University of Cyprus

DEPARTMENT OF BIOLOGICAL SCIENCES

DEVELOPMENT OF ALGORITHMS AND SOFTWARE FOR UNRAVELLING THE BIOLOGICAL ROLE OF LOW COMPLEXITY REGIONS IN PROTEIN SEQUENCES

IOANNIS KIRMITZOGLOU

A Dissertation submitted to the University of Cyprus in Partial Fulfilment of the requirements for the degree of Doctor of Philosophy.

JULY 2014

VALIDATION PAGE

Doctoral Candidate: Ioannis Kirmitzoglou

Doctoral Thesis Title: Development of algorithms and software for unravelling the biological role of low complexity regions in protein sequences.

*The present Doctoral Dissertation was submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at the **Department of Biological Sciences** and was approved on the 22th of July by the members of the **Examination Committee**.*

Examination Committee:

Research Supervisor: Vasilis J. Promponas, Assistant Professor, Department of Biological Sciences, University of Cyprus.

Committee Member: Leondios G. Kostrikis, Professor, President of the Examination Committee, Department of Biological Sciences, University of Cyprus

Committee Member: Paris A. Skourides, Assistant Professor, Department of Biological Sciences, University of Cyprus

Committee Member: Christos Ouzounis, Head of the Biological Computation & Process Laboratory, Chemical Process & Energy Resources Institute, Centre for Research and Technology Hellas, Greece

Committee Member: Ioannis Iliopoulos, Lecturer, Department of Basic Sciences, Faculty of Medicine, University of Crete.

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Ioannis Kirmitzoglou

Abstract [in Greek language]

Πολλές πρωτεΐνες είναι εμπλουτισμένες σε περιοχές με ακραία αμινοξική σύσταση. Έχει δειχθεί ότι τέτοιες πρωτεΐνες διαδραματίζουν σημαντικούς βιολογικούς ρόλους αλλά, λόγω των ιδιαίτερων βιοχημικών ιδιοτήτων τους, η μελέτη τους έχει αποδειχθεί δύσκολη μέχρι τώρα. Επιπρόσθετα, τέτοιες περιοχές με χαμηλή πολυπλοκότητα, αποκλίνουν ξεκάθαρα από το μοντέλο τυχαίας σύστασης που χρησιμοποιείται για τον υπολογισμό της στατιστικής σημαντικότητας των αποτελεσμάτων αναζήτησης σε βάσεις δεδομένων πρωτεϊνικών αλληλουχιών. Αυτό έχει ως αποτέλεσμα την παραγωγή (συχνά μεγάλου αριθμού) ψευδώς αληθών αποτελεσμάτων. Έχουν προταθεί πολλές προσεγγίσεις για την αντιμετώπιση αυτού του προβλήματος, αλλά η εύρεση του βέλτιστου τρόπου αντιμετώπισης αυτών των περιοχών εξακολουθεί να είναι αντικείμενο ενεργής έρευνας. Επιπρόσθετα, υπάρχει έλλειψη εργαλείων για την αναζήτηση και απεικόνιση τέτοιων περιοχών χαμηλής πολυπλοκότητας (ΠΧΠ) που θα μπορούσαν να βοηθήσουν σε πιο πολύπλοκες (υπολογιστικά) προσπάθειες για την κατανόηση του βιολογικού ρόλου πρωτεϊνικών αλληλουχιών με ΠΧΠ.

Στα πλαίσια αυτής της διατριβής, ελέγξαμε και επιβεβαιώσαμε την αποτελεσματικότητα όλων των τρόπων αντιμετώπισης των ΠΧΠ που προσφέρονται από το BLAST με την προσθήκη του φιλτραρίσματος της βάσης δεδομένων με τα προγράμματα λογισμικού SEG και CAST. Επινοήσαμε μια εμπεριστατωμένη μέθοδο επαλήθευσης των σωστών αποτελεσμάτων και δείξαμε ότι οι αποτελεσματικότερες μέθοδοι αντιμετώπισης των ΠΧΠ ήταν κάποιες που δεν είχαν αξιολογηθεί μέχρι τώρα. Με βάση τα αποτελέσματα μας, προτείνουμε την εφαρμογή του φιλτραρίσματος τόσο της αλληλουχίας-ερώτημα όσο και της βάσης δεδομένων με το CAST σε όλες τις μεγάλης κλίμακας υπολογιστικές μελέτες συγκριτικής γονιδιωματικής, ειδικά εάν τα σύνολα δεδομένων είναι πλούσια σε ΠΧΠ. Αυτή η προσέγγιση δείχθηκε να μειώνει τις υπολογιστικές απαιτήσεις αυξάνοντας ταυτόχρονα την ευαισθησία της εύρεσης ομολόγων. Προκειμένου να επιταχύνουμε την ευρύτερη υιοθέτηση του CAST από την επιστημονική κοινότητα, αναπτύξαμε μία νέα έκδοση με σημαντικές βελτιώσεις στην ταχύτητα του και με επιπρόσθετα χαρακτηριστικά που κάνουν τη χρήση σε πλήρως αυτοματοποιημένα περιβάλλοντα ακόμα ευκολότερη.

Επιπρόσθετα, αναπτύξαμε τον LCR-eXXXplorer, ένα καινοτόμο διαδικτυακό τόπο με μοναδικά διεθνώς εργαλεία στοχευμένα στους ερευνητές που ενδιαφέρονται για τις ΠΧΠ. Συγκρινόμενος με τις λίγες άλλες παρόμοιες υπηρεσίες, ο LCR-eXXXplorer όχι μόνο προσφέρει στους ερευνητές την δυνατότητα να αναζητήσουν εύκολα και γρήγορα μεταξύ εκατομμυρίων σχολιασμένων ΠΧΠ, αλλά και να τις απεικονίσουν με ένα ευέλικτο και λειτουργικό τρόπο που επιτρέπει την άμεση σύγκριση με σχολιασμούς άλλων βάσεων δεδομένων (όπως η UniProtKB) αλλά και προγνώσεων που σχετίζονται με τις ΠΧΠ.

Χρησιμοποιώντας τα εργαλεία που αναπτύχθηκαν κατά την διάρκεια αυτής τη μελέτης, εξερευνήσαμε την χρηστικότητα γνωρισμάτων που βασίζονται στην ολική και τοπική αμινοξική σύσταση πρωτεϊνών που κωδικοποιούνται από πλήρη γονιδιώματα για την πρόγνωση φαινοτυπικών χαρακτηριστικών των αντίστοιχων ειδών. Συγκεκριμένα, χρησιμοποιήσαμε μια σειρά από πλήρως αλληλουχημένα γονιδιώματα στελεχών του γένους *Escherichia* για την πρόγνωση της παθογονικότητας κάθε στελέχους. Πραγματοποιώντας εκτενείς προσομοιώσεις με τεχνητά σύνολα δεδομένων που προσομοίωναν πρωτεϊνικά υποσύνολα από ημιτελείς μετα-γονιδιωματικές συλλογές, δείξαμε ότι η πρόγνωση της παθογονικότητας με υψηλή ακρίβεια είναι δυνατή ακόμα και με τόσο περιορισμένη πληροφορία.

Τα πρωτότυπα αποτελέσματά μας ανοίγουν νέες κατευθύνσεις για επιπρόσθετη έρευνα τόσο στην ορθή επικύρωση των εργαλείων αναζήτησης ομοιότητας όσο και στην σωστή υλοποίηση μεγάλης κλίμακας πειραμάτων αναζήτησης αλληλουχιών. Επιπρόσθετα, αναμένουμε ότι η προσέγγισή μας για την αξιοποίηση υπογραφών βασισμένων στην αμινοξική σύσταση (πιθανώς ημιτελών) (μέτα-) γονιδιωματικών δεδομένων μπορεί εύκολα να επεκταθεί για να καλύπτει και άλλα είδη και μπορεί να χρησιμοποιηθεί σε σημαντικές εφαρμογές βιοασφάλειας όπως είναι η συνεχής επιτήρηση για ξεσπάσματα επιδημιών.

Η εργασία αυτή χρηματοδοτήθηκε εν μέρει από το ίδρυμα Προώθησης Έρευνας μέσω του προγράμματος ΠΕΝΕΚ/ΕΝΙΣΧ/0308/77.

Abstract

Many proteins are enriched in segments of extreme amino acid compositions. Such compositionally biased proteins are increasingly shown to play important biological roles but, due to their biochemical properties, they have been hard to study so far. In addition, such segments of low complexity clearly deviate from the random model used to evaluate the statistical significance of database search algorithms, leading to the production of (often large numbers of) false positive similarity detection cases. Several approaches have been proposed for addressing this drawback, but finding the most effective way to deal with such regions is still a subject of active research. Moreover, there is a lack of readily accessible tools for searching and visualizing low complexity regions (LCRs) which may facilitate more advanced (computational) approaches for elucidating the biological roles of LCR-containing proteins.

In this work we tested and verified the effectiveness of all LCR-handling methods offered by BLAST with the addition of database masking with CAST or SEG. We devised a comprehensive validation approach and demonstrated that yet untested schemes are the most appropriate for this task. More specifically, we propose that two-way CAST masking should be adopted in large-scale computational comparative genomics studies, especially for datasets with high LCR content. This approach is shown to guarantee the reduction of necessary computational resources (CPU time, storage space) while increasing the sensitivity of homolog detection. To facilitate the wider adoption of CAST, we also developed a new version with significant speed-up improvements and pipeline-friendly features.

Moreover, we developed LCR-eXXXplorer, a novel web-based system with unique properties and features for researchers interested in LCRs. Compared to the few similar services available, LCR-eXXXplorer not only provides researches with the ability to easily and accurately search among millions of annotated LCRs, but also to display them in an attractive and functional fashion, allowing direct comparison with annotations stored in other online databases, such as UniProtKB, and predicted properties commonly associated with LCRs. This system is designed in a modular way, enabling future addition of other datasets or support of additional LCR-detection algorithms.

Using the tools developed in this study, we investigated the utility of global and local compositional features computed from proteins encoded in complete genomes for predicting phenotypic traits of the respective species. More specifically, we used a number of completely sequenced genomes of species/strains of the genus *Escherichia* for predicting the pathogenicity of each strain. By performing extensive simulations with artificial datasets resembling protein subsets from incomplete metagenomic assemblies we illustrate that even with such limited information accurate prediction of pathogenicity is feasible.

Our original findings open new directions for further research both in the proper validation of sequence similarity search tools and in the proper implementation of large-scale sequence search pipelines. Furthermore, we anticipate that our approach for utilizing compositional signatures from (possibly incomplete) (meta-) genomic data may be easily extended to cover other lineages and be directly applicable in important biosafety applications, such as epidemiological monitoring.

This work was funded in part by the Cyprus Research Promotion Foundation through the grant ΠΕΝΕΚ/ΕΝΙΣΧ/0308/77.

Acknowledgments

First and foremost I would like to thank my Research Supervisor Dr. Vasillis J. Promponas for all the guidance and support he provided. He was also the one who came up with the idea of my thesis' subject, which turned out to be both fascinating and original. Vasillis, besides being a brilliant researcher and supervisor, is also a great friend and has always helped me when things got tough. I also owe him some of the funniest memories of my life which, almost always, took place in one of the many national and international meetings and conferences we attended together. I hope our collaboration and friendship will endure for many years to come.

I would also like to thank all the other members of my Examining Committee, Dr. Ioannis Iliopoulos, Dr. Leontios Kostrikis, Dr. Christos Ouzounis & Dr. Paris Skourides for their critical comments and useful remarks on my manuscript and presentation. Their input was invaluable and helped the overall quality of my thesis. Many thanks also go to Dr. Niovi Santama who also gave invaluable comments as a member of my Thesis Proposal Committee but, unfortunately, was unable to attend my *viva voce*.

A big thanks goes to all my colleagues in the lab and especially Athina Theodosiou, Stella Tamana, Ioanna Kalvari and Maria Xenophontos not only for always helping me when in need but also for being part of a tight, friendly and funny team. I wish them the best in their lives and I have no doubts that they will all produce brilliant science.

I would also like to thank the Cyprus Research Promotion Foundation for funding and supporting a big part of this research through a ΠΕΝΕΚ grant (ΠΕΝΕΚ/ΕΝΙΣΧ/0308/77). Hopefully they also feel that their money was spent for a good cause ☺.

During the last year of my PhD I was lucky enough to work as part of a great team led by Dr. George Christophides for a project irrelevant with my thesis. Despite that, George and my colleagues demonstrated a great amount of patience and understating and essentially allowed me to deal exclusively with my PhD for the past 3 months. I am grateful to them because without their help and tolerance I seriously doubt I would have been able to finalize my thesis on time.

I am more than grateful to Marilena Aplikioti, my life partner for the past 11 years. I owe so much to her… her love, beauty, patience, and endurance helped me to keep up with my dream even at times I was sure that I was never meant to become a researcher. Thank you very much Marilena, I will always love you.

Finally, I would like to thank my parents and my brother. Since only my brother speaks and reads English, allow me to write this part in Greek.

Αγαπημένοι μου γονείς και αδερφέ, δεν μπορώ να εκφράσω με λόγια πόσα σας χρωστώ και πόσο σας αγαπώ. Σας ευχαριστώ που ήσασταν πάντα δίπλα μου και δεν πάψατε να πιστεύετε σε εμένα και στο όνειρό μου ούτε ένα λεπτό. Να ξέρετε ότι σας σκέφτομαι ό-που και αν βρίσκομαι. Σας αγαπώ πολύ!.

Ioannis Kirmitzoglou.

TABLE OF CONTENTS

List of Figures

List of Tables

1 Introduction

1.1 Extremes in Local Sequence Composition

The continuous explosion in the amount of biological sequence data stored in public databases (Figure 1-1) requires fast and reliable tools for its proper utilisation. Amongst the most prevalent tools for extracting biological knowledge out of macromolecular sequences are sequence comparison tools, which enable the identification of similarities between a newly described sequence and (possibly annotated) sequences within a database. Those similarities can provide essential information for the detection of homologues, which is the first step towards the (automated) *in silico* inference of functional, structural and evolutionary features of protein molecules (Andrade *et al.*, 1999; Hoersch *et al.*, 2000). Efficient and sensitive algorithms, such as FASTA (Pearson and Lipman, 1988; Pearson, 1990) and BLAST (Altschul *et al.*, 1997, 1990), have been developed to facilitate the rapid execution of similarity searches. BLAST is perhaps the most heavily used mainly because of its speed, sensitivity, and the incorporation of rigorous statistics for the evaluation of hits. Inherent to statistical estimates of significance of database hits, lies the assumption that amino acid sequences are random polypeptides drawn from an underlying distribution (Altschul *et al.*, 1990; Karlin and Altschul, 1990; Karlin *et al.*, 1990).

However, there exist segments of extreme local amino acid composition –also known as compositionally biased or 'low complexity' regions (LCRs) (Wootton, 1994a) – that clearly deviate from the random model used to evaluate statistical significance of hits. Those segments confound the ability of the above methods, leading to the production of erroneous significant (false positives) and insignificant (false negatives) hits between a biased query sequence and some database entries. Furthermore, such low complexity regions were found to be very common. In a seminal paper, Wootton and Federhen (1993) have shown that about half of the protein sequences deposited in UniProt/SwissProt[1] (The UniProt Consortium, 2014) contain such a region; more recently, Golding (1999) demonstrated that LCRs are very frequent within yeast protein sequences, something which is

[1] http://www.uniprot.org/; more specifically, when Wootton and Federhen compiled their results they used a release of the SwissProt database (Bairoch and Boeckmann, 1992).

also true (and more prevalent) for the malaria parasite *Plasmodium falciparum* (Pizzi and Frontali, 2001; Promponas *et al.*, 2000; Tsoka *et al.*, 1999). Thus, it was soon realized that low complexity regions require special treatment, since they tend to produce many spurious hits with specious high scores leading to large and confusing output lists, where hits with a strong biological significance may be buried (Altschul *et al.*, 1994; Wootton, 1994b).

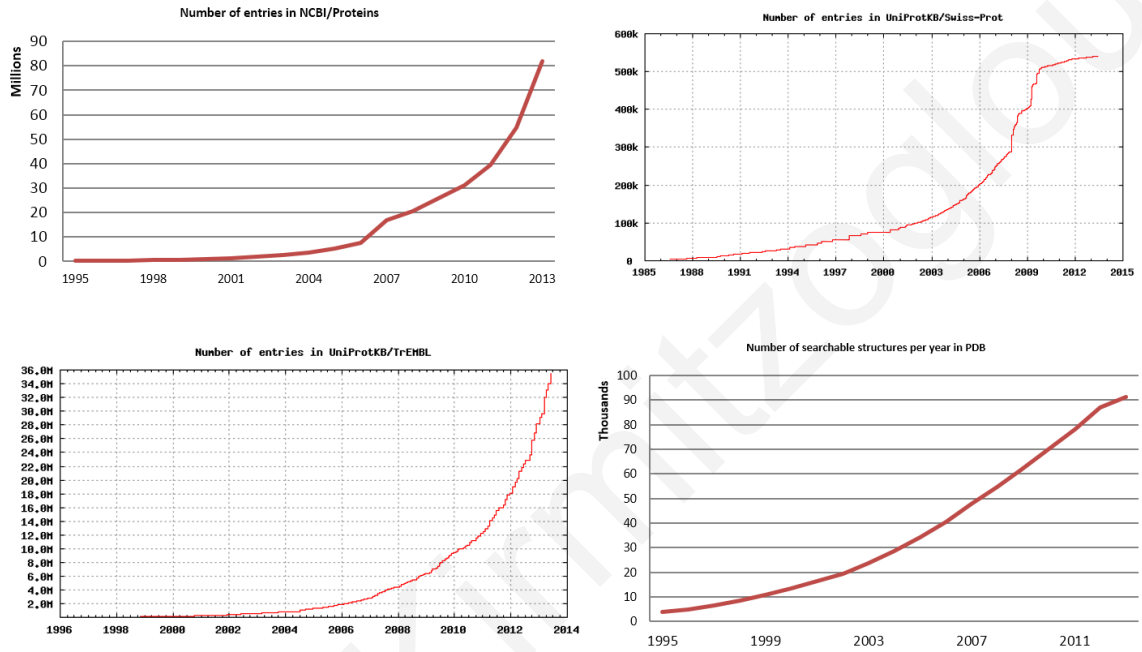


Figure 1-1. *Protein (sequence) databases grow exponentially with time.*
Growth graphs for 4 of the most commonly used public protein (sequence) databases.
Top left: *NCBI/Proteins (source http://www.ncbi.nlm.nih.gov/protein)*
Top right: *UniProt/SwissProt (source http://www.expasy.org/sprot/relnotes/relstat.html)*
Bottom left: *UniProt/TrEMBL (source http://www.ebi.ac.uk/uniprot/TrEMBLstats/)*
Bottom right: *Protein Data Bank*
(source http://www.pdb.org/pdb/static.do?p=general_information/pdb_statistics/index.html)

The surprisingly simple, yet powerful, solution adapted, was to 'mask' those regions of the query sequence with a residue character treated as neutral by database search software ('X' in protein and 'N' in nucleic acid sequences respectively). Detecting such regions though is far from simple and many different formulations of such solutions exist.

1.2 Definitions of Protein Sequence Segments with Low Complexity

1.2.1 First of all, some "history"

The very first algorithms introduced to detect and mask compositionally biased regions were SEG (Wootton and Federhen, 1993) and XNU (Claverie and States, 1993) for protein and DUST (Morgulis *et al.*, 2006; Tatusov and Lipman, unpublished) for nucleotide sequences. SEG and DUST were incorporated in the BLAST suite of programs since 1992 (Wootton, 1994b) and were used as the default option until recently, when they got replaced by compositional based statistics (CBS) (more info in section 1.4 below). Many more algorithms exist today for this purpose (see section 1.3) and –while some of them are quite similar– they deal with the problem of LCR detection in different ways, thus producing varying results. Importantly, benchmarks have been carried out in different contexts. For example, CAST seems to be a better choice than SEG for masking protein sequences prior to database search where detection specificity plays an important role (Kirmitzoglou and Promponas, 2007; also this work; Kreil and Ouzounis, 2003; Promponas *et al.*, 2000). On the other hand BIAS seems to be more sensitive than CAST and SEG in the detection of compositionally biased regions rich in specific, user-defined, residues (Kuznetsov and Hwang, 2006).

