobacteria in our dataset, for the majority of genomes only few occurrences of operons were reported. Additional file 1 Section 4.2, gives an overview of the number of found gene clusters in the dataset. The sets of reported operons found by ACSI and RGC are not entirely overlapping. Instead, ACSI finds operons which RGC does not find and vice versa. A complete overview of unique findings for algorithms and parameter settings is shown in Table 2.

Conclusions

In this work we introduced a new model to detect gene clusters based on the study of (approximate) weak common intervals in indeterminate strings. In context of gene family-free analysis, we presented a scoring scheme for (approximate) weak common intervals which rates both interval size and the degree of similarity between the contained genes of an (approximate) weak common interval pair. We use our gene family-free model to predict gene clusters between pairs of genomes. This approach is evaluated in comparison with the common intervals-based reference gene cluster model.

In addition to the use case of detecting gene clusters, our algorithms can also be helpful to identify synteneous blocks in a gene family-free analysis. The hierarchical nature of common intervals is maintained in our weak common intervals model, which makes it ideal for studying potential synteneous blocks of arbitrary resolution. The basic concept of common intervals in strings has seen many generalizations in the past years which have greatly increased its utility for biological studies, in particular the

Table 2 Unique findings (with 100% overlap) of operons by ACSI and RGC with minimum cluster size $s = 2$ and varying parameters.

Unique to. . .	RGC $\delta = 0$	RGC $\delta = 2$	ACSI $\delta = 0$, $f = 0.0$	ACSI $\delta = 0$, $f = 0.9$	ACSI $\delta = 2$, $f = 0.0$	ACSI $\delta = 2$, $f = 0.9$
RGC $\delta = 0$	-	118	133	119	190	175
RGC $\delta = 2$	0	-	56	49	80	72
ACSI $\delta = 0$, $f = 0.0$	4	45	-	0	61	52
ACSI $\delta = 0$, $f = 0.9$	11	59	21	-	82	62
ACSI $\delta = 2$, $f = 0.0$	0	8	0	0	-	0
ACSI $\delta = 2$, $f = 0.9$	5	20	11	0	20	-

Each column shows the number of unique findings of an algorithm and parameter setting indicated by the column heading in comparison to algorithms and parameter settings specified in the rows.

simultaneous consideration of more than two strings, requiring common intervals to occur in all or at least a certain number of them. This generalization of (approximate) weak common intervals in indeterminate strings is undoubtedly an interesting direction for future work.

Additional material

Additional file 1: Supporing Information

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

Authors' contributions
All authors were involved in the early conception of the project. DD, KJ and JS developed the methods and designed the analysis. DD and KJ performed the evaluation and wrote the manuscript; all authors discussed the results, commented on the manuscript, and read and approved its final version.

Acknowledgements
DD receives a scholarship from the CLIB Graduate Cluster Industrial Biotechnology. KJ is funded by DFG grant ST 431/5-1.

Declarations
We acknowledge support for the Article Processing Charge by the German Research Foundation and the Open Access Publication Fund of Bielefeld University Library.
This article has been published as part of *BMC Genomics* Volume 15 Supplement 6, 2014: Proceedings of the Twelfth Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics. The full contents of the supplement are available online at http://www.biomedcentral.com/bmcgenomics/supplements/15/S6.

Authors' details
[1]Genome Informatics, Faculty of Technology, Bielefeld University, Bielefeld, Germany. [2]Institute for Bioinformatics, Center for Biotechnology (CeBiTec), Bielefeld University, Bielefeld, Germany. [3]Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Jena, Germany. [4]Computational Biology Group, Department of Biosystems Science and Engineering, ETH Zürich, Basel, Switzerland.