1.2.2 Definitions

Karlin and Altschul were the first who examined the problems caused in sequence similarity searches by the presence of *"unusual patterns in a nucleic acid or protein sequence"* (1990). The reported patterns involve clusters of either heavily charged, hydrophobic or cysteine residues. Wootton and Federhen (1993) used a well-established information theory measure, the Shannon entropy (Shannon, 1948) as a proxy to the more expensive computation of combinatorial complexity, to define and identify regions with unusual amino acid composition which they termed as *"low complexity"*, since Shannon entropy is a measure of the information encoded in a signal[2]. About the same time Claverie and States (1993) dealt with the artefacts introduced into database search results by regions

[2] Actually, there were previous works introducing information theoretic measures as an estimate of biological sequence complexity – e.g. Salomon and Konopka (1992), whose approach was only used for the study of nucleotide sequences.

containing internal or intrinsic repeats; they described the latter as *"low complexity"* or *"low entropy"* segments. While both of the above definitions share the same name, the one by Wootton and Federhen is broader and covers most of the regions reported by the methods presented in section 1.3 below. Furthermore, their definition is well established due to the use of a robust mathematical framework[3].

In any case, many definitions were given for sequences that match the criteria of *"low complexity",* even though many of the definitions use neither complexity nor entropy measures to describe these sequences. In the following paragraphs we will try to summarize the most important such definitions that exist in the literature, especially those accompanied with relevant software.

Low Complexity Regions

The most common definition used for these amino acid clusters is *"Compositionally Biased Regions"* or *"Low Complexity Regions".* The basic principle, first described by Wootton and Federhen (1993), is that complexity can be derived by the vector representation of the compositional state of a sequence segment. Let us assume the following hypothetical protein sequence as an example:

ASSWSARYILST

this sequence is $\mathrm{L}=12$ residues long and its state complexity vector is defined as

$$(4,2,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0)$$

which contains the counts of amino acid types of the example sequence in descending order. Notice that the sum of all the numbers in the vector equals the length of the sequence. Using this vector as a starting point we can define two measures that describe the complexity of any subsequence:

$$K_1 = \frac{1}{L} log_{20}\left(\frac{L!}{\prod_{i=1}^{20} n_i!}\right) \qquad (1\text{-}1)$$

[3] Strictly speaking, many of the algorithms presented in this work do not detect "Low complexity regions" as defined by Wootton and Federhen. Despite this, due to the popularity of the term among researches of the field, we will refer to all such detected regions as LCRs, unless otherwise noted.

$$K_2 = -\sum_{i=1}^{20} \frac{n_i}{L}\left(\log \frac{n_i}{L}\right) \qquad (1\text{-}2)$$

where K_1 is the local compositional complexity of the sequence (or subsequence), K_2 its compositional entropy (or Shannon entropy) and n_i are the 20 counts in the complexity vector[4]. A third useful measure is the probability P_0 for the occurrence of a specific complexity state. In the case of equal prior probabilities for the appearance of the 20 amino acids, P_0 is:

$$P_0 = \frac{1}{20^L}\left(\frac{L!}{\prod_{i=1}^{20} n_i!}\right)\left(\frac{20!}{\prod_{k=0}^{L} r_k!}\right) \qquad (1\text{-}3)$$

where r_k are the counts of the number of occurrences of each number in the complexity state vector.

It is clear that K_1, K_2 and P_0 are all functions of the complexity state vector; this in turn means that many different protein sequences can give equal complexity measures as long as their vectors are the same (e.g. sequences `AAWSTR`, `IILAYD` and `CKILLQ`).

Moving a step further, Kuznetsov & Hwang (2006), divided the general term of compositional bias into global and local compositional bias. The term of global bias is referring to the entire protein sequence that is consisting of a large amount of specific residue types and becomes one large compositionally biased segment. On the other side, the local bias is more about protein sequences that their global composition conforms the random independence model but contain local clusters of over or under-represented residue types.

Intrinsically Disordered proteins

Another term that is frequently used to describe proteins sequences rich in LCRs is *"intrinsically disordered proteins"*. The term refers to proteins that (globally or partially) fail to fold into a fixed three-dimensional (3D) structure. This may be the case for segments or

[4] In fact, K_2 was introduced in SEG as an approximation of K_1, since computation of factorials was very demanding with the computers available in early 90s. Within the SEG software logarithms were tabulated and approximated by interpolation, so actually SEG performed a fast lookup to compute K_2.

even the entirety of the protein chain. Often, these chains do fold into a fixed structure upon interaction with another protein (Romero *et al.*, 2001; Schnell *et al.*, 2007). Nevertheless, while it is true that LCRs are frequently associated with protein disorder, *"intrinsically disordered proteins"* and *"low complexity regions"* are not equivalent terms and one can exist without the other (Romero *et al.*, 2001). Well documented exemplary cases are the reports demonstrating that extended Q/N-rich regions lead to the formation of amyloids –i.e. self-propagating, *β*-sheet-rich, fibrillar protein aggregates associated with a number of diseases (Koo *et al.*, 1999; Michelitsch and Weissman, 2000).

Tandem, Cryptic and Interspersed repeats

It is quite common for protein and nucleotide sequences to contain segments that are identical (or near identical) and are repeated throughout the protein sequence in regular or irregular patterns. Such repeats are commonly associated with specific functional and structural properties and may indeed lower the complexity of the sequence either globally and/or locally.

The three major categories of protein repeats (Albà *et al.*, 2002; Li and Kahveci, 2006) are (all examples contain dashes for easier comprehension of the repeats structure):

i. **Tandem repeats**, as the name implies, are identical subsequences that occur consecutively in protein sequences (e.g. ADP-ADP-ADP-ADP*)*. Tandem repeats are often detected as LCRs, something that depends on the complexity and length of each subsequence and the number of times is repeated.
ii. **Interspersed repeats** are similar to tandem repeats but they are not consecutive, i.e. there are at least two identical subsequences separated by a non-repeat segment (e.g. ADP-K-ADP-LSN-ADP).
iii. **Cryptic repeats** may be tandem or interspersed repeats of subsequences that are similar but not identical. Since similarity heavily relies on the method and the scoring scheme used to calculate it, different algorithms may not detect the same sequence segments as cryptic repeats. Whether or not cryptic repeats can also be LCRs depends not only on the complexity, length and the number of the repeated sub-segments, but also on their similarity (the more similar the sub-segments, the higher the chance of being detected as LCRs). If the substitution matrix used to calculate similarity is BLOSUM62

(Supplementary Figure 1) (Henikoff and Henikoff, 1992), then a typical example for this category would be **QDH**-K-**EDH**-LS-**QDY** (since substitutions for Q to H and H to Y both give a positive score of 2).

In any case, it should be noted that a large fraction of the repeats present in protein sequences (especially the longer ones) are not of low complexity and thus these two terms cannot be used interchangeably.

Simple Sequences

Another widely used definition for LCRs, is *"simple sequences"* which are perfect repeats of a single amino acid type. These homopolymers are primarily composed by specific amino acid types such as *Q, N, S, T, P, H, G, A, D* and *E*, and are usually shorter than 20 residues long (Huntley and Golding, 2002). Because of their inherent low complexity most of the simple sequences in protein sequences are also detected as LCRs by algorithms depending on complexity measures such as SEG. However, *"simple sequences"* cannot describe the full spectrum of LCRs and therefore can only be considered a subset of them (Albà *et al.*, 2002; Sim and Creamer, 2004).

Linguistic Complexity

Linguistic complexity is considered as a quite different measure of compositional complexity since is mostly defined by the structural (in terms of words and vocabulary, not 3D-structure) complexity of the sequence and not by its composition (Pietrokovski *et al.*, 1990; Popov *et al.*, 1996). Despite having been defined primarily to describe human-written text, linguistic complexity can be applied to any sequence of characters, including nucleotide and protein sequences. Two of the most obvious similarities between human and biological texts (sequences) are their linear structure and the fact that both contain certain letter combinations more frequent than others in a repetitive pattern (Popov *et al.*, 1996). Contrary to human text though, biological sequences are much more repetitive (Troyanskaya *et al.*, 2002), thus sub-segments of high linguistic complexity usually indicate the presence of a biological signal.

In that sense, linguistic complexity is better suited to detect segments of biological sequences that are of unusually high structural complexity, compared to the rest of the sequence, and thus biologically potentially interesting. Still, several studies (e.g. Pietrokov-

ski *et al.*, 1990; Popov *et al.*, 1996; Troyanskaya *et al.*, 2002) have demonstrated that there is often a direct connection between subsequences with low linguistic complexity and low compositional complexity (i.e. entropy). As such, linguistic complexity is another measure that can be used to detect probable LCRs and such formulations have been described (e.g. ScanCom; Nandi *et al.*, 2003a).

1.3 Algorithms Capable of Detecting LCRs

Over the years researchers have developed a large number of algorithms for the detection and, most of the times, masking of low complexity regions (Table 1). We present, in chronological order, the most important ones in terms of usage and algorithmic approach/diversity. While the assessment of the importance of each method is largely a subjective process, we tried to use some objective criteria to select the methods presented herein. Such criteria include novelty, performance, adoption by the scientific community and –crucially– availability.

1.3.1 SEG (Wootton and Federhen, 1993)

SEG has been the golden standard for identifying and masking low complexity regions since the early- to mid-1990's, when it was incorporated into the BLAST suite of programs as a default option. Its sensitivity along with its speed has facilitated the elimination of the vast majority of spurious hits produced by BLASTP, practically enhancing its sensitivity. Because of the measures it uses [equations (1-1) to (1-3)] it can detect most of the non-random regions in protein sequences including homopolymers, short period repeats and aperiodic mosaics of few residue types. SEG is a two stage window based algorithm that accepts 3 user-defined parameters: the initial window length $L(1)$ and two complexity thresholds; the trigger K_t and the extension complexity K_e.

During the first stage SEG identifies "trigger windows" or seed segments of length $L(1)$ and of K_2 complexity smaller or equal to the trigger complexity K_t. After merging overlapping trigger windows, SEG extends those merged "raw segments" to both directions by including any window of length $L(1)$ that has complexity $K_2 \leq K_e$ until no further extension is possible. Equation (1-2) is preferred at this stage as an approximation of equation (1-1) for computational efficiency. In the second stage SEG reduces the above segments to *"optimal low complexity segments"*, i.e. sub-segments with the lowest occurrence

probability P_0. For this purpose it uses equations (1-1) and (1-2) which are well suited for optimization of sequences with different lengths. As an optional step, SEG can mask the aforementioned segments prior to a similarity search to reduce the number of produced spurious hits.

While SEG performs notably well in most of the cases, its usage as a default option prior to BLASTP searching until recently, has been the source of scepticism by various authors (Kreil and Ouzounis, 2003; Kuznetsov and Hwang, 2006; e.g. Promponas *et al.*, 2000; Wan *et al.*, 2003). Most of them agree that SEG masks significantly more residues than those really needed to eliminate spurious hits and in some cases it even masks functionally important regions that would lead to the detection of true homologues (e.g. Koonin *et al.*, 1996). This can be mainly attributed to the origins of SEG as a tool to analyse the local complexity of proteins sequences and to automatically identify non-globular protein domains (Wootton, 1994a, 1994b), both being applications that favour sensitivity of detection upon specificity. Furthermore, by design, SEG masks whole segments rather than the amino acid residues that are responsible for the computed low complexity of a specific region; by doing so it removes potentially crucial information for homolog detection. Another target of criticism was the use of equal priors for all residue types in combination with a segmentation threshold derived by random sequences. This decision is justified by the composition of low complexity regions which is very different from the general composition of protein databases (Wootton and Federhen, 1993). It also has some important consequences though, most notably the independence of detection from the sequence attributes derived from its amino acid types composition (Wan *et al.*, 2003) and the inability to estimate measures of statistical significance for the detected LCRs (Kuznetsov and Hwang, 2006). Finally, the usage of the sliding window makes SEG strongly biased in the detection of LCRs with lengths similar to the window length (Kreil and Ouzounis, 2003) which is typically short and by default equal to 12.

1.3.2 CAST (Promponas *et al.*, 2000)

With the aim of filtering amino acid sequences for enhanced database searches, Promponas and colleagues built on the *"biasdb"* algorithm (Andrade *et al.*, 1999) to formulate a more intuitive masking algorithm in the context of database search. More specifically, an artificial database of homopolymeric amino acid sequences of all twenty types

is searched against by any sequence of interest to identify potential regions of extreme compositional bias. Apparently, using a local similarity search algorithm along with a proper scoring scheme, sequence segments with one or few amino acid residues in excess (thus deviating from the background amino acid distribution) will achieve significant similarity scores and thus may be characterized as compositionally biased. A careful formulation of the problem, enabled the development of an iterative procedure based on a local alignment dynamic programming algorithm with infinite gap penalties (*detection step*) followed by masking the most significant region scoring above a user defined threshold (*filtering step*).

Assume an amino acid sequence $R = r_1 r_2 .. r_i .. r_n$ (query) and a substitution matrix M with elements m_{α,r_i} for scoring the match of two residues of type α and r_i respectively. Then the dynamic programming recursion for computing the score for the match of the query sequence up to the residue r_i and the homopolymer of type α becomes

$$s_\alpha^i = m_{\alpha,r_i} + \begin{cases} s_\alpha^{i-1}, & \geq 0 \\ 0, & < 0 \end{cases} \qquad (1\text{-}4)$$

Any stretch of the query sequence with positive scores extending from a position with zero score to a locally maximal one is a candidate region. Applying this algorithm for all residue types, CAST can identify the stretch with the overall maximum score, which is destined for filtering provided the threshold criterion is fulfilled. The detection step conceptually detects similarity of a sequence region (compositionally biased region) to a single homopolymer type (type of composition bias), thus enabling 'surgical' masking of the query sequence, which seems to preserve useful information for more sensitive database searches (Kreil and Ouzounis, 2003; Promponas *et al.*, 2000; Tsoka *et al.*, 1999). Importantly, the compositional bias type may be used to characterize and crudely classify regions with unusual composition detected by CAST. By definition, CAST detects local composition bias; however, compositionally biased segments often span regions with dozens or even hundreds of residues, surpassing the *"fixed window"* limitation of SEG.

While CAST enabled BLASTP searches with increased sensitivity and specificity when compared to SEG, it is not flawless. A major point of criticism has always been the slow performance of the algorithm, which can be mainly attributed to the computationally naïve implementation of the iterative local similarity search adopted by its original develop-

ers. Furthermore, since CAST's internal searches against the database of homopolymers always happen with a specific order *and* after the protein sequence has been masked from the previous searches, artefacts can occur (e.g. CAST may only mask residues of type S in a region rich both in S and T because the evaluation of the score against poly-T happened after all the residues of type S in the region were masked; since the pair S-T produces a positive score[5] but the pair X-T a negative one, masked residues will lower the total score of the region against poly-T and thus CAST may fail to detect it). To address the criticism around performance and to speed up the masking of data-sets containing hundreds of thousands of sequences[6], we have created a newer version with vastly improved performance; CAST v2.1 is presented in section 3.3 of this manuscript.

1.3.3 0j.py (Wise, 2001)

0j.py (Wise, 2001) measures the complexity of a protein sequence by encoding it into regular expressions. By doing that it transforms the sequence into a compressed representation; the lower the complexity of the sequence the higher the compression. Following this paradigm, 0j.py uses compressibility as a measure of complexity.

To better understand how 0j.py works let us consider a protein sequence A (dashes are added for readability):

$$\mathrm{A{=}\mathbf{ADP}\text{-}\mathbf{ADP}\text{-}KK\text{-}\mathbf{ADP}\text{-}L\text{-}SSSS\text{-}N\text{-}\mathbf{ADP}}$$

It is obvious that A contains 2 tandem and 2 interspersed repeats of ADP, as well as 2 simple sequences, namely KK and SSSS. Assuming that the resulting regular expressions can contain tokens that can either be a letter (for amino acids) or a number (for a dictionary entry) then the regular expression would be (again with extra dashes):

$$\mathrm{A'{=}\mathbf{1^2}\text{-}K^2\text{-}\mathbf{1}\text{-}L\text{-}S^4\text{-}N\text{-}\mathbf{1}}$$

with a dictionary containing just one entry

$$1\text{: ADP}$$

[5] According to BLOSUM62 S-T pairing has a score of 1 while X-T a score of -1, see also Supplementary Figure 1.

[6] Such as the ones used by LCR-eXXXplorer, see also section *"Description of LCR-eXXXplorer"* at page 68.

Superscripts are used to denote the number of times each token is repeated. The total length of the regular expression is calculated by adding all the tokens from the expression (in this case 7) as well as the dictionary (3). The final score is calculated by subtracting the total length of the regular expression (10) from the length of the original sequence (20). Obviously, higher scores indicate higher compressibility and thus lower complexity.

0j.py is unique because it provides researchers with a list (the dictionary) of sub-segments that are repeated throughout the sequence. It is also quite sensitive in detecting repeats or simple sequences that are as short as 2 residues. On the other hand, the final score describes the whole sequence and therefore can be considered as a measure of global instead of local bias. Therefore, 0j.py cannot be used to effectively mask protein sequences prior to database searches.

1.3.4 DSR (Wan *et al.*, 2003)

Shannon entropy is a measure derived by the complexity state vector of a sequence (see section *"SEG* (Wootton and Federhen, 1993)*"* above). That characteristic makes it inappropriate for the comparison of sequences with significantly different sizes since it does not take into account the sequence length (e.g. sequences AAAAT, ATAAA and $\mathrm{LLLLLLLLTT}$ all have the same entropy of $K_2 = 0.167$). In order to overcome that limitation Wan and colleagues defined a new measure derived directly from Shannon entropy, which they named *"reciprocal complexity"*

$$C(S) = \left(\frac{1}{n}\right)^{L} \prod_{i=1}^{n} \left(\frac{L}{v_i}\right)^{v_i} \qquad (1\text{-}5)$$

where n is the count of letters in the alphabet used (20 for proteins), L is the sequence length and v_i the number of occurrences of each residue type in the complexity state vector. Reciprocal complexity takes sequence length into account and allows for comparison of LCRs of substantially different size. A good example, pointed by the authors of DSR, is the case of the DNA sequence $\mathrm{GTGTGTAC}$ in which the obvious longest *"simple segment"*, and thus the one with the lowest complexity, should be GTGTGT. Using Shannon entropy it is impossible to identify GTGTGT as the longest simple sub-segment while using reciprocal complexity it is. In addition, DSR takes the (amino acid) composition of the database into account by incorporating scoring schemes (which are actually substitution

matrices) in the calculation of the reciprocal complexity. Thus equation (1-5) now becomes

$$C^*(S) = L\log L - L\log n - \sum_{i=1}^{n} u_i^* \log u_i^* \qquad (1\text{-}6)$$

where u_i^* represents a normalized version of the scoring scheme or substitution matrix in use. The incorporation of scoring schemes also allows for a software that can be fine-tuned to identify LCRs of specific compositional features (e.g. hydrophobicity), while at the same time being able to detect LCRs in a totally unbiased way (like SEG does) with the usage of a scoring scheme that assumes equal probabilities. Wan *et al.* (2003) suggested that the main usage of DSR should be *"the masking of simple sequences for searching databases"* and demonstrated some test cases of heavily biased sequences that revealed a clear advantage of DSR over SEG in the elimination of spurious hits returned by BLASTP. When it comes to efficiency, both algorithms perform comparably, something expected since DSR employs the same 2-step procedure already described for SEG. Nevertheless, DSR has not been extensively used by the scientific community, compared to its predecessor.

1.3.5 CARD (Shin and Kim, 2005)

The CARD algorithm is based on the complexity analysis of subsequences delimited by a pair of identical repeating subsequences (Shin and Kim, 2005). Given a protein sequence, CARD calculates the suffix tree of the sequence[7], it then searches for subsequences that are positioned in tandem or overlap each other; the region containing them is marked as a candidate LCR. In subsequent passes CARD searches for segments of the sequence that are located between two identical subsequences R. It then splits the surrounded segment to sub-segments of length L equal to the length of R and calculates the Shannon entropy for each of them. If *each and every one* sub-segment has lower entropy than the left and right R subsequences it marks the segment spanning from the start of the left to the end of the right R as an LCR. The procedure is repeated until no more LCRs can be detected.

[7] Suffix trees are special data structures for computing all repeating substrings in a sequence. They are fast to build and search but at the expense of much larger storage, compared to the sequence itself.

1.3.6 BIAS (Kuznetsov and Hwang, 2006)

BIAS (Kuznetsov and Hwang, 2006) is a recent addition to the arsenal of methods designed to detect and mask LCRs. As such, the developers of BIAS had the opportunity to study and try to overcome the drawbacks of existing methods. One such major drawback was that existing widely used algorithms treated all the amino acid residue types equally when searching for LCRs. In other words, the user was not able to target the search to LCRs formed by specific residue types[8]. BIAS addresses that issue by allowing the user to define custom alphabets, which then scans for over- (or under-) presentation in the protein sequence with the use of discrete scan statistics.

In summary, what BIAS does is, for each residue type belonging to the user's alphabet, encode into a binary vector its presence (1) or absence (0) for every position in the sequence. In the next step, closely located clusters of 1s are merged and then using a sliding window approach, the significance of each detected cluster is calculated against the random independence model. All clusters with a *p*-value below a used defined threshold (default is 0.05) are marked as compositionally biased and can, optionally, be masked out of the protein sequence.

Alternatively, BIAS can be instructed to provide an exact estimate of the global compositional bias of the protein sequence and compare each cluster against it, thus detecting only regions that are biased against the composition of the sequence and not the random independence model. Notice that we are purposely avoiding the use of the term LCRs since regions detected by BIAS can either be of unusually low *or* high complexity, which is another major feature of this algorithm.

In terms of the amount of biased regions detected, when compared against SEG and CAST, BIAS holds a middle ground by simultaneously avoiding the excessive masking of SEG and the, often picky, surgical nature of CAST (Kuznetsov and Hwang, 2006).

However, BIAS's main advantage is also its main drawback; it will always ignore biased regions composed of residue types that are not included into the user supplied alphabet

[8] It should be noted that in the case of CAST, a user could construct special substitution matrices fine-tuned for the detection of LCRs of specific residue types. While this solution is not as straightforward as using BIAS it could produce similar results.

which, in the context of sequence database searches, may cause severe issues with spurious hits.

1.3.7 LPS (Harbi *et al.*, 2011; Harrison and Gerstein, 2003; Harrison, 2006)

The LPS algorithm (Harrison and Gerstein, 2003; Harrison, 2006) and the LPS-annotate[9] server (Harbi *et al.*, 2011; which is where the method is formally described) detect regions of compositional bias by probability minimization. Such least likely subsequences are termed LPSs (Lowest Probability Subsequences) and in a similar fashion to CAST, detected LPSs are also characterized by one or more residue types. Furthermore, detected LPSs are accompanied by a *p*-value helping researchers assess their statistical significance. Finally, LPS supports the use of *"composition files"* (constructed by simple FASTA files) which can be used as the distribution against which the LPS performs the probability minimization. Therefore, detected LPSs can be tuned to have the lowest probability in the context of a user-supplied database composition. LPS can also mask biased regions out of the protein sequence.

The main drawbacks of LPS are **(i)** its complexity; detecting biased regions requires multiple steps from the user and **(ii)** slow performance; this can be attributed to the fact the LPS is an iterative, sliding window algorithm and for each input sequence a decreasing series of window sizes, typically from 500 to 1, is used.

Very recently (Antonets and Nizhnikov, 2013) a different research group created a novel implementation of the same LPS algorithm. SARP has the exact same accuracy with LPS and the detected biased regions are identical but it is about 230-fold faster than the original implementation, bringing it up to speed with faster methods such as SEG and CAST.

1.3.8 Others

Several other formulations have been proposed throughout the last fifteen years to detect "LCRs", most of them directly or indirectly based on the concept of detecting internal repeats (perfect or imperfect). Table 1 lists some of those methods, along with the ones presented above, with the aim of covering the most diverge algorithmic approaches.

[9] More details in section *"Available services to search and visualize LCRs"*, page 19.

Table 1. *Methods for detecting regions of non-standard local amino acid composition in protein sequences.*

Name	Authors	Access
SAPS	(Brendel *et al.*, 1992)	Web[10]
XNU	(Claverie and States, 1993)	On request
SEG	(Wootton and Federhen, 1993)	Open[11]
CAST	(Promponas *et al.*, 2000)	Web[12]; On request
0j.py	(Wise, 2001)	On request
SIMPLE	(Albà *et al.*, 2002)	Web[13]; On request
DSR	(Wan *et al.*, 2003)	On request
ScanCom	(Nandi *et al.*, 2003a)	On request
CARD	(Shin and Kim, 2005)	Offline[14]
BIAS	Kuznetsov & Hwang, 2006	Web[15]; On request
GBA	(Li and Kahveci, 2006)	On request
LPS	(Harbi *et al.*, 2011; Harrison and Gerstein, 2003; Harrison, 2006)	Web[16]
SubSeqer	(He and Parkinson, 2008)	Web[17]
SARP	(Antonets and Nizhnikov, 2013)	On request

1.4 Composition Based Statistics (Schaffer *et al.*, 2001; Yu and Altschul, 2005)

Masking LCRs prior to database searches, especially with SEG, often had the side-effect of removing potentially useful information from the query sequence that could negatively impact the quality of the results. Schaffer *et al.* (2001) formulated a solution to this problem: instead of trying to alter the composition of the query sequence in an effort to

[10] http://brendelgroup.org/bioinformatics2go/bioinformatics2go.php
[11] ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/
[12] http://athina.biol.uoa.gr/cgi-bin/CAST/cast_cgi
[13] http://www.biochem.ucl.ac.uk/bsm/SIMPLE/
[14] http://bioinfo.knu.ac.kr/research/CARD/
[15] http://lcg.rit.albany.edu/ProBias/
[16] http://cedra.biol.mcgill.ca/LPS/lps-annotate.html
[17] http://compsysbio.org/subseqer/

match the composition of the database (which is essentially what happens when masking LCRs), they rescale the scoring matrix to match the composition of the sequence. Few years later, Yu & Altschul (2005) proposed a novel method which also scaled the scoring matrices to match the compositions of the query and database. This method usually yielded better results, especially in cases where the compositions differed by a wide margin. To achieve that, Yu & Altschul rescaled each element of the substitution matrix individually (at the cost of computational time) as opposed to the global scaling used by Schaffer and colleagues. Since in most cases the global scale works just as well as the localized one, BLASTP by default employs global scaling except under certain specific conditions for which it has been empirically determined to be beneficial to individually scale each matrix element (Altschul *et al.*, 2005).

1.5 Effects of LCRs and LCR-handling tools in protein database search

Sequence database searches are undoubtedly one of the most widely used bioinformatics applications, by both computational and wet-lab biologists. A lot of research effort has been devoted since the early 1970's in the development of sophisticated algorithms and computer software that facilitates pairwise sequence comparison (Needleman and Wunsch, 1970; Smith and Waterman, 1981). In particular, the BLAST (Altschul *et al.*, 1990) suite of applications has had a tremendous impact on modern molecular biology, since it has enabled the fast and relatively accurate comparison of sequences of biological macromolecules against ever increasing databases, making a significant contribution to the entrance in the genomics era.

BLAST's major advantage (apart from speed gained by smart heuristics) is the incorporation of robust statistical significance estimates of acquired hits. The calculation of statistical significance estimates by protein-protein BLAST (BLASTP) relies heavily on the amino acid composition of the database searching against (Karlin and Altschul, 1990). LCRs (and obviously other compositionally biased segments) clearly deviate from the aforementioned distribution, thus making the statistics employed for significance estimation unreliable (Wootton, 1994b).

We already described several techniques that have been proposed to identify such regions of extreme local composition in sequences and possibly mask them prior to data-

base searches, thus drastically reducing the number of false positive hits. Query sequence masking with SEG has been for years the golden standard in database searches and several studies have pointed out the advantages of this approach (e.g. Altschul *et al.*, 1994; Forslund and Sonnhammer, 2009; Harrison and Gerstein, 2003; Koonin *et al.*, 1996; Kreil and Ouzounis, 2003; Promponas *et al.*, 2000). Surprisingly though very few performed wide-scale benchmarks to verify the effectiveness of query masking itself (Forslund and Sonnhammer, 2009; Koonin *et al.*, 1996) with most authors comparing masking methods to each other and pointing to the same article, by Altschul *et al.* (1994), when discussing the necessity for masking. Database masking (i.e. masking the protein sequences of the database in combination or instead of masking the query sequence) was discussed only in one study by Schaffer *et al.* (2001). Despite the authors reporting increased accuracy when using PSI-BLAST against masked databases, we couldn't locate any follow up studies or mentions of this approach in the literature.

More recent versions of BLAST introduced composition-based statistics and compositional score matrix adjustments (Schaffer *et al.*, 2001; Yu and Altschul, 2005), which replaced SEG masking as the default option. While the superiority of these (and similar) approaches over query masking with SEG are well documented (Altschul *et al.*, 2005; Coronado *et al.*, 2006; Forslund and Sonnhammer, 2009; e.g. Schaffer *et al.*, 2001; Wan *et al.*, 2003) we are not aware of any studies comparing them with other masking algorithms or with the concept of database masking.

Furthermore, a fundamental issue in benchmarking bioinformatics methods is the existence of *"golden standard"* data sets, a role typically filled by ASTRAL (Chandonia *et al.*, 2004), a collection of sequences with known structural, functional and evolutionary relationships based on the classification of PDB[18] (Berman *et al.*, 2000) sequences by SCOP (Andreeva *et al.*, 2007; Murzin *et al.*, 1995). However, not all authors agree that ASTRAL is a proper data-set to test the performance of LCR-handling algorithms, since is derived from PDB, a database known to be poor in LCRs (Wootton and Federhen, 1993). Consequently, most of the benchmarks are performed with drastically different data-sets making it very difficult to compare the results. Recently, Forslund & Sonnhammer proposed a

[18] http://www.rcsb.org/

standardized procedure for the creation of data-sets based on Pfam[19]-domains (Finn *et al.*, 2010) and whole proteomes that are much more suited to the task of benchmarking LCR-handling methods (Forslund and Sonnhammer, 2009). The problem is that the proposed procedure is complex, computationally intensive and must be executed by the user; i.e. there is no web-service that offers pre-compiled data sets for download. This is in sharp contrast with ASTRAL, which makes it very easy to download current and past data-sets in just a few clicks, a fact that certainly helped raise its popularity among bioinformaticians.

1.6 Available services to search and visualize LCRs

LCRs are known to cause severe issues to several methods key for modern bioinformatics research; among these are sequence database search (Altschul *et al.*, 1994), multiple sequence alignments (Mistry *et al.*, 2013) and motif discovery (Bailey, 2008). For that reason, several databases and services (Bradshaw *et al.*, 2006; Di Domenico *et al.*, 2014; Harbi *et al.*, 2011; Hunter *et al.*, 2012; Rost *et al.*, 2004; Sonnhammer and Wootton, 2001; Ye *et al.*, 2006) pre-annotate stored sequences with detected LCRs (most commonly using SEG) or related features. These annotations are often presented graphically along other important sequence regions like motifs, PFAM (Finn *et al.*, 2010) domains or secondary structure. On the other hand, only a small number of services allow the researcher to specifically search for protein sequences containing particular types of LCRs [LPS annotate (Harbi *et al.*, 2011), HRap (Lobanov *et al.*, 2014) and RepeatsDB (Di Domenico *et al.*, 2014)] but present other issues that potentially limit their utility. For a detailed review of such services please refer to section "6.1. Services Similar to LCR-eXXXplorer" of the Supplement.

The only currently available service developed to search and visualize LCRs is LPS-annotate; a web-service for *"the annotation of compositionally-biased regions, and searching for similar regions in other sequences"* (Harbi *et al.*, 2011; Harrison and Gerstein, 2003). Protein sequences contained in UniProt/SwissProt are annotated for lowest-probability subsequences, a type of compositional bias (CB) previously described (Harri-

[19] http://pfam.xfam.org/

son and Gerstein, 2003) using a software called LPS. These biased regions are then searchable by CB signatures, which are actually strings of amino-acid residue types that characterize an LCR. Depending on the search mode, searching for a CB signature of "PR" will either return (i) LCRs with all possible permutations of Proline and Arginine (mode "*All*"), (ii) LCRs rich only in P and R (mode "*Exact*") or (iii) LCRs with a predominant type of P (mode "*First*").

Matching regions are displayed in a simple HTML view (**Error! Reference source not found.**) along with the original protein sequence, the same sequence masked by SEG and regions predicted as disordered by DISOPRED (Buchan *et al.*, 2010). Results also contain the predominant bias type, the count of biased residues, the binomial P-value of the biased region and the mean protein-disorder propensity of the sequence. Finally, LPS-annotate indicates whether a sequence originally contained multiple biased regions that were merged in a larger one.

LPS-annotate also allows a user to search by UniProt/SwissProt ID or Accession number, perform a BLAST search against the annotated UniProt/SwissProt database or search for biased regions in an arbitrary protein sequence of choice.

One major drawback of LPS-annotate is the lack of a more descriptive visual representation; the plain text/HTML view works well for saving the results in a text file but is also hard to interpret and navigate. Also, the addition of more sequence features (such us protein domains, binding sites, sequence topology, GO terms) would have made the results and the whole service much more useful for the discovery of the function or other properties of LCRs. This also holds true for the services provided, which are limited to search and downloads. Finally, the underlying database has not been updated since 2009 when the service was originally published, although the option to annotate a custom sequence with biased regions works fine.

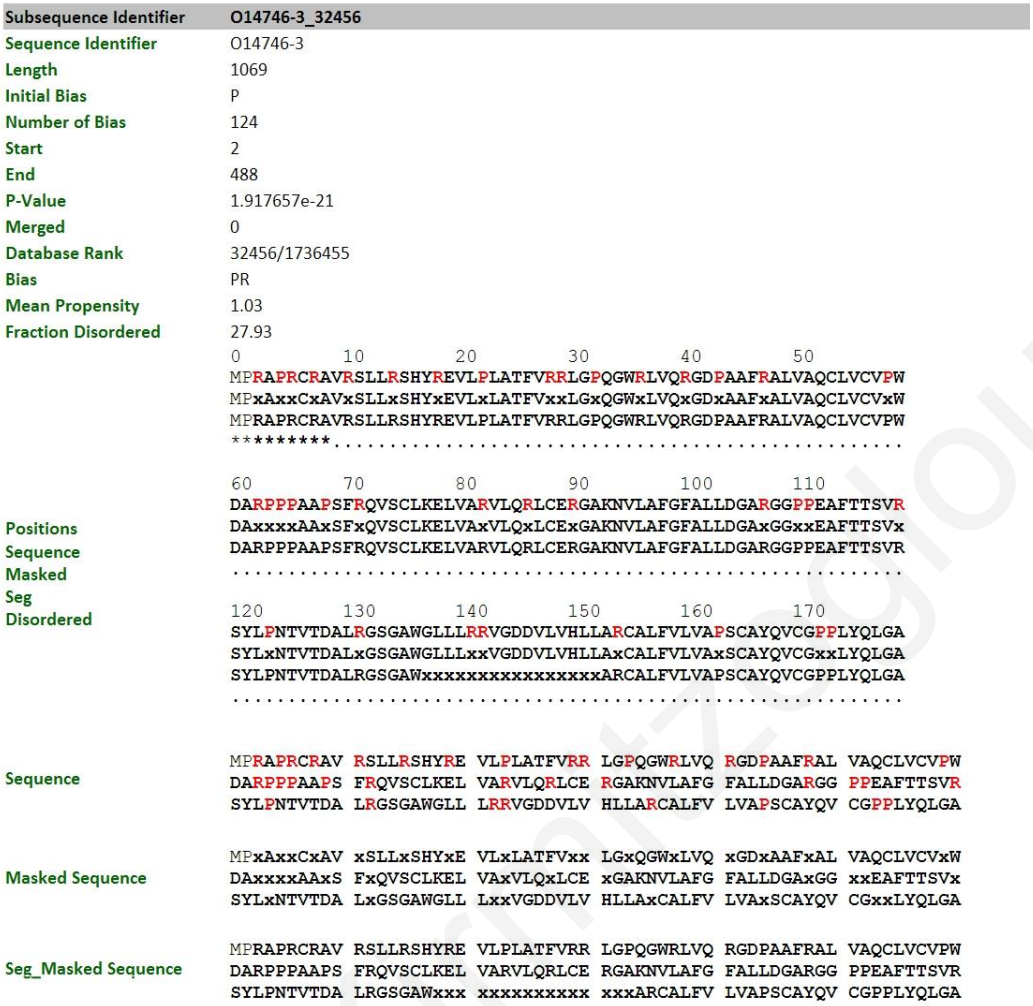


Figure 1-2. *The "protein details" view of LPS-annotate.*

The view is a minimally annotated representation of the sequence, containing minimal information apart from the location of the biased regions and two masked versions of the sequence (with LPS and SEG) [sequence trimmed to 180 AAs for space constrains].

1.7 Compositional Properties of Protein Sequences and Biological Relevance

Not all questions related with the composition of protein sequences and compositional bias are about database searches. Their abundance in eukaryotic and prokaryotic genomes suggests that, while they are “uncommon” according to the random independence model, they are in fact very common in “real-life” sequences. This in turn implies that the biological relevance of compositionally biased regions may be much stronger than originally thought. The first types of LCR sequences to be studied (not surprisingly the most trivial to detect) were homopolymeric runs of single residue types (Karlin and Altschul, 1990). Aside from some obvious implications in protein function and structure

[e.g. elevated tendency for aggregate formation by polyQ repeats (Koo *et al.*, 1999; Michelitsch and Weissman, 2000)], such studies were often triggered by the potential to establish some relatively plausible molecular mechanisms (such as DNA polymerase slippage) for LCR biogenesis and evolution. Nevertheless, different works evaluate the biological relevance of LCRs from different perspectives: LCR evolution, structure and function or their association to (human) disease. Likewise the different algorithmic formulations presented in the preceding sections for detection of LCRs, different approaches have been used for the assessment of LCR biological significance; thus those results (although often coherent or at least complementary) are oftentimes non-comparable.

It is commonplace that protein structure, function and evolution are tightly interwound and there is no reason to believe that something different holds for LCR containing proteins. Wootton (1994b) was the first to compile a list of important *"interactions, functions and phenotypes"* attributed to LCRs; among them DNA or RNA binding, tumour genesis and suppression, and interactions during transcription. More recently, Romero and colleagues (Romero *et al.*, 2001) demonstrated a connection between LCRs and the lack of regular structure; their results illustrated that about 85% of UniProt/SwissProt chains found to contain LCRs using SEG, were also predicted to contain at least one long disordered region. The same authors also reported that some low complexity sequences predicted to be disordered might indeed form ordered structure when participating in the formation of a complex. This also explains why Wootton (1994a) was able to determine non-globular structured domains by their low complexity; many of these form higher-order structures such as two-stranded and three-stranded coiled-coils (Cohen and Parry, 1990). Additionally, several other authors have demonstrated the biological relevance of such regions but in a sparse fashion (Anurag *et al.*, 2011; Coletta *et al.*, 2010; Daily *et al.*, 2005; e.g. Dunker *et al.*, 2002; Kriško *et al.*, 2010; Schlessinger *et al.*, 2011; Uversky *et al.*, 2005).

Apart from local composition, global composition has also been shown to be associated or even predict biological function. Echols *et al.* (2002) demonstrated that there is a clear difference in the composition of genes and pseudogenes both at the DNA and protein sequence level. Genome-wide DNA composition was found to be species-depended (Grantham *et al.*, 1980), while composition-based signatures remained species-specific even for 50-kb segments of the genome (Campbell *et al.*, 1999; Karlin *et al.*, 1997). Kreil & Ou-

zounis (2001) were able to identify thermophilic bacterial species based on global amino acid composition alone. More recently, *Escherichia* strains were successfully clustered based on their pathogenicity status (pathogens versus non-pathogen) using simple signatures from global or local amino acid composition (Promponas, 2009).