Published: 17 October 2014

References
1. Tamames J, *et al*: **Evolution of gene order conservation in prokaryotes.** *Genome Biol* 2001, **2(6)**:1-0020.
2. Wolfe KH, Shields DC: **Molecular evidence for an ancient duplication of the entire yeast genome.** *Nature* 1997, **387**:708-713.
3. Heber S, Stoye J: **Algorithms for finding gene clusters.** *Proceedings of WABI 2001 LNCS* 2001, **2149**:252-263.
4. Schmidt T, Stoye J: **Quadratic time algorithms for finding common intervals in two and more sequences.** *Proc of CPM 2004 LNCS* 2004, **3109**:347-358.
5. Heber S, Mayr R, Stoye J: **Common intervals of multiple permutations.** *Algorithmica* 2011, **60(2)**:175-206.
6. Bergeron A, Corteel S, Raffinot M: **The algorithmic of gene teams.** *Proceedings of WABI 2002 LNCS* 2002, **2452**:464-476.
7. He X, Goldwasser MH: **Identifying conserved gene clusters in the presence of homology families.** *J Comp Biol* 2005, **12(6)**:638-656.
8. Ling X, He X, Xin D: **Detecting gene clusters under evolutionary constraint in a large number of genomes.** *Bioinformatics* 2009, **25(5)**:571.
9. Rahmann S, Klau GW: **Integer linear programs for discovering approximate gene clusters.** *Proceedings of WABI 2006 LNBI* 2006, **4175**:298-309.
10. Böcker S, Jahn K, Mixtacki J, Stoye J: **Computation of median gene clusters.** *J Comput Biol* 2009, **16(8)**:1085-1099.
11. Jahn K: **Efficient computation of approximate gene clusters based on reference occurrences.** *J Comput Biol* 2011, **18(9)**:1255-1274.
12. Tatusov RL, Fedorova ND, Jackson JD, Jacobs AR, Kiryutin B, Koonin EV, Krylov DM, Mazumder R, Mekhedov SL, Nikolskaya AN, Rao BS, Smirnov S, Sverdlov AV, Vasudevan S, Wolf YI, Yin JJ, Natale DA: **The COG database: an updated version includes eukaryotes.** *BMC Bioinformatics* 2003, **4**:41.
13. Powell S, Szklarczyk D, Trachana K, Roth A, Kuhn M, Muller J, Arnold R, Rattei T, Letunic I, Doerks T, Jensen LJ, von Mering C, Bork P: **eggNOG v3.0: orthologous groups covering 1133 organisms at 41 different taxonomic ranges.** *Nucleic Acids Res* 2012, **40(Database)**:284-9.
14. Waterhouse RM, Zdobnov EM, Tegenfeldt F, Li J, Kriventseva EV: **OrthoDB: the hierarchical catalog of eukaryotic orthologs in 2011.** *Nucleic Acids Res* 2011, **39(Database)**:283-8.
15. Shi G, Peng MC, Jiang T: **MultiMSOAR 2.0: an accurate tool to identify ortholog groups among multiple genomes.** *PLoS one* 2011, **6(6)**:20892.
16. Li L, Stoeckert CJ, Roos DS: **OrthoMCL: identification of ortholog groups for eukaryotic genomes.** *Genome Res* 2003, **13(9)**:2178-2189.
17. Ostlund G, Schmitt T, Forslund K, Köstler T, Messina DN, Roopra S, Frings O, Sonnhammer ELL: **InParanoid 7: new algorithms and tools for eukaryotic orthology analysis.** *Nucleic Acids Res* 2010, **38(Database)**:196-203.
18. Song N, Sedgewick RD, Durand D: **Domain architecture comparison for multidomain homology identification.** *J Comput Biol* 2007, **14(4)**:496-516.
19. Joseph JM, Durand D: **Family classification without domain chaining.** *Bioinformatics* 2009, **25(12)**:45-53.
20. Frech C, Chen N: **Genome-wide comparative gene family classification.** *PLoS one* 2010, **5(10)**:13409.
21. Liu J, Rost B: **Domains, motifs and clusters in the protein universe.** *Current Opinion in Chemical Biology* 2003, **7(1)**:5-11.
22. Holub J, Smyth WF: **Algorithms on indeterminate strings.** *Proc of AWOCA 2003* 2003, 36-45.
23. Uno T, Yagiura M: **Fast algorithms to enumerate all common intervals of two permutations.** *Algorithmica* 2000, **26(2)**:290-309.
24. Didier G, Schmidt T, Stoye J, Tsur D: **Character sets of strings.** *J Discr Alg* 2007, **5(2)**:330-340.
25. Ciccarelli FD, Doerks T, von Mering C, Creevey CJ, Snel B, Bork P: **Toward automatic reconstruction of a highly resolved tree of life.** *Science* 2006, **311(5765)**:1283-1287.
26. Pesquita C, Faria D, Bastos H, Ferreira AE, Falcão AO, Couto FM: **Metrics for GO based protein semantic similarity: a systematic evaluation.** *BMC Bioinformatics* 2008, **9(Suppl 5)**:4.
27. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: **Basic local alignment search tool.** *J Mol Biol* 1990, **215(3)**:403-410.
28. Lechner M, Findeiß S, Steiner L, Marz M, Stadler PF, Prohaska SJ: **Proteinortho: detection of (co-)orthologs in large-scale analysis.** *BMC Bioinformatics* 2011, **12**:124.
29. Salgado H, Peralta-Gil M, Gama-Castro S, Santos-Zavaleta A, Muñiz-Rascado L, García-Sotelo JS, Weiss V, Solano-Lira H, Martínez-Flores I, Medina-Rivera A, Salgado-Osorio G, Alquicira-Hernández S, Alquicira-

Hernández K, López-Fuentes A, Porrón-Sotelo L, Huerta AM, Bonavides-Martínez C, Balderas-Martínez YI, Pannier L, Olvera M, Labastida A, Jiménez-Jacinto V, Vega-Alvarado L, Del Moral-Chávez V, Hernández-Alvarez A, Morett E, Collado-Vides J: **RegulonDB v8.0: omics data sets, evolutionary conservation, regulatory phrases, cross-validated gold standards and more.** *Nucleic Acids Res* 2013, **41(Database)**:203-13.

doi:10.1186/1471-2164-15-S6-S2
Cite this article as: Doerr *et al.*: **Identifying gene clusters by discovering common intervals in indeterminate strings.** *BMC Genomics* 2014 **15**(Suppl 6):S2.