1.8 Scope of this study

All the above clearly demonstrate that global and local composition of protein sequences are features of high biological interest, yet understudied. While several published studies showcase the abundance and importance of such regions on the molecular/structural (Dunker and Obradovic, 2001; e.g. Dunker *et al.*, 2001, 2002; Lovell, 2003; Radivojac *et al.*, 2006), functional (Echols *et al.*, 2002; Fogel *et al.*, 2005; e.g. Mitas, 1997), organismic (e.g. Pizzi and Frontali, 2001; Stern *et al.*, 2001; Müller *et al.*, 2002; Benita *et al.*, 2006; Romov *et al.*, 2006; Miskinyte *et al.*, 2013) and habitat level (e.g. Nandi *et al.*, 2003b; Tress *et al.*, 2006), the biological role of LCRs remains largely unknown and more effort is needed to elucidate their biological roles. At the same time a large number of studies during the last 30 years dealt with the identification and elimination of LCRs in order to increase accuracy of database searches.

The incorporation of composition based statistics as the default LCR-handling option in BLASTP has definitely led to a change of direction in the field; since sequence masking has gone out of fashion some researchers have shifted their efforts in creating tools aimed in the discovery of LCRs (Harbi *et al.*, 2011; Lobanov *et al.*, 2014). Yet, masking is still a viable option in many cases such as **(i)** for algorithms like FASTA or SSEARCH (Pearson, 1990) that don't offer a built-in way to handle LCRs, **(ii)** services not relying to BLASTP for sequence alignment –for example OrthoDB (Waterhouse *et al.*, 2012) and **(iii)** services choosing to employ masking instead of compositionally based statistics as is the case of OrthoMCL (Li *et al.*, 2003). Furthermore, no detailed studies were carried out that proved the superiority of CBS over other masking algorithms besides SEG. Another unexplored possibility is that database masking may indeed be a viable option; however, apart from anecdotal references (Altschul *et al.*, 2005) this was never fully investigated. At the same time, there is only a small number of tools or services capable of helping biologists to study LCRs in depth. Most of the methods capable of detecting LCRs were created for the sole purpose of masking them and are meant to be used from the command line as part of a sequence analysis or search pipeline.

All the above have a negative impact in the study of LCRs. In this work, we aimed to document best practices in using sequence masking and at the same time develop novel tools (and refine existing ones) to assist better study of LCRs. Our strategy was three-fold:

I. investigate and document the best way to deal with LCRs in protein database searches (and sequence alignment in general), towards improving the accuracy of sequence search tools,
II. build novel tools that allow researchers to search, visualize and share LCRs in protein sequences in ways not possible before, and
III. use these tools to demonstrate how LCR detection methods may prove to be useful in settings other than just assisting database search.

To achieve the aforementioned objectives we (I) performed a series of exhaustive benchmarking experiments with different database search strategies, (II) developed an enhanced version of the CAST LCR-detection tool (CAST2) with significant performance improvements and new features, as well as the LCR-eXXXplorer[20] web-based environment for handling LCR data, and (III) inspired by a recent study (Promponas, 2009), we demonstrate an effective and fast way to predict the pathogenicity of *Escherichia* strains, using simple signatures of global and local amino acid composition.

[20] http://repeat.biol.ucy.ac.cy/mgb2/gbrowse

2 Data & Methods

2.1 Effects of LCRs and LCR-handling tools in protein database search

2.1.1 Data collection

For the all-against-all self-comparisons we used the protein sets from four complete genomes (*Escherichia coli*-K12, *Salmonella enterica typhimurium*-LT2, *Methanocaldococcus jannaschii* and *Plasmodium falciparum*). Three of those genomes were selected because they were known to vary greatly in compositional features and sizes (*E. coli*, *M. jannaschii* and *P. falciparum*) and cover the three main domains of life, while *S. enterica* was chosen as a close relative of *E. coli*, thus serving as a reference protein set. The four full proteomes were obtained from the GenBank FTP site[21] in FASTA format.

The data source for all the ASTRAL-based benchmarks was the ASTRAL[22] compendium. More specifically, we downloaded all the genetic domain sequence subsets which, based on PDB SEQRES records, had less than 40% and 20% identity to each other (referred as $ASTRAL_{40}$ and $ASTRAL_{20}$ respectively). We also generated $ASTRALNS_{40}$ and $ASTRALNS_{40}$ which are subsets of $ASTRAL_{40}$ and $ASTRAL_{20}$ containing only entries belonging to SCOP families with two or more sequences. The version of ASTRAL used was 1.73.

To compare the LCR distribution of ASTRAL versus UniProt/SwissProt in two different time points we used versions of each database released roughly at the same time; SwissProt 40 (10/2001) and $ASTRAL_{40}$ 1.57 (02/2002), SwissProt 52.6 (09/2008) and $ASTRAL_{40}$ 1.73 (09/2008). To showcase the evolution through time of LCR content in protein and structure databases we split the datasets into subsets covering a period of 12 months, starting from 1999 to 2008. Data was split based on the date of first appearance of each sequence in the datasets. The version of UniProt/TrEMBL used was 14.1 (released on 09/2008).

[21] ftp://ftp.ncbi.nlm.nih.gov/genbank/genomes/
[22] http://astral.berkeley.edu/

2.1.2 LCR masking

CAST (Promponas *et al.*, 2000) and SEG (Wootton and Federhen, 1993) were applied with default parameter settings to all the original FASTA files. Filtered sequences were saved in separate FASTA files. Where appropriate (see "Running modes" below), those masked files were used as input query/database files for BLASTP which had its internal filtering capabilities disabled. When running CAST we always enabled the parameter –stat (or –tab in versions 2.x), which resulted in generation of additional files containing "statistics" about the detected low complexity regions (such as the amino acid type of the LCR, location in the sequence and its score according to CAST).

2.1.3 Running modes

All of our evaluations were performed using NCBI BLAST 2.2.18 (Altschul *et al.*, 1990) under different LCR handling schemes utilizing SEG, CAST and the various modes of composition based statistics (CBS) offered by BLASTP (Figure 2-1). Using the masking software independently we masked the query or database sequences (or both). For SEG these modes are referred as SM1, SM2 & SM3 respectively· CAST modes use the CM abbreviation. When no masking was applied we use NM (see also Table 2). Compositional-based score adjustments are abbreviated as CB1, CB2, CB3 corresponding to the argument provided to BLASTP using the –C switch. We also evaluated the unified p-value calculation that it can be used in conjunction with the various CBS modes of BLASTP by appending a U character to the –C arguments, although this setting was marked as experimental by BLAST developers and has been removed from newer versions of the software.

We evaluated the performance of the various combinations of the aforementioned modes (including CBS) resulting in a total of 49 evaluation scenarios [e.g. both query and database masked using SEG in combination with CBS, as in Yu & Altschul (2005) abbreviated as SM2CB2].

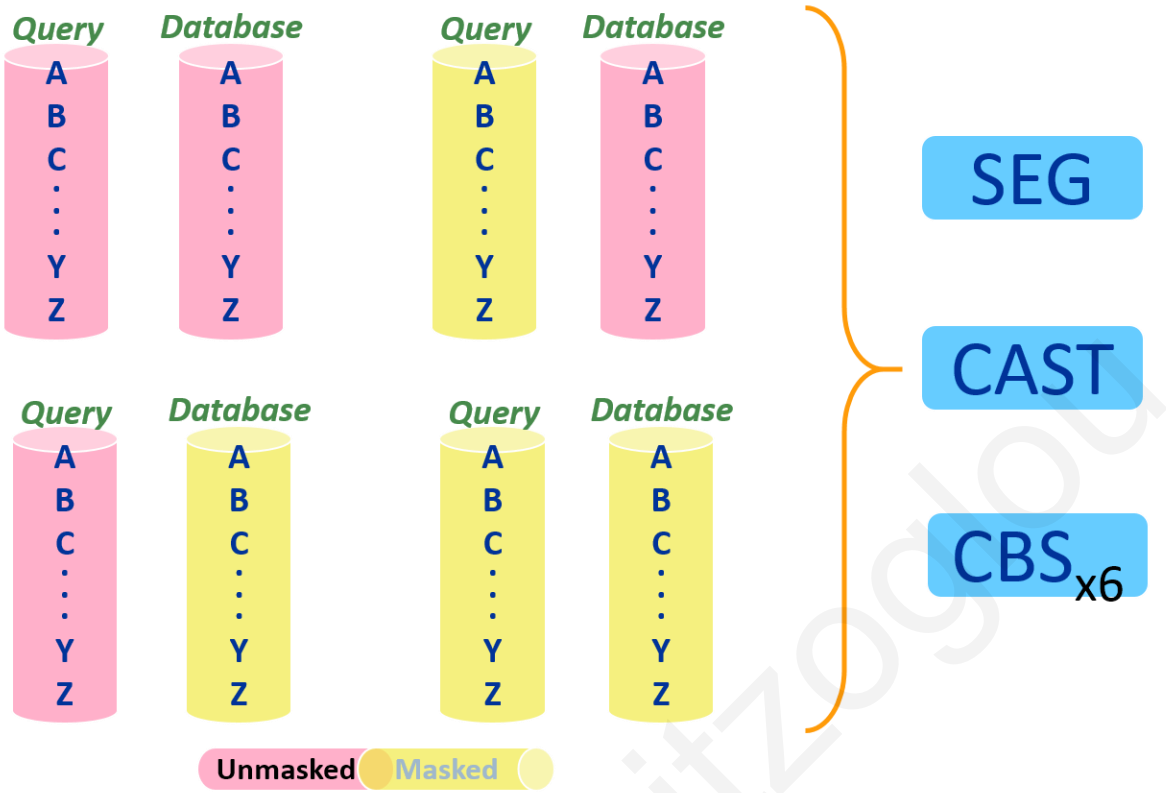


Figure 2-1. *Running modes evaluated in this study.*

In total we evaluated 49 modes for the full genomes and ASTRAL based data-sets.

After evaluating the results for all 49 modes 9 of them were chosen for presentation in this study, depending on their performance and spread of use (Table 2).

Table 2. *Running modes chosen for presentation in this study.*

These modes where either top-performing or widely used in BLASTP searches.

Abbreviation	Description
NM	This is the bare BLASTP mode (i.e. unfiltered query versus unfiltered database) and was used as the reference for our comparisons.
NMCB2	Same as NM with the employment of -C 2 option. This is the default LCR handling mode of recent BLAST versions (including the NCBI-BLAST web server).
SM1	Query sequences were masked by SEG. This used to be the default LCR handling mode of BLASTP until recently and is still widely used in routine database search and benchmarking studies.
SM2-3	Database (SM2) or both query and database (SM3) sequences were masked by SEG. These modes were selected to evaluate possible benefits of database masking
CM1-3	Query (CM1), database (CM2) or both (CM3) were masked by CAST. These modes were included because of their notable performance and for direct comparison with the respective SEG modes
SM2CB2	Database masked with SEG and employment of -C 2 option. This mode was used in the review by Altschul *et al.* (2005) and was included for comparison purposes

2.1.4 Sequence comparisons

All BLASTP comparisons were performed on a PC with an Intel CoreDuo dual core 2.4GHz CPU and 4 GB of RAM using BLASTP (version 2.2.18 x64) in multiple CPUs mode (option –a 2), under Linux.

The internal query masking offered by BLASTP was always disabled (option –F F). For the creation of all output files we employed the –m 8 switch (tabular format), with all other BLASTP parameters having their default values except for the –C option (CBS) which was set accordingly to the mode under investigation. E-value threshold for full genomes runs was set to 0.001. ASTRAL-based comparisons had an E-value threshold of 100, a value which is not biologically relevant but which has been used in the literature (Altschul *et al.*, 2005; Yu *et al.*, 2006) in order to compute ROCn for relatively large values of n (>5000),

especially in small datasets like ASTRAL. We also performed ASTRAL-based benchmarks with an E-value threshold of 0.001 to assess the importance of this settings in the resulting execution times and output file sizes. For generating the BLASTP databases `formatdb` was run with default parameters using as source the original or masked FASTA files.

2.1.5 Evaluation of full genome self-comparisons

Using full-genome self-comparisons has the advantage of knowing the correct hit beforehand (i.e. the query sequence itself!), and we could reliably obtain best-case performance estimates for the different masking schemes employed in our study (Figure 2-2).

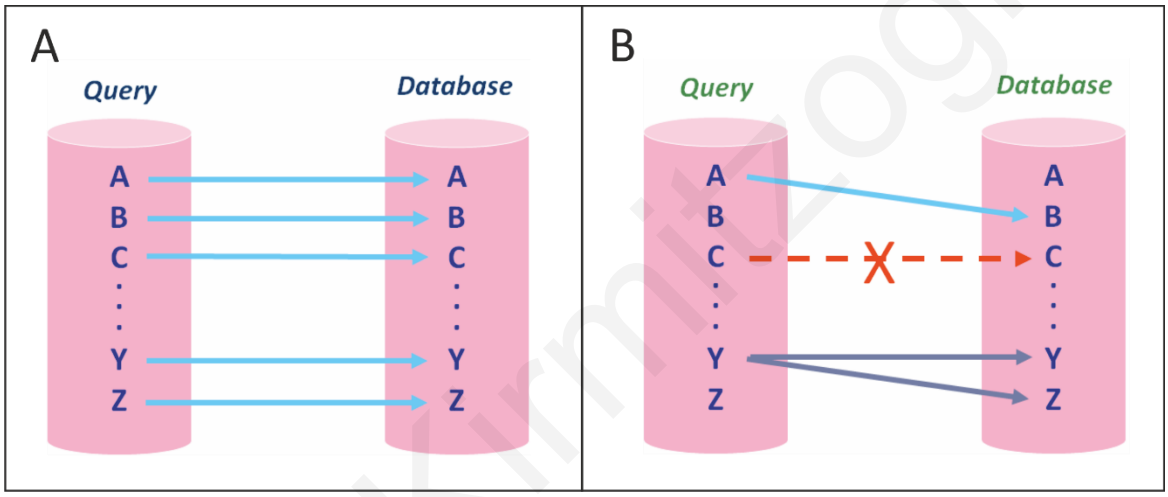


Figure 2-2. *Self-comparison best hits: ideal case versus real-life.*

A. *In a self-comparison we ideally expect each protein sequence to return itself as the best hit.*

B. *Because of various reasons, including BLAST's heuristic nature and the presence (and handling) of LCRs, a protein sequence may actually return* ***(i)*** *another protein as the best hit (blue),* ***(ii)*** *no hits at all (red) or* ***(iii)*** *more than one sequences with the same $E-value$ (and possibly bit score; grey) even though the hit sequences may not be identical.*

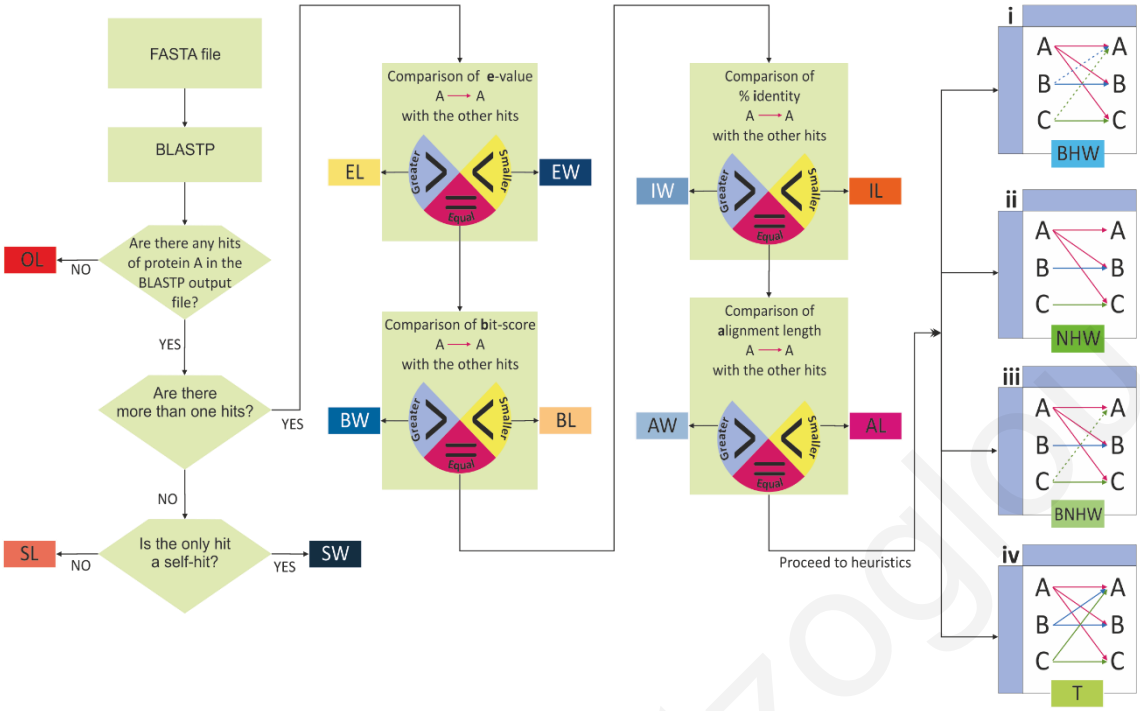


Figure 2-3. *Flowchart of the method used to evaluate the performance of the different masking modes.*

See text for details.

To resolve cases where multiple top scoring hits share the same E-value we utilize a series of features contained in the BLASTP output files (E-value, bit score, percent identity matches and alignment length) in combination with novel heuristics. The flowchart of the proposed method is presented in Figure 2-3.

This flowchart illustrates how the files produced by BLASTP for a query *protein A* are examined (small rectangles correspond to the various outputs of our method). In the first pass the algorithm examines whether it can match *protein A* with itself (considering that this is the expected result in an all-against-all self-comparison) by sequentially utilizing features contained in the output files, namely: E-value, bit score, percent identity and alignment length. A second pass is employed when the aforementioned features give multiple top-scoring hits, and heuristics are used to evaluate the following scenarios, regarding all protein sequences significantly hit by protein A. More specifically, we inspect the hits those protein sequences give:

i. They all give themselves as the best hit along with a significant hit with A.

ii. They all do not give a significant hit with A.

iii. For all of them either (i) or (ii) are true.

iv. Either (i) or (ii) are false for at least one of them.

Based on what the output of the algorithm is, the protein sequence is classified in one of the categories described in Table 3. The final method was implemented as a pair of Perl scripts and was optimized to parse and analyse BLASTP output files that were very big (i.e. larger than 4GBs) (Code Snippet 6-8, page 179 & Code Snippet 6-9, page 185).

2.1.6 Evaluation of ASTRAL-based benchmarks

We performed BLASTP searches based on $ASTRAL_{40}$ and $ASTRAL_{20}$. Only sequences belonging to SCOP families with two or more members in each set ($ASTRALNS_{40}$ and $ASTRALNS_{20}$ respectively) were used as queries. According to standard practice, for each query sequence we considered as false positives (FP) all those hits belonging to a different SCOP superfamily, while hits belonging to the same family were treated as true positives (TP). Hits belonging to the same superfamily but to a different family were ignored since, for such cases, a clear homology relationship cannot be inferred.

All TP and FP hits produced for each query sequence were merged in a single set and sorted by ascending E-value. Apparently, a database sequence may be encountered multiple times, if hit by more than one query sequences. In the case of multiple HSPs for a given query-hit pair, only the best HSP (ranked by E-value and, in case of E-value ties, by bit score) was taken into consideration. Self-hits were also excluded. In the case of tied hits (hits with the same E-value and bit score), we assign to tied observations the same midrank, which is the average of the ranks of these observations.

Table 3. *Classification of protein sequences based on the output of the BLASTP comparisons. Class colours match the ones used in Figures 2-3 and 3-7.*

Class	Description
OL	Sequence didn't produce any hits with an E-value lower than the threshold.
SW	Only one hit was found in the m8 file for this sequence and that was with itself.
EW	Multiple hits were for this sequence but the one with itself had the smaller E-value.
BW	Multiple hits for this sequence had the same E-value but the one with itself had the highest bit score.
IW	Multiple hits for this sequence had the same E-value and bit score, but the one with itself had the highest Identity Percentage.
AW	Multiple hits for this sequence had the same E-value, bit score and identity percentage, but the one with itself had the longest Alignment length.
BHW	Sequence had multiple hits with all features (E-value, bit score, identity percentage, alignment length) equal but all other sequences reported as hits gave themselves as best hit.
NHW	Sequence had multiple hits with all features (E-value, bit score, identity percentage, alignment length) equal but all other sequences reported as hits didn't produced any hits with this sequence.
BNHW	Sequence had multiple hits with all features (E-value, bit score, identity percentage, alignment length) equal but all other sequences reported as hits either gave themselves as best hit or didn't produced any hits with this sequence.
SL	Only one hit was found in the m8 file for this sequence and that wasn't with itself.
EL	Multiple top-hits were found for this sequence but the self-hit didn't have the smallest E-value.
BL	Multiple top-hits (including the self-hit) for this sequence had the same E-value but the self-hit didn't have the highest bit score.
IL	Multiple top-hits (including the self-hit) for this sequence had the same E-value and bit score, but the self-hit didn't have the highest Identity Percentage.
AL	Multiple top-hits (including the self-hit) for this sequence had the same E-value, bit score and identity percentage, but the one with itself didn't had the longest Alignment length.
T	All features checked (E-value, bit score, identity percentage, alignment length plus the heuristics described for BHW, NHW, BNHW) couldn't separate the self-hit from the rest of the hits. Therefore, this case refers to an unresolvable tie.

We then proceeded in the generation of Receiver Operating Characteristics (ROC) plots, which are graphical plots of the sensitivity (TP fraction) versus the 1-specificity (FP fraction) for a binary classifier with a continuously varying decision threshold. More simply, in sequence matching evaluation a ROC plot is created by classifying the produced hits (positives) as either true or false and sorting them using a predefined diagnostic measure as described in the previous paragraph. Each of the sorted hits defines a point on a 2-D plot with the x-axis (y-axis) indicating the fraction of FP (TP) hits with equal or higher e-values. The area under the ROC curve measures the probability of correct classification (Gribskov and Robinson, 1996) and is equal to the value of a Wilcoxon non-parametric test (Bamber, 1975). The area under the ROC curve until n FP hits are found may also be plotted to create truncated ROC (ROC*n*) plots. The x-axis of a truncated ROC plot has the count of FPs encountered in ascending E-value order with their corresponding ROC*n* value on the y-axis.

Data for the creation of "Errors Per Query versus E-value" graph was generated using the method described by Green & Brenner (2002) as was implement in the `EPQvScore.pl` v1.10 perl script[23].

Execution of all experiments and creation of the graphics (including ROC*n* and Errors Per Query plots) was automated with the use of custom Perl scripts (e.g. Code Snippet 6-10, page 189) and the gnuplot[24] utility.

[23] http://compbio.berkeley.edu/people/ed/ProcIEEE/index.html
[24] http://www.gnuplot.info

2.2 LCR-eXXXplorer, a service to search, visualize and share low complexity regions in protein sequences.

2.2.1 Data Collection

Protein sequences stored in LCR-eXXXplorer were originally downloaded from UniProt/SwissProt (The UniProt Consortium, 2014). The current version of LCR-eXXXplorer is based on UniProt/SwissProt release 2014_05[25], which contains 545,000 manually annotated protein sequences. LCR-eXXXplorer will be updated regularly; a mechanism to automatically update LCR-eXXXplorer whenever a new version of UniProt/SwissProt is released is being planned for the near future.

2.2.2 System Description & Architecture

Development of LCR-eXXXplorer was based on the following principles:

1. Maximum compatibility with all major internet browsers (Google Chrome, Mozilla Firefox, Internet Explorer, Safari) and desktop operating systems (Windows, OSX, Linux).
2. No need for third-party frameworks or plugins such as Java or Adobe Flash.
3. The web service is meant to be used on desktop computers or laptops but should also retain most of its capabilities when used from a mobile device.
4. The system will run exclusively on open source or free software.
5. Adaptation of existing software will be prioritized over the creation of solutions from scratch.

Based on these principles the system was built on top of the Generic Genome Browser (GBrowse), a *"web-based application for displaying genomic annotation and other features"* (Stein *et al.*, 2002). As GBrowse was primarily designed to work with genomic data, we had to adapt it to properly work with protein sequence data. All adaptations, either to

[25] http://www.uniprot.org/news/2014/05/14/release

the core modules of GBrowse or with the creation of novel plugins, were written in Perl[26] with the extensive use of BioPerl (Stajich *et al.*, 2002), both of which form the core of GBrowse.

The whole system runs on a server with an Intel® Core™ 2 Duo CPU clocked at 3.0GHz and 8 gigabytes of RAM, under the Ubuntu Server 10.04 LTS[27] operating system. Web requests are served by Apache (Fielding and Kaiser, 1997) version 2.2.14. Protein sequences and annotation data are stored in a MySQL database[28], while user information and uploaded data are stored using a SQLite database[29]. BLAST (Altschul *et al.*, 1990) searches on LCR-eXXXplorer are performed using NCBI blast+ version 2.2.29, while the front-end relies on a modified version of SequenceServer (Priyam *et al.*, n.d. in preparation). Protein sequence queries are masked using the segmasker version 1.0 implementation of SEG (Wootton and Federhen, 1993) and version 2.0 of CAST (Promponas *et al.*, 2000; Kirmitzoglou et al., in preparation) using the default parameter values.

2.2.3 Database of protein sequences and annotations

Protein sequences and LCR-related annotations are all stored in a MySQL database. The system is flexible enough to allow several databases to co-exist and served concurrently by LCR-eXXXplorer. Details for each database (such as name, description, storage paths, end-user visibility) are configured with the use of simple GBrowse configuration files (config files), as described in the GBrowse manual[30].

Database schema

Each database is based on the SeqFeature schema of GBrowse and consists of nine primary tables where protein sequences and their annotations are stored (Figure 2-4). Tables "*features*" and "*names*" store the protein sequences and their names respectively. General protein information (e.g. name, organism, gene names) and LCRs are stored inside

[26] http://www.perl.org
[27] http://www.ubuntu.org
[28] http://www.mysql.com
[29] http://www.sqlite.org
[30] http://gmod.org/wiki/GBrowse_2.0_HOWTO

table *"attribute"* in normalized form. A list of all annotation types is stored in table *"attributelist"*.

Since all annotations are stored in normalized form, searching for protein sequences having specific attributes is both complex and slow. To alleviate this drawback, we expanded the SeqFeature schema to include an additional table (*"attributespivot"*) were all searchable annotations are de-normalized and stored. This pivot table is recreated every time a new UniProt/SwissProt release is loaded into LCR-eXXXplorer with a series of CREATE VIEW SQL statements (Code Snippet 6-1, page 137). Due to the size of the database, the final view is converted to a normal table to boost search speed but in the expense of storage and memory requirements.

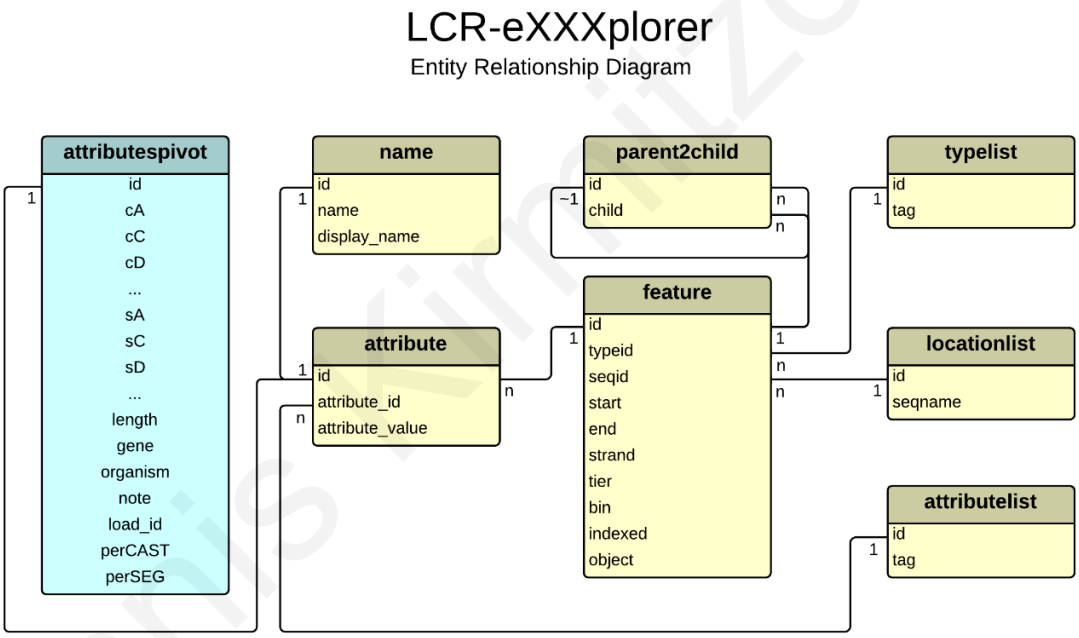


Figure 2-4. *The database schema of LCR-eXXXplorer.*

Items in yellow belong to the original SeqFeature schema of GBrowse. The table ***attributespivot*** *is a pivot table generated by de-normalizing the* ***attribute*** *table. While this adds to the database size it allows fast performance and detailed searches using the "Advanced search plugin". Fields named cX and sX correspond to the count of masked residues of type X by CAST and SEG respectively. Due to space constrains we don't display all the cX and sX fields in this figure.*

GFF3 files: creation and loading into the database

To store the sequences and LCR annotations into the database the sequences are first processed by SEG and CAST and then converted into GFF3 format using a custom Perl

script (Code Snippet 6-2, page 143). This script converts locations of residues masked by the two algorithms and low complexity regions detected by CAST (which are saved into a separate text file) into GFF3 compliant features. Furthermore, attributes such as the number of masked amino acids residues per type are added to the primary protein features, allowing for the generation of composition graphs for each sequence on-the-fly (Figure 3-17). The resulting GFF file is loaded into the database using the bp_seqfeature_load.pl Perl script that is part of the BioPerl distribution.

2.2.4 The web service

LCR-eXXXplorer is built upon a big collection of technologies but the part interaction with the end-user relies heavily on GBrowse –for searching and displaying protein sequences and annotations– and on SequenceServer for BLASTP searches.

Gbrowse

Installation & Configuration

GBrowse version 2.43 and all prerequisite software were installed following the instructions described in the GBrowse installation manual[31]. Furthermore, in order to boost performance, mod_Perl[32] was installed and Apache was configured to use it as the default Perl interpreter. Configuration of GBrowse was done following the instructions in the GBrowse manual.

Modifications

While GBrowse was designed to work on top of genomic data it retains almost all of its functionality when working with protein sequences. Each protein sequence is treated like the equivalent of a chromosome and LCR-related annotations are the equivalent of chromosomal features, such as genes, miRNAs etc. One issue we encountered was the proper display of the protein sequence in the tracks browser using the built-in glyphs (i.e., Perl classes that set the shape and behaviour of each feature rendered in the tracks browser). For example, using the *"dna"* glyph resulted in the display of an erroneous GC content graph in low magnifications, while on the other hand using the built-in *"protein"* glyph

[31] http://gmod.org/wiki/GBrowse_2.0_Install_HOWTO
[32] http://perl.apache.org/

worked well in low magnifications but forced GBrowse to erroneously "translate" the already translated protein sequence when zoomed-in. To overcome such issues we created a novel glyph type that was based on the *"protein"* one which correctly treats the underlying data as protein sequences.

Another issue encountered was displaying the search results using the built-in GBrowse search engine. Since GBrowse was designed for genomic data it normally displays an overview of the results in a karyotype view, overlaying the location of the retrieved features on top of the genomes's chromosomes. LCR-eXXXplorer database can be seen as the equivalent of a genome with over half a million of really small chromosomes, which made the karyotype view impossible to navigate for a variety of queries and also a computational bottleneck: depending on the search term, hundreds of "chromosomes" might need to be drawn and displayed. The redesigned results page (Figure 2-5) is a lightweight table view with some added information, including source organism name, which was unnecessary in the original view where all the results belonged to the genome of the same species.

The "SubmitBLAST" plugin

GBrowse offers a powerfull extensibily framework by providing developers the option to create novel pieces of functionality that can be added to any GBrowse installation in the form of plugins.

LCR-eXXXplorer facilitates innitiating BLASTP searches on external servers (such as NCBI[33]) using as query sequence any sequence stored within the LCR-eXXXplorer database; masking is offered using any combination of annotations stored in the database. To accomplish that we developed a novel *"dumper"* plugin for GBrowse called "SubmitBlast" (Code Snippet 6-3, page 154). The plugin is written in Perl and can be easilly expanded to support other web-blast servers in the future.

[33] http://blast.ncbi.nlm.nih.gov/Blast.cgi

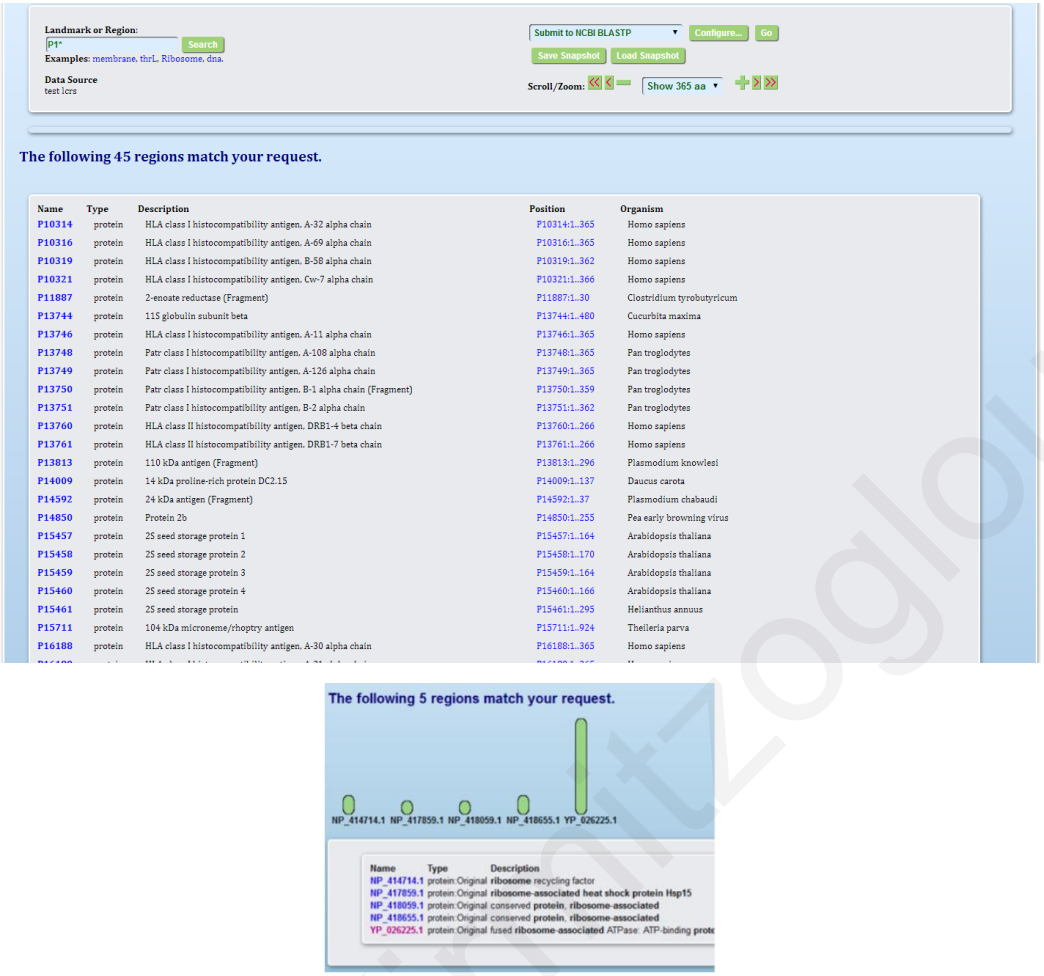


Figure 2-5. *Search results in LCR-eXXXplorer.*

The redesigned search results page of LCR-eXXXplorer (top) versus the original karyotype view (bottom). The chromosomes were removed in favour of a simple table with more fields compared to the original.

The "AdvancedSearch" plugin

Due to the normalized nature of GBrowse's database schema and the fact that it is focused on feature visualization, as opposed to discovery, searching for features with any term other than those available in names or descriptions was not possible. Furthermore, combining more than one search terms was not proPerly supported. To overcome that we could either store all LCR-related annotations as alternative names (called Aliases in GBrowse terminology) or expand the feature schema and built a plugin specifically for advanced searches. The former solution would create more issues than it would solve, thus we chose to implement the latter.

Alterations to the SeqFeature schema have already been described in section *"Database schema"*. The *"Advanced Search"* form is a GBrowse *"finder"* plugin (Code Snippet 6-4,

page 159). The plugin is written in Perl but instead of using standard GBrowse Perl objects we opt to perform the search by constructing SQL SELECT statements based on the search criteria entered by the user. This allowed us to take advantage of the modified database schema and at the same time provided significant performace gains. SQL statements are built on-the-fly using the excellent SQL::Abstract[34] Perl library and executed through the DBI[35] library, which provides the additional benefit of protecting the server from SQL-injection attack attempts. Finally, the *"Advanced Search"* form performs a client-side validation of the entered search terms (e.g. length must always be a number) through the use of pure Cascading Style Sheets (CSS). At its current state the form only returns protein sequences matching all search criteria (BOOLEAN AND).

Generation of the "Record Summary" table

Record summary tables in LCR-eXXXplorer are a combination of annotations stored locally and of data fetched remotely from the UniProtKB[36] servers. Annotations on the left side of the table (such as protein and gene names, GO terms, PDB structures) are fetched remotely from UniProt/SwissProt using a cgi-bin Perl script (Code Snippet 6-5, page 165). If this fails then the script uses the annotations stored locally whenever possible, ensuring that a user always receives up-to-date annotations.

The same script is responsible for the generation of the composition graphs on the right side of the table. Precompiled composition and masking data for the whole UniProt/SwissProt database are stored as arrays within the source code. Equivalent data for each protein sequence are stored as attributes inside the LCR-eXXXplorer database. When a user requests to view details for a protein sequence relevant graphs are created on-the-fly using gnuplot[37] and are stored in PNG format inside a publicly accessible directory. Caching the PNG files ensures that CPU cycles are not wasted regenerating graphs for sequences that have been accessed in the recent past. A CRON job runs at regular intervals and deletes PNG files older than a month.

---

[34] http://search.cpan.org/dist/SQL-Abstract/lib/SQL/Abstract.pm
[35] http://search.cpan.org/~timb/DBI-1.631/DBI.pm
[36] http://www.uniprot.org/
[37] http://www.gnuplot.info

Generation of the "Selected annotations (features) from UniProtKB" track

LCR-eXXXplorer offers the option to display annotations stored remotely on the UniProtKB servers as a separate track in the tracks browser. This is possible by utilizing the ability of GBrowse to display remote data that is served in a valid GFF3 format. Although we could directly use the GFF3 files served by UniProt/SwissProt, incompatibility issues combined with the need to further customize the display of the UniProt/SwissProt annotations led us to develop a custom cgi-bin Perl script (Code Snippet 6-6, page 171).

This script sits between GBrowse and UniProt/SwissProt and acts as the "remote" data source for GBrowse, even though it resides on the same server. It accepts just one parameter, which is the UniProt Accession number of the protein requested by the user. It then contacts the UniProtKB server, downloads the data, groups annotations, generates the formatting options for the tracks and outputs everything in a GFF3 file that is served to GBrowse. The whole process is seamless for the end-user and in normal operating conditions takes less than half a second. If the UniProtKB servers are not reachable LCR-eXXXplorer displays an error message inside the track but the all the other tracks are displayed normally.

Generation of the "Predicted Regions from ANCHOR and IUPRED" track

The "Predicted Regions from ANCHOR and IUPRED" track is also generated on-the-fly every time a user is viewing a protein. The procedure is similar to the one described above although this time is does not involve fetching data from a remote server. A cgi-bin Perl script (Code Snippet 6-7, page 174) accepts the UniProt Accession number of the protein as an input and then retrieves the protein sequence and runs local instances of ANCHOR (Dosztányi *et al.*, 2009) and IUPRED (Dosztányi *et al.*, 2005a). In the next step it parses the output files generated by the two software tools and converts them into wiggle track format (WIG), which is a binary file type optimized for storing data meant to be displayed as graphs. At the final step, the script generates a GFF3 file and serves the GFF and WIG files to GBrowse, which then displays them using the appropriate glyphs.

The whole process takes less than half a second to generate all the files but it also caches the WIG files, eliminating the need to execute ANCHOR and IUPRED if the protein has already been displayed to any user in the recent past. A separate CRON job cleans the temporary directory containing the GFF3 and WIG files in regular intervals.

Local BLAST searches

To initiate BLASTP searches against the sequence data stored in the LCR-eXXXplorer system and present the results to the end-users LCR-eXXXplorer utilizes SequenceServer[38], a Ruby-based BLAST front-end optimized for speed of use and simplicity. Installation and configuration was performed following the instructions on http://www.sequenceserver.com. Instead of using the built-in web-server we opted for serving SequenceServer through Apache. Appropriately formatted BLAST databases were created using the `format-databases` subcommand of the `sequenceserver` executable. LCR-eXXXplorer offers three versions of the same database; one with unmasked sequences and two with sequences masked using SEG and CAST respectively. Sequence masking was performed using the default parameter values for both methods.

To enable cross-linking of the BLAST results with GBrowse data we applied minor modifications to the source code of the `sequenceserver` module responsible for parsing and displaying BLAST results. This enabled us to directly link to the hit sequence in GBrowse and also display the BLAST result as an extra track in the tracks browser.

[38] http://www.sequenceserver.com/

2.3 PREDICTING THE PATHOGENICITY OF *ESCHERICHIA* STRAINS BASED ON LOCAL AND GLOBAL AMINO ACID COMPOSITIONAL SIGNATURES

2.3.1 Data Collection

For our analysis we chose the *Escherichia* genus of gram-negative bacteria commonly found in the normal flora of the intestine of warm-blooded organisms as it is probably one of the most studied living creatures on earth. *E. coli* has served as a model organism in bacterial genetics (especially the commensal strain K-12) and has become the work-horse in several biotechnological applications. However, pathogenic *Escherichia* strains are long known to exist, and have been implicated, among other diseases, in urinary tract infection, sepsis/meningitis and gastro-intestinal infections (Nataro and Kaper, 1998), and very recently in Crohn's disease (Martinez-Medina *et al.*, 2009).

Importantly, it has been recently reported that *E. coli* pathogenicity does not necessarily follow phylogeny, i.e. pathogenic and non-pathogenic species mix throughout the *E. coli* phylogeny (Carlos *et al.*, 2010). Moreover, while the notion of pathogenicity islands in bacterial genomes –which are regions mainly encoding virulence genes– has been around for some time (Blum *et al.*, 1995; Lee, 1996; Morschhäuser *et al.*, 1994; Rakin *et al.*, 1995) their utility as predictors of pathogenicity has been repetitively challenged by several authors [for a recent review see (Merhej *et al.*, 2013)]. With an increasing number of *Escherichia* genomes available in the public databases we set to identify whether a holistic view of local and global compositional properties are associated with pathogenicity as recorded in the international literature.

We collected genomic data for 31 fully sequenced genomes from strains of the bacterial genus *Escherichia* from the Bacterial Genomes section of GenBank. We ignored non-coding genes as well as genes residing to plasmids. More specifically, we used all the annotated protein-coding chromosomal genes as described in the relevant `.ptt` files. Amino acid sequences were obtained from the respective `.faa` files. Of the 31 genomes, 30 belong to various *E. coli* strains and one to *E. fergusonii.*

By collecting and analysing the relevant literature we managed to verify the pathogenicity for 28 out of the 31 genomes (Table 4). Pathogenicity type (e.g. Enteroagreggative - EAEC

or Enterohemmorhagic - EHEC) for each genome was only used for presentation purposes and was not taken into account for any of the analyses in this study.

We also collected all of the *Escherichia* genomes from the Genome Project section of GenBank which were fully sequenced but only partially assembled and were provided in the form of 2 or more scaffolds (79 genomes up to March 2011). For each of the partially assembled genomes we tried to utilize the strain cards of GOLD but, unfortunately, many of these descriptions were unreliable, since they were either incomplete or inaccurate (Nikos Kyrpides, personal communication). Therefore, we decided to exclude partially assembled genomes from this study.

2.3.2 Detection of LCRs

We applied CAST v1.1 on every genome with two different score thresholds (15, 40) and the `-tab` parameter switch. Using custom Perl scripts, we parsed the original and masked genomes as well as the accompanying tab-delimited statistics file (`*.tab`) and for every protein sequence we counted the number of (i) residues, (ii) masked residues and (iii) detected LCRs, for each amino acid residue type. Counts were stored in separate files and were later utilized for the generation of compositional signatures. Storing counts for each protein (as opposed for each genome) allowed us to rapidly recalculate the compositional signatures whenever we generated new datasets by sampling genes from the original genomes.

Table 4. *Completely sequence genomes of Escherichia strains used in this study.*

All genomes were collected from the Bacterial Genomes section of GenBank. The pathogenicity of each strain (3rd column) was verified by manually searching the relevant literature (last column). We were unable to verify the pathogenicity of 3 out of the 31 strains (marked with a red U in the 3rd column). Column 4 contains pathogenicity type for each strain: APEC = Avian Pathogenic; EAEC = Enteroagreggative; EHEC = Enterohemmorhagic; ETEC = Enterotoxogenic; EPEC = Enteropathogenic; NMEC= Neonatal Meningitis; UPEC = Uropathogenic. Genomes marked with an asterisk were used as the training set for all predictive models described in this study. The rest of the verified genomes were always included in the test set used for the validation of each predictive model.

Species	GenBank Accession	P	Type	# of genes	Source
* *E. coli* str. K-12 substr. W3110	AC_000091	NP		4226	(Moriel *et al.*, 2010)
* *E. coli* str. K-12 substr. MG1655	NC_000913	NP		4144	(Moriel *et al.*, 2010)
* *E. coli* O157:H7 EDL933	NC_002655	P	EHEC	5298	(Perna *et al.*, 2001)
* *E. coli* O157:H7 str. Sakai	NC_002695	P	EHEC	5229	(Makino *et al.*, 1999)
* *E. coli* CFT073	NC_004431	P	UPEC	5338	(Moriel *et al.*, 2010)
* *E. coli* UTI89	NC_007946	P	UPEC	5021	(Moriel *et al.*, 2010)
* *E. coli* 536	NC_008253	P	UPEC	4619	(Moriel *et al.*, 2010)
* *E. coli* APEC O1	NC_008563	P	APEC	4428	(Moriel *et al.*, 2010)
* *E. coli* HS	NC_009800	NP		4378	(Rasko *et al.*, 2008)
* *E. coli* E24377A	NC_009801	P	ETEC	4749	(Rasko *et al.*, 2008)
E. coli ATCC 8739	NC_010468	U		4199	[39]
* *E. coli* str. K-12 substr. DH10B	NC_010473	NP		4126	(Moriel *et al.*, 2010)
E. coli SMS-3-5	NC_010498	U		4743	(Fricke *et al.*, 2008)
* *E. coli* O157:H7 str. EC4115	NC_011353	P	EHEC	5315	[40]
* *E. coli* SE11	NC_011415	NP		4679	(Oshima *et al.*, 2008)
* *E. coli* O127:H6 str. E2348/69	NC_011601	P	EPEC	4552	(Iguchi *et al.*, 2009)
E. fergusonii ATCC 35469	NC_011740	U		4264	[41]
* *E. coli* IAI1	NC_011741	NP		4351	(Touchon *et al.*, 2009)
* *E. coli* S88	NC_011742	P	NMEC	4692	(Touchon *et al.*, 2009)
* *E. coli* ED1a	NC_011745	NP		4915	(Touchon *et al.*, 2009)
* *E. coli* 55989	NC_011748	P	EAEC	4759	(Touchon *et al.*, 2009)
* *E. coli* IAI39	NC_011750	P	UPEC	4730	(Touchon *et al.*, 2009)
* *E. coli* UMN026	NC_011751	P	UPEC	4825	(Touchon *et al.*, 2009)
* *E. coli* BW2952	NC_012759	NP		4084	(Ferenci *et al.*, 2009)
* *E. coli* BL21(DE3)	NC_012947	NP		4228	(Chart *et al.*, 2000)
E. coli B str. REL606	NC_012967	NP		4205	(Jeong *et al.*, 2009)
E. coli O157:H7 str. TW14359	NC_013008	P	EHEC	5255	(Kulasekara *et al.*, 2009)
E. coli O103:H2 str. 12009	NC_013353	P	EHEC	5054	(Ogura *et al.*, 2009)
E. coli O26:H11 str. 11368	NC_013361	P	EHEC	5364	(Ogura *et al.*, 2009)
E. coli O111:H- str. 11128	NC_013364	P	EHEC	4972	(Ogura *et al.*, 2009)
E. coli O55:H7 str. CB9615	NC_013941	P	EHEC	5014	(Zhou *et al.*, 2010)

[39] http://www.expasy.ch/sprot/hamap/ECOLC.html
[40] http://expasy.org/sprot/hamap/ECO5E.html
[41] http://www.genoscope.cns.fr/spip/-E.-fergusonii-coli-.html

2.3.3 Generation of compositional signatures

For each genomic dataset used in this study we calculated 3 types of signatures based on the global and local composition of the sequences belonging to the dataset. Each signature is a 20-dimensional numerical vector, one element corresponding to each amino acid type. The three types of signatures generated were (see Figure 2-6 for more details):

i. Global composition (GC): this signature is based on the global amino acid composition of each dataset.
ii. LCRs composition (LB): the elements of this vector derive from the counts of LCRs for each amino acid residue type averaged over the total number of sequences in the data-set.
iii. Composition of masked residues (XB): same as LB but instead of counting LCRs we are counting masked residues per type.

Using custom Perl scripts we generated the signatures and stored them in files containing all three signatures for each data-set concatenated in one 60-element vector.

Figure 2-6. *Generation of compositional signatures.*

For each protein sequence we generate 3 types of compositional signatures (vectors), which are based on ***(GC)*** *the global amino acid composition of the sequence,* ***(LB)*** *the count of LCRs detected for each amino acid type and* ***(XB)*** *the count of masked residues per type. During the final step each vector is averaged over the whole genome/dataset.*

2.3.4 Genome clustering

Using the compositional signatures, genomes where clustered with hierarchical and k-means clustering by utilizing the corresponding `hlust` and `kmeans` functions of the R statistical package (R Core Team, 2014). Based on the observations of Promponas (2009) we decided to employ Ward's minimum variance method of hierarchical clustering. Since the genomes are expected to belong to one of the two pathogenicity categories, trees produced using Ward clustering were automatically cut in the appropriate height into two groups using the `cutree` function of R. k-Means clustering was employed for all three signatures with a value of k equal to 2. The similarity of the resulting clusters to the ideal

clustering of the genomes based on their pathogenicity was calculated using the SIM measure as described in Promponas (2009).

2.3.5 Generation of simulated incomplete genomes

To assess whether compositional signatures can be utilized to predict the pathogenicity of partially sequenced *Escherichia* strains, for each strain in our dataset we generated 100 randomized sub-sampled artificial "proteomes" with varying numbers of proteins. Along these lines, we performed random sampling without replacement thus generating artificial datasets of 9 different sizes (100, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000). Depending on the strain, these protein numbers correspond to coverage between 2% and 98% of the corresponding complete genome. The 4000 limit in particular translates to a genome coverage ranging from 75% for the uropathogenic *E. coli* CFT073 to 98% for the experimental BW2952 strain of the commensal *E. coli* K-12. In practice, this was accompliced by randomly generating 100 permutations for each of the original genomes, which were then stripped from top to match the desired sequence limit. The procedure resulted in producing a total of 900 datasets originating from each genome. Compositional signatures generation for every dataset was performed as described in section 2.3.3.

2.3.6 Predictive models: creation and validation

We arbitrarily split the 28 *Escherichia* genomes in a training set (22 genomes submitted to GenBank before July 15, 2009) and a test set (6 genomes submitted after July 15, 2009).

We used the training set to train a binary classification model utilizing the k-nearest neighbours algorithm (k-NN). We purposely chose k-NN as the classification algorithm to test the predictive performance of the compositional signatures using one of the simplest (and, in theory, least powerful) classifiers. The predictive model was implemented in R with the use of the powerful `caret` library (Kuhn, 2008; Kuhn *et al.*, 2014). The optimal value of k was selected among 5 values (1, 3, 5, 7, 9) by iteratively building 5 different models and validating their performance using leave-one-out cross-validation. At the end we kept the k producing the model with the greatest accuracy.

Apart from leave-one-out cross-validation we also measured the performance of the final model against the independent test set containing 6 genomes with verified pathogenicity that were not used during the training of the model. The measures used were:

Accuracy: $$ACC = (TP + TN)/(P + N) \quad (2\text{-}1)$$

Sensitivity: $$SNS = TP/P = TP/(TP + FN) \quad (2\text{-}2)$$

Specificity: $$SPC = TN/N = TN/(FP + TN) \quad (2\text{-}3)$$

where

TP, FP : true and false positives respectively,

TN, FN : true and false negatives,

$$P = TP + FN$$

$$N = TN + FP$$

2.3.7 Generation of chimeric genomes

In an effort to identify subsets of specific proteins responsible for the predictive power of the final model we generated chimeric genomes containing only specific classes of genes based on a pathogenicity-centered view of the *Escherichia* pangenome.

Initially, we clustered all proteins belonging to the 22 genomes of the training set based on their pairwise sequence similarity using `blastclust` with default parameters (which means LCR-handling was disabled). Then, with the help of a custom Perl script, we classified all the proteins (including those in singletons, i.e. clusters with only one sequence) in the following 4 classes based on the properties of their parent clusters:

i. Core-proteins, from clusters that contained proteins from all 22 genomes.
ii. P-only proteins, from clusters that contained proteins only from pathogenic genomes.
iii. NP-only proteins, from clusters that contained proteins only from non-pathogenic genomes
iv. Mixed-proteins, from clusters not fulfilling any of the above criteria.

Following the classification of the genes we now had a way to generate artificial chimeric genomes with custom properties (e.g. genomes containing P and NP-only sequences in equal numbers). To do that we followed 2 distinct strategies, described in the following paragraphs.

Transformation of the original 22 genomes

By keeping only the proteins matching the respective criteria we generated 3 different versions for each of the original 22 genomes having only **(i)** Core-proteins **(ii)** P or NP-only proteins (depending on the pathogenicity of the genome) and **(iii)** Mixed-proteins. The next step was to train 3 predictive models, one from each class of genomes, following the same procedure described in section 2.3.6 above. Each of the generated models was externally validated the same way we validated the model built from the complete genomes, i.e. against the test set containing the 6 remaining *complete* genomes.

Generation of artificial genomes with specific properties

We built 2 gene pools by selecting sequences, from all 22 genomes, that were either P- or NP-proteins only. We also created a third pool containing all 102,686 proteins from the 22 complete genomes. Using bootstrap resampling (i.e. resampling with replacement) from the above protein pools we generated chimeric genomes having (i) P-only, (ii) NP-only and (iii) P & NP-only genes in equal numbers. We also generated a fourth category of chimeras containing proteins randomly sampled from all genomes. Each class of chimeric genomes had 14 subclasses that differed in the number of proteins contained in each dataset (i.e. 100, 500, 1000, 2000, 2500, 3000, 3500, 4000, 5000, 6000, 7000, 8000, 9000 or 10000 genes). For each subclass we generated 1000 bootstraps bringing the number of chimeric genomes per class to 14,000.

Using the predictive k-NN model built from the 22 full genomes (see section 2.3.6) we predicted the pathogenicity of each bootstrap. Results were grouped together per chimeric class and subclass.

3 Results & Discussion

3.1 Effects of LCRs and LCR detection tools in protein database search

3.1.1 LCR content

Full proteomes

Both SEG and CAST agree in their ranking of the four arbitrarily selected proteomes based on the low complexity content (Figure 3-1). Furthermore, it is obvious that SEG is much more aggressive than CAST in reporting low complexity segments, especially in 3 out of the 4 proteomes, a consequence of the completely different rationale used by the two algorithms to identify such segments (Promponas *et al.*, 2000). In the case of *P. falciparum*, which is known to be extremely rich in LCRs (Pizzi and Frontali, 2001; Promponas *et al.*, 2000; Tan *et al.*, 2010; Tsoka *et al.*, 1999), the difference between the number of proteins with LCRs detected by the two algorithms is much smaller.

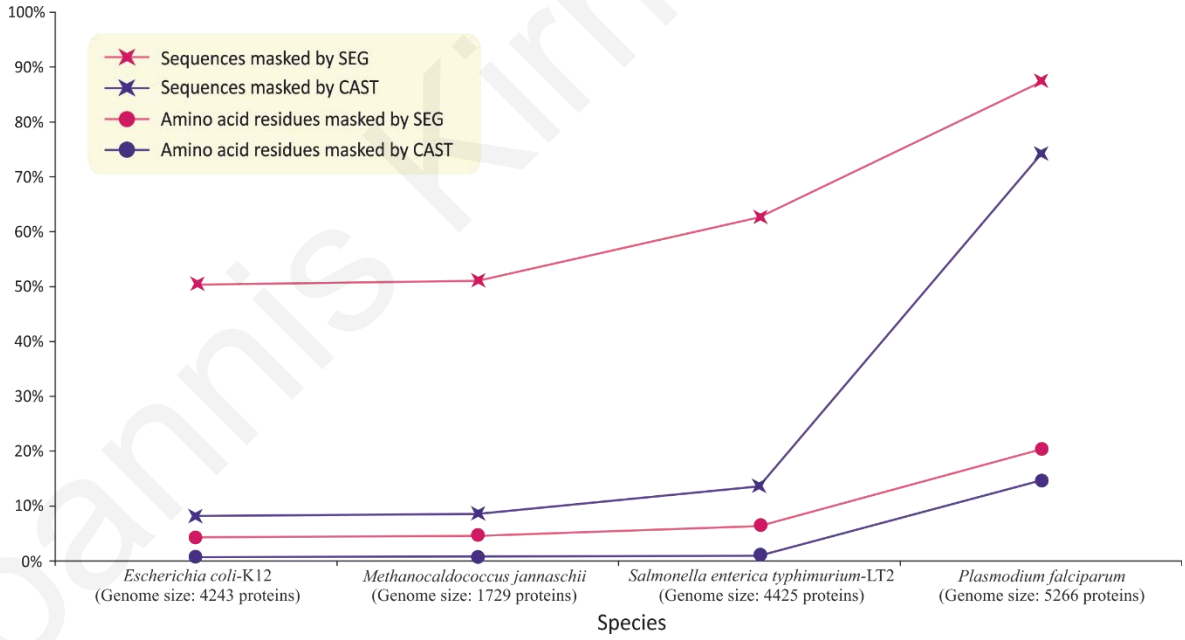


Figure 3-1. *Low complexity content of the four proteomes used in this study, according to SEG and CAST.*

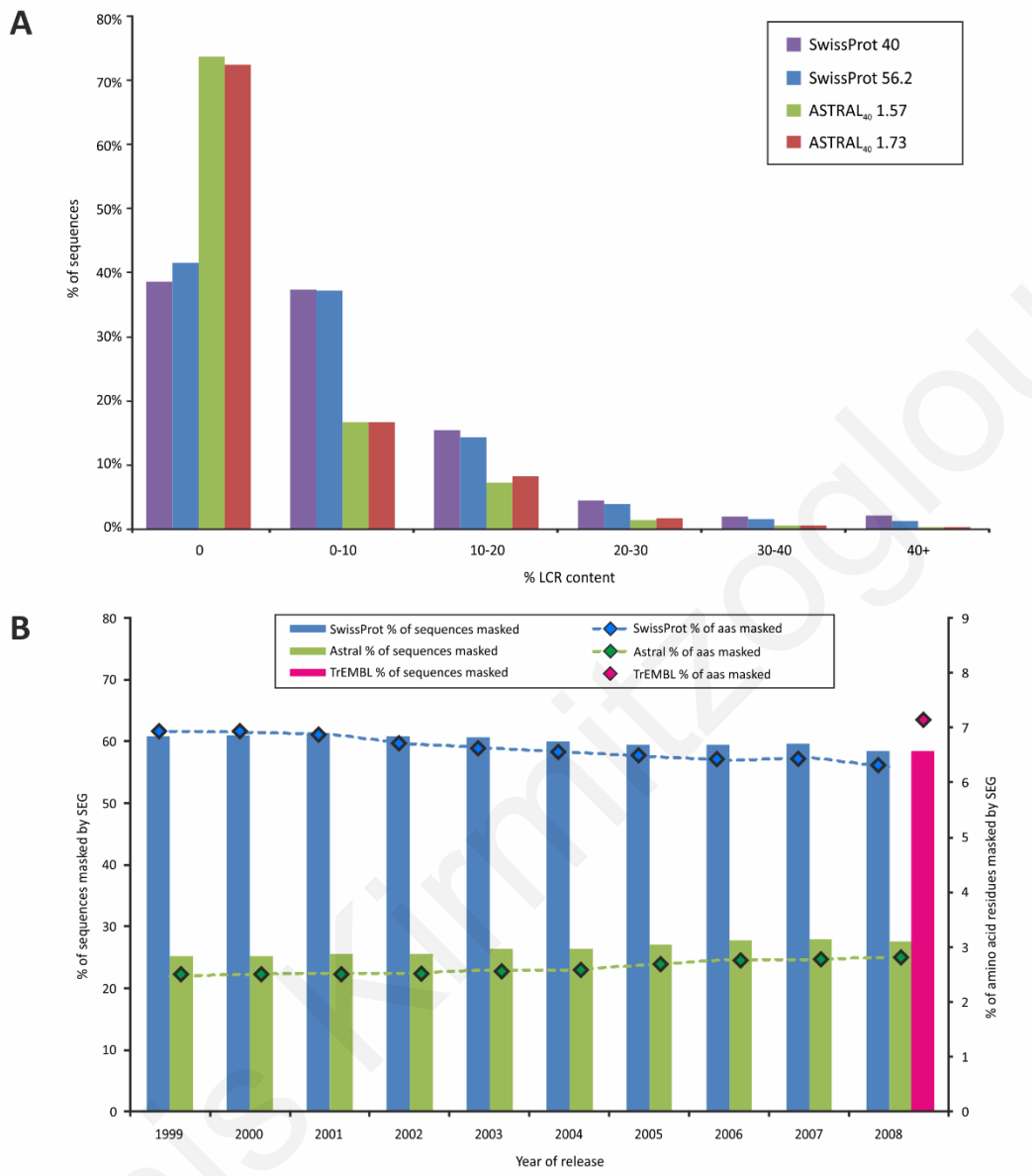


Figure 3-2. *LCRs distribution in sequence and structure databases and their evolution in time.*

A: *LCR distribution in UniProt/SwissProt and ASTRAL$_{40}$. Using SEG with the default settings we detected the LCR content of protein sequences contained in those databases and classified them according to the % of amino acids masked by SEG. We used versions of each database released roughly at the same time; UniProt/SwissProt version 40 (10/2001; purple), and ASTRAL$_{40}$ 1.57 (02/2002; green), SwissProt 52.6 (09/2008; blue) and ASTRAL$_{40}$ 1.73 (09/2008; red).*

B: *LCRs in sequence and structure databases through time. Blue bars (dotted lines) represent % of UniProt/SwissProt entries (amino acid residues) with LCRs detected by SEG. Respective figures for the ASTRAL$_{40}$ data sets in green. Magenta colour indicates respective figures for the current version of UniProt/TrEMBL (version 14.1), which is enriched in recently sequenced eukaryotic genomes. Time points roughly correspond to successive 12-month periods.*

ASTRAL$_{40}$ versus UniProt/SwissProt

Low complexity regions in proteins are often associated with lack of structure and are often absent from crystal structures or even prevent crystallization of respective protein molecules (Romero *et al.*, 2001). As Wootton (1994) pointed out, almost all protein sequences in the PDB (Berman *et al.*, 2000) are of high complexity, which is in sharp contrast with the UniProt/SwissProt sequences. In 1994 UniProt/SwissProt contained about 40,000 sequences, compared to ~2,900 PDB entries. In 2008 390,000 sequences were registered in UniProt/SwissProt, whereas ~53,000 were deposited in PDB, which has shown an 18-fold expansion in 14 years. Motivated by this fact we carried out an analysis similar to Wootton's in a retrospective manner (Figure 3-2B).

Our results indicate that despite the fact that both databases grow exponentially (with an almost double growth rate of PDB in comparison to UniProt/SwissProt; data not shown), the low complexity content of the two databases has only slightly changed. Throughout time, more than 60% of protein sequences in UniProt/SwissProt contain at least one LCR according to SEG and the percentage of residues masked was consistently higher than 6%. By contrast, the same figures in ASTRAL$_{40}$ were 25% and 3% respectively. Since the dataset used to represent PDB was ASTRAL with a redundancy limit of 40%, we expect the LCR content for the full PDB to be even lower as most of the proteins don't contain any LCRs at all.

LCR content in UniProt/TrEMBL is similar to UniProt/SwissProt; the slightly higher percentage of masked amino acids can be attributed to the much higher amount of unknown residues in UniProt/UniProt/TrEMBL (which are represented with the character X) compared to UniProt/SwissProt (3.78% versus 0.43%)[42].

Still, PDB does present a marginal increase in the low complexity content, which may be attributed to the increasing incorporation of protein structures solved by NMR techniques. In contrast, the percentage of both sequences and amino acid residues masked by

[42] Our methodology doesn't count occurrences of X that were present in the protein sequences prior to masking but SEG takes those characters into account when searching for LCRs. Since unknown residues are usually clustered in long runs, they may cause SEG to falsely detect these clusters and neighbouring regions as LCRs, which in turn increases the percentage of masked amino acids in the produced files.

SEG has slowly dropped for UniProt/SwissProt. This in turn can be explained by the fact that most of the newly annotated entries belong to high priority prokaryotic species[43], which on average contain considerably fewer LCRs than the eukaryotic ones (Promponas *et al.*, 2000). As more eukaryotic genomes are sequenced and annotated we expect to see an increase for the low complexity content of UniProt/SwissProt.

LCR distribution in UniProt/SwissProt and $ASTRAL_{40}$ (Figure 3-2A) illustrates that these databases have distinct characteristics when LCRs are concerned, especially when it comes to sequences with no or low (≤ 20%) LCR content. Amino acid sequences with no detected LCR are almost two-fold more frequent in ASTRAL compared to UniProt/SwissProt, and comprise more than 70% of this data set (even with the more sensitive SEG algorithm).

43 http://expasy.org/sprot/relnotes/relstat.html

3.1.2 Execution Times and Output File Sizes

Full proteome self-comparisons

Run times are not directly correlated to the genome size or the size of the output files produced by BLASTP (Figure 3-3). What seems to play a prominent role is the LCR content of each genome, especially when no masking is applied (NM & NMCB2). This is most probably due to the high number of spurious hits returned by BLASTP. Since composition-based statistics are applied when calculating the final alignment score, run times for these modes can be greatly reduced with masking which seems to lower the number of High Scoring Pairs (HSPs) identified by BLASTP. An exception to this trend is two-way masking using SEG (SM3 mode); we speculate that this is due to the extensive masking employed by SEG, which actually fragments some of the sequences into several smaller ones and thus increases the number of HSPs. In the case of *P. falciparum* masking both the query and the database with CAST (CM3) speed-ups execution time by a factor of almost 40x, although any mode that employs masking (especially CM1-3) greatly reduces run times.

In general, with an E-value threshold of 0.001, the same trends seem to apply for BLASTP output file sizes. For all species, with the exception of *P. falciparum*, there are no notable differences in the BLASTP output file sizes between the various running modes (Figure 3-4). In the case of *P. falciparum*, two-way masking with CAST reduces file size (and thus the number of returned results) by about 15%. This difference in even more profound when employing CAST and using a BLAST E-value threshold of 10, a value commonly selected by many researchers when using the command line version of BLASTP. Specifically, we observed a 60-fold reduction of the file size (Supplementary Figure 2). This observation is definitely related with the abundance of low complexity segments in proteins encoded by the specific genome and the elimination of most of the spurious hits otherwise produced.

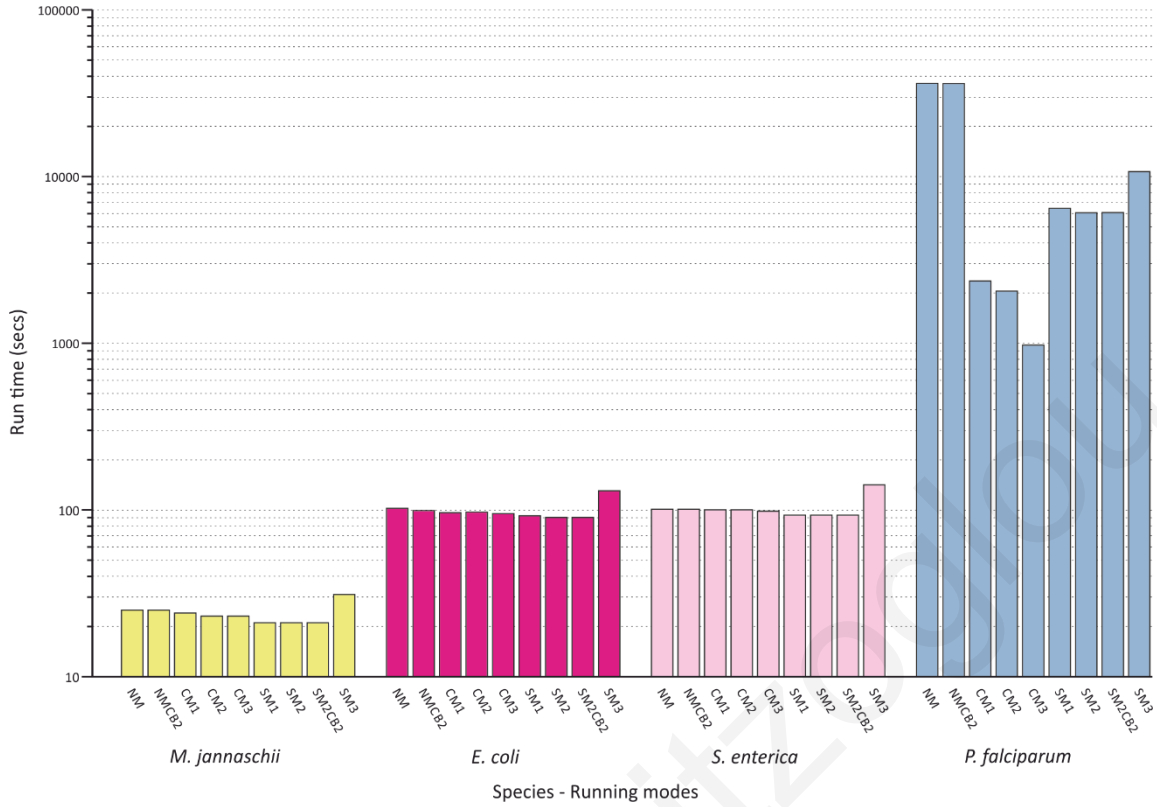


Figure 3-3. *Execution times for the completion of BLASTP all-against-all self-comparisons. For definitions of running modes refer to Table 2.*

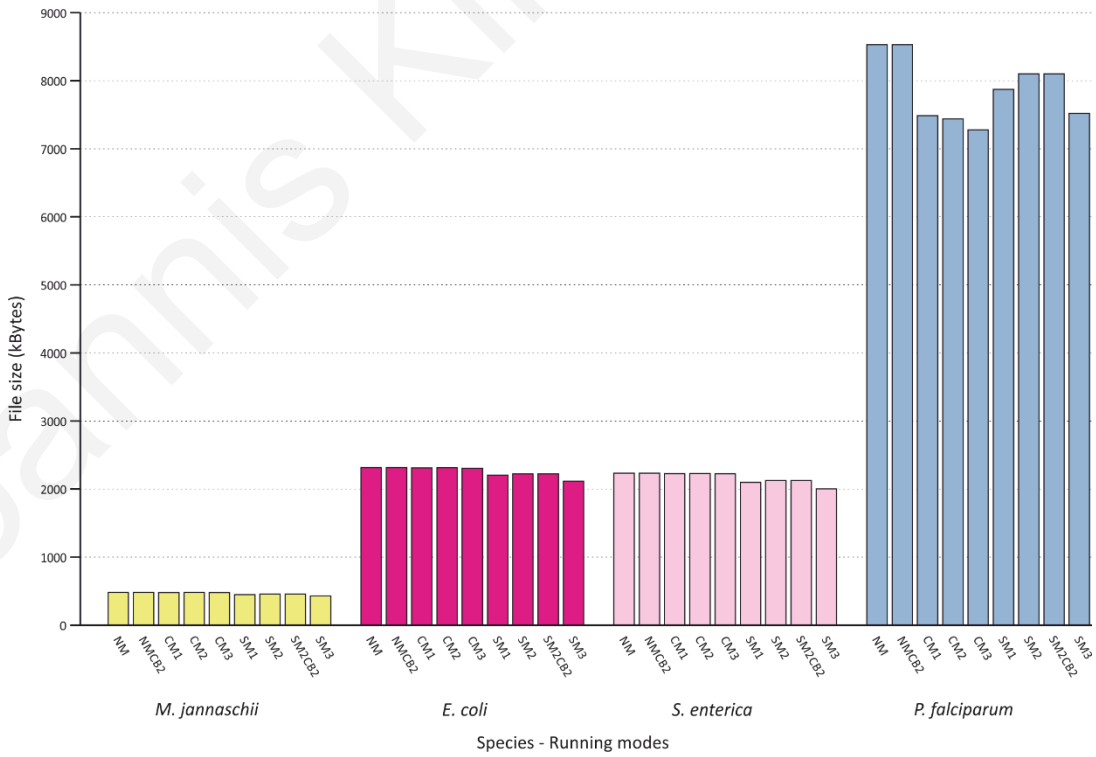


Figure 3-4. *Sizes of BLASTP all-against-all self-comparisons output files. For definitions of running modes refer to Table 2.*

ASTRAL-based comparisons

In general, run times for the ASTRAL-based benchmarks between the various running modes are similar, with the exception of the modes employing composition-based statistics (Figure 3-5). This difference can be easily observed when running BLASTP with an E-value threshold of 100, which is commonly employed for ASTRAL-based benchmarks, where NMCB2 and SM2CB2 are almost twice as slow compared to the other modes. This observation is in agreement with what other studies have pointed out (Altschul *et al.*, 2005; e.g. Schaffer *et al.*, 2001) and can be attributed to the inherent computational cost of compositionally-based statistics.

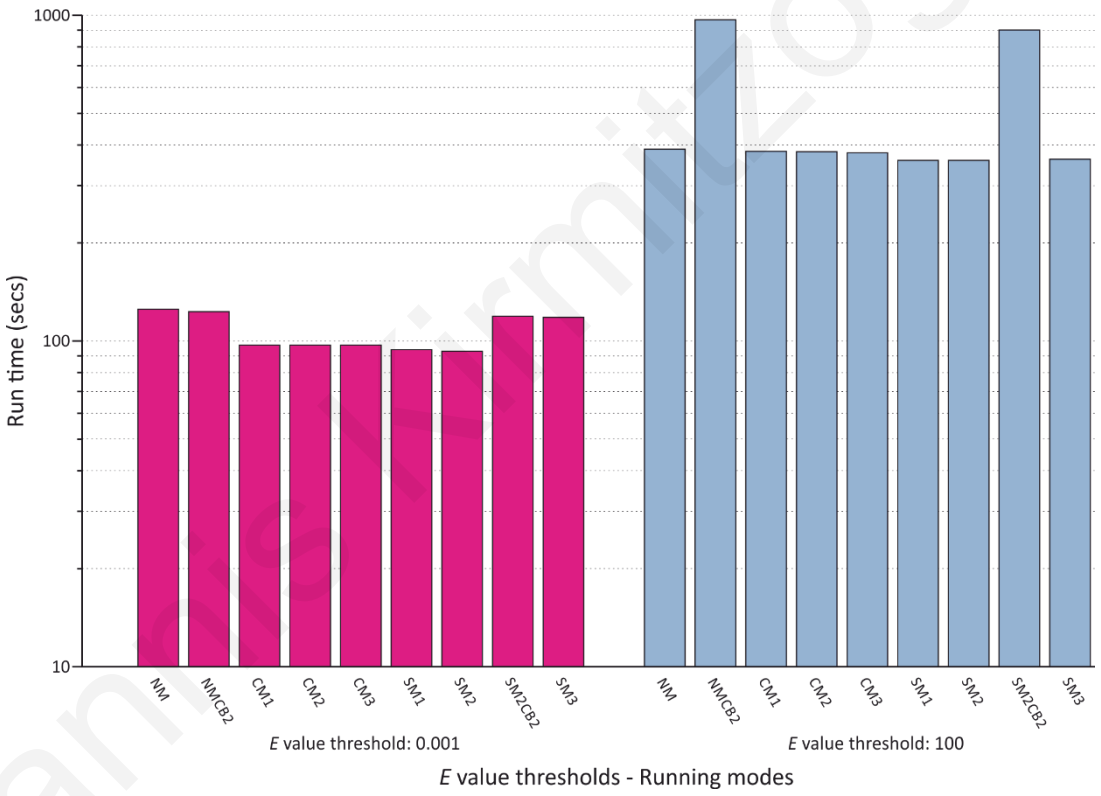


Figure 3-5. *Execution times for ASTRAL-based BLASTP comparisons with two different E-value thresholds; 0.001 (magenta) and 100 (cyan).*

For definitions of running modes refer to Table 2.

Output file sizes, and therefore number of returned results, are similar between masking modes (CM1-3 & SM1-3) and about 10% smaller for NMCB2 and SM2CB2 (Figure 3-4).

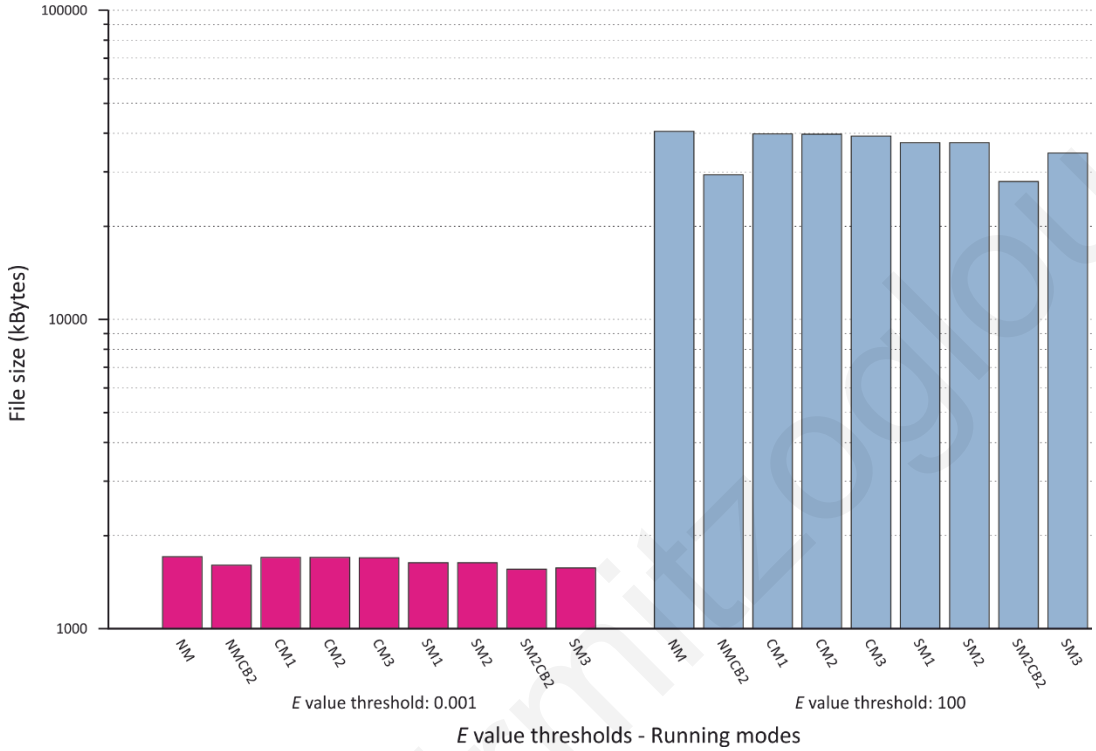


Figure 3-6. *File sizes for ASTRAL-based BLASTP comparisons with two different E-value thresholds; 0.001 (magenta) and 100 (cyan).*

For definitions of running modes refer to Table 2.

In general, we see similar trends between the ASTRAL-based benchmarks and the self-comparisons of *E. coli*, *M. jannaschii* and *S. enterica*, a direct result of their similarity with regards to their LCR content.

3.1.3 Performance of the various running modes

Full proteome self-comparisons

Using E-value as the only criterion, a significant number of proteins do not return themselves as the best result for all four genomes analysed (Figure 3-7A). Consequently, if those proteins were queries of a similarity search using the best bidirectional hit technique (Overbeek et al., 1999) their true orthologs would most probably be untraceable. Utilizing a series of additional features contained in the BLASTP output (bit score, percent identities and alignment length) combined with heuristics (see section 2.1.5) significantly improve the results (Figure 3-7A). Nevertheless, a significant portion of approximately 1% of proteins in each genome still remains undetected, with the two-way filtering masking modes (CM3 and SM3) being slightly more accurate than their one-way counterparts. That 1% are either sequences returning no hits at all (at the 0.001 E-value threshold) or sequences returning multiple identical hits, which may correspond to recently duplicated paralogous genes. This is most probably the case for all sequences of *E. coli, M. jannaschii* and *S. enterica*, as well as the *P. falciparum* sequences with no LCRs (Figure 3-7B), where the number of proteins with ties remains almost identical between the various running modes. The same seems to hold true for the number of sequences not returning any hits at all; these are probably sequences too short to produce any hits with significant scores.

Looking at the *P. falciparum* sequences with at least one LCR (Figure 3-7B), we see that SEG modes (SM1-3 & SM2CM2) are significantly less effective even compared against the NM mode, which does not employ any LCR-handling method. At the same time NM is as effective, if not more, as CM1-3 and NMCB2. And when taking into account the extra features from the BLASTP output file then all modes behave remarkably similar. Of course, this is to be expected since the specific experiment only checks the validity of the best hit for each sequence and ignores all other hits. At the same time, the fact that all running modes struggle to detect the correct hit based on E-value alone certainly demonstrates that LCRs can indeed cause serious issues to BLAST. Furthermore, even in datasets with almost no or no LCRs at all, BLASTP fails to detect the correct hit for a respectable fraction of the sequences, a reminder that BLAST is a heuristic algorithm and is not guaranteed to return the correct hit every time.

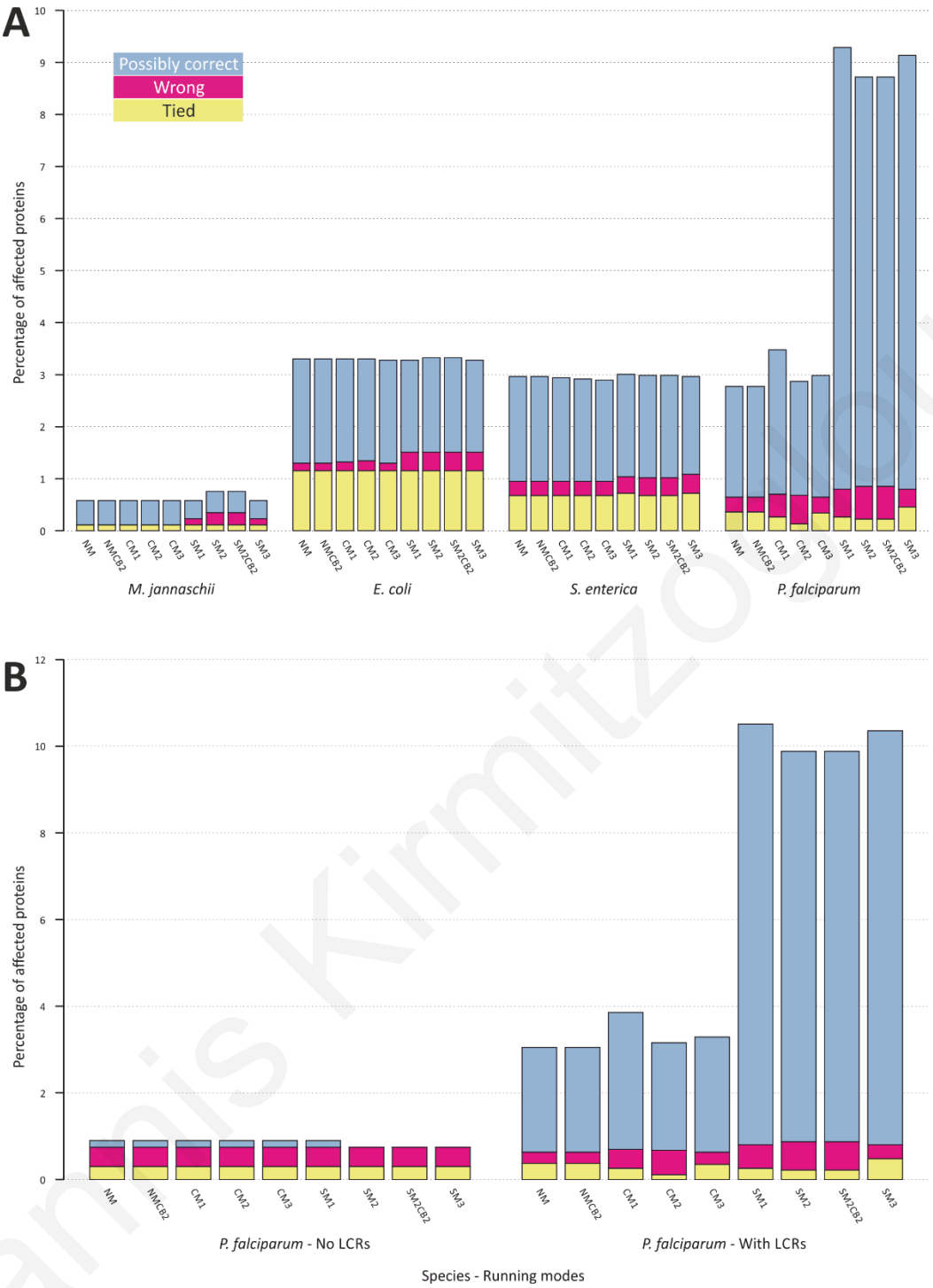


Figure 3-7. *Comparison of the results obtained by the various masking modes.*
*Possibly correct (**blue**) are sequences with multiple tied results according to E-value, that return themselves as the best hit when taking into account more features from the BLASTP output files. Ties (**yellow**) are sequences with multiple tied results even after taking into account all the extra features. Wrong (**magenta**) are sequences not returning themselves as the best hit even after taking into account all the extra features. A big portion of proteins within the "wrong" class does not return any hits at all.*
***A**: Selecting the best hit based on E-value only, returns a significant amount of false hits. By utilizing extra features (see section 2.1.5) the amount or wrong hits lessens but is still close to 1%.*
***B:** In the case of P. falciparum the amount of possibly correct hits (blue) drops almost to zero for sequences with No LCRs (according to SEG). For definitions of running modes refer to Table 2.*

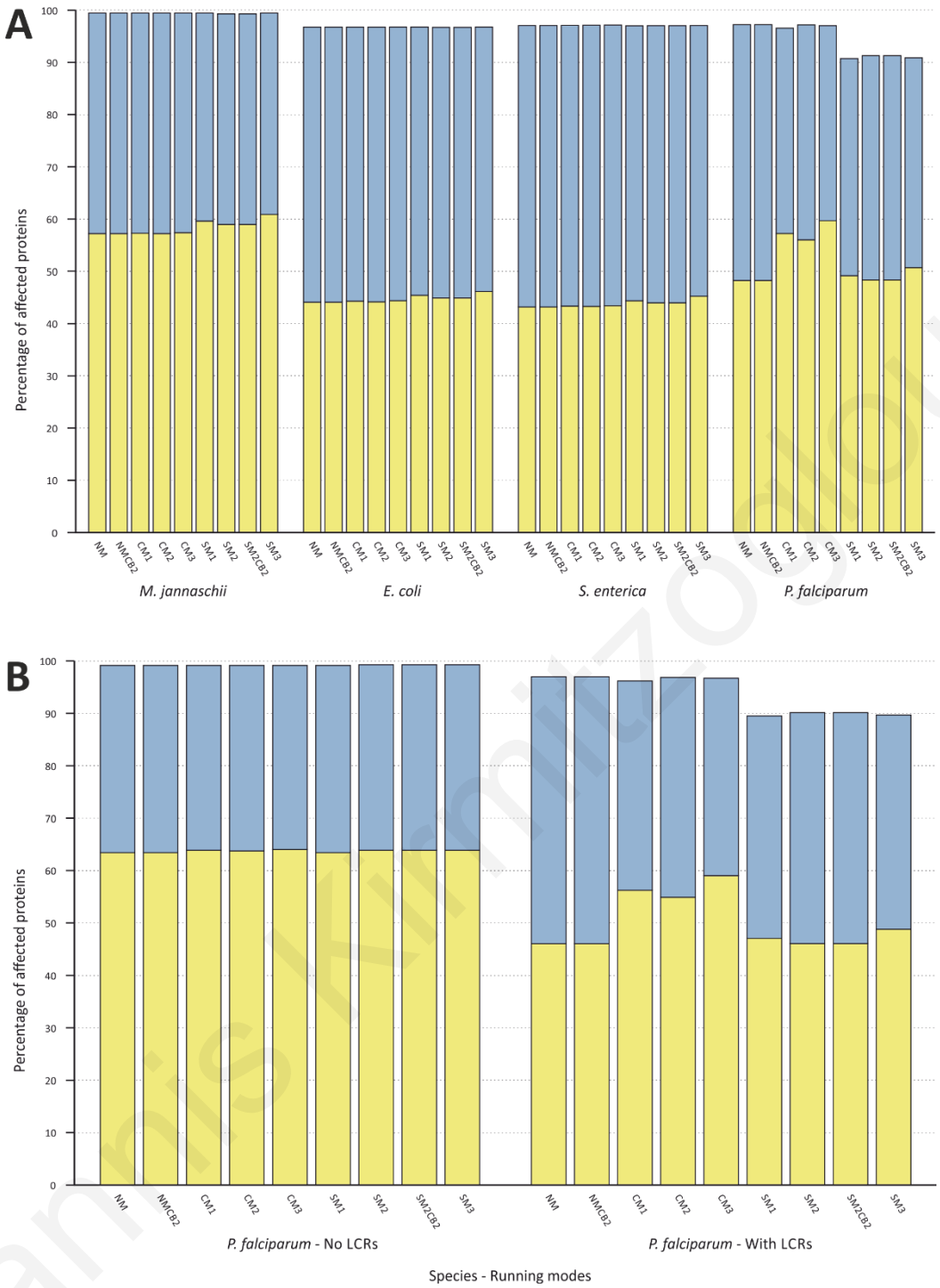


Figure 3-8. *Comparison of the results obtained by the various running modes.*

*E-value wins (**blue**) refer to sequences with multiple results but only one best hit (which is also the self-hit) according to E-value. Single wins (**yellow**) are sequences that return themselves as the only hit.*

***A**: All species. **B:** P. falciparum* sequences with no (left) or at least one (right) low complexity region.

For definitions of running modes refer to Table 2.

ASTRAL-based benchmarks

In general, we can classify searches against sequence databases into two categories (i) manual searches with a low E-value threshold and (ii) automated, *"deep"* searches with a much higher threshold. In the case of manual searches, the researcher is usually interested in a small number of results having the lowest E-values. As such, a good LCR-handling method for manual searches must have high sensitivity in the "biologically relevant" E-value zone ($e<10^{-3}$) (i.e. produce the most true positive results) while keeping the number of false positives at a minimum. The same criteria apply to *"deep"* searches although maintaining high specificity and sensitivity with higher E-value thresholds can be much more challenging. To better understand the behaviour of LCR-handling modes under both scenarios, most graphs in this chapter provide two views of the same data, for example Figure 3-11 shows the counts of true positive hits for each of the first 100 or 10000 false positive hits ranked by E-value.

Our results demonstrate a clear superiority of bare BLAST (mode NM) over the modes commonly used along with it (SM2, NMCB2). In the "biologically relevant" E-value zone ($e<10^{-3}$), NM also produces the smallest E-values (Figure 3-9A, Figure 3-10), which in the context of routine searches would mean more results, i.e. more sensitivity. This is further supported by the number of TP hits produced at given FP limits (Figure 3-11A), and the corresponding ROC*n* values (Figure 3-12, Table 5).

Also of interest is the notable performance of CAST masking modes (CM1-3) with ROC_{10000} identical to NM and considerably better than all the other modes (Figure 3-12, Table 5). When comparing ROC_{100} values the CM3 mode (2-way masking with CAST) is the clear (although marginal) winner (Table 5), while at the same time managing to produce smaller files (Figure 3-6) than those created by NM BLAST and in less time (Figure 3-5). In fact at this FP limit all CM modes are better than their counterparts, including NM.

It is noteworthy to point out that even though ROC_{10000} is the most common metric in such comparisons it barely carries any biological or practical meaning, especially in the context of manual searches. In our results it is apparent that even at the limit of 100 FP hits, E-values of the produced hits are already close to 0.01. Thus we consider ROC_{100} being a more representative metric for evaluations resembling routine search scenarios.

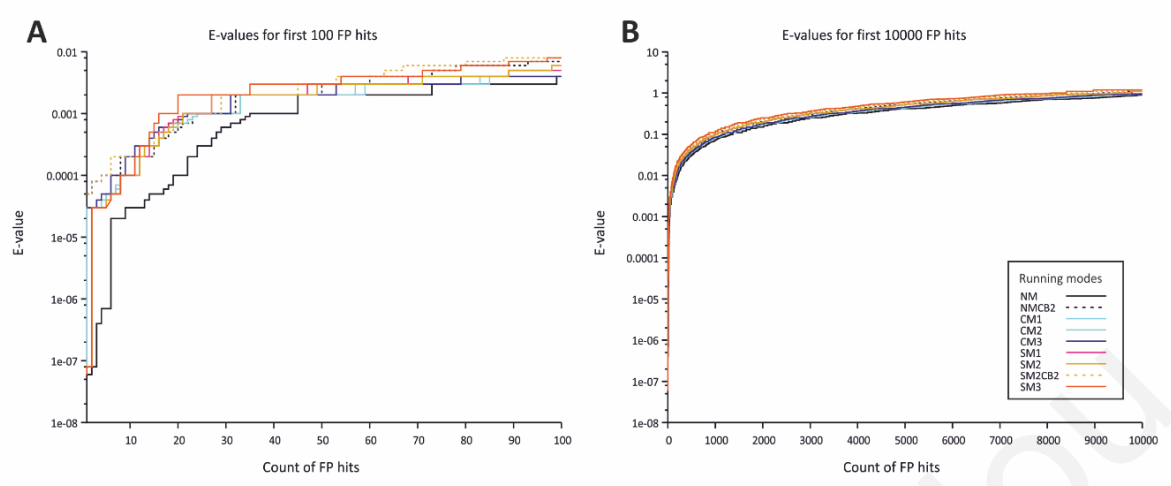


Figure 3-9. *E-value versus FP rank.*

E-values for the first FP hits up to rank N produced when comparing ASTRALNS$_{40}$ 1.73 data set against all sequences of ASTRAL$_{40}$ 1.73 using BLASTP. (A) N=100, (B) N=10000 FP hits encountered in this order were plotted. For the definition of ASTRALNS$_{40}$, running modes and FP hits in ASTRAL$_{40}$ context refer to sections 2.1.1 & 2.1.6 of "Data & Methods"

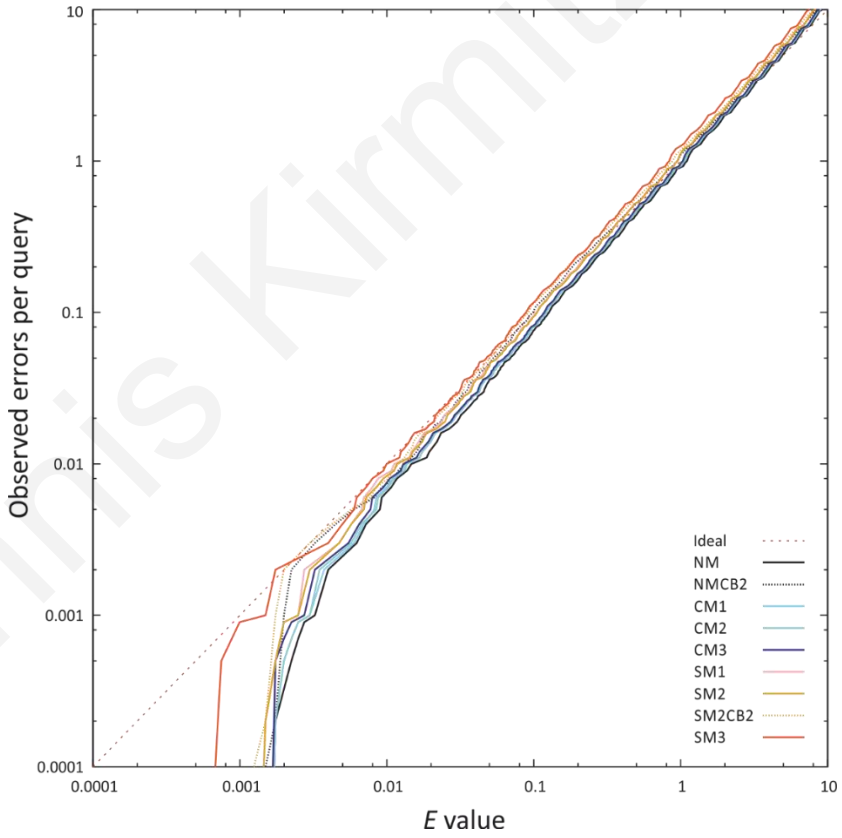


Figure 3-10. *Errors per query versus E-value.*

The number of observed errors per query is plotted against the statistical score at which they occurred. The diagonal dotted line represents an ideal scoring scheme. Methods below the dotted line are conservative in terms or the reported score. More conservative modes report more hits for a given E-value threshold.

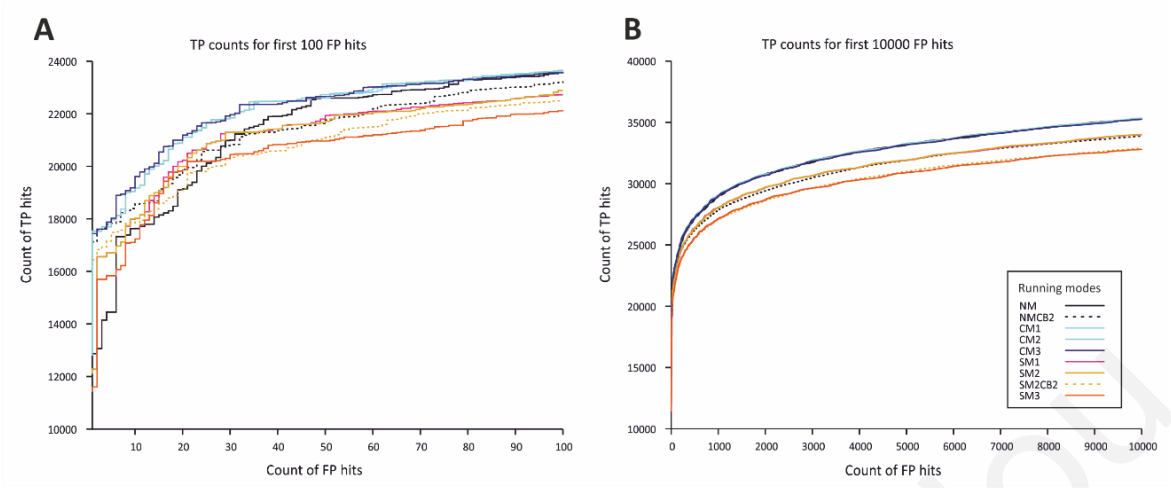


Figure 3-11. *Number of TP versus FP hits.*

Counts of the true positives (TP) hits for each ranked false positive (FP) hit produced when comparing ASTRALNS$_{40}$ 1.73 data set against all sequences of ASTRAL$_{40}$ 1.73 using BLASTP.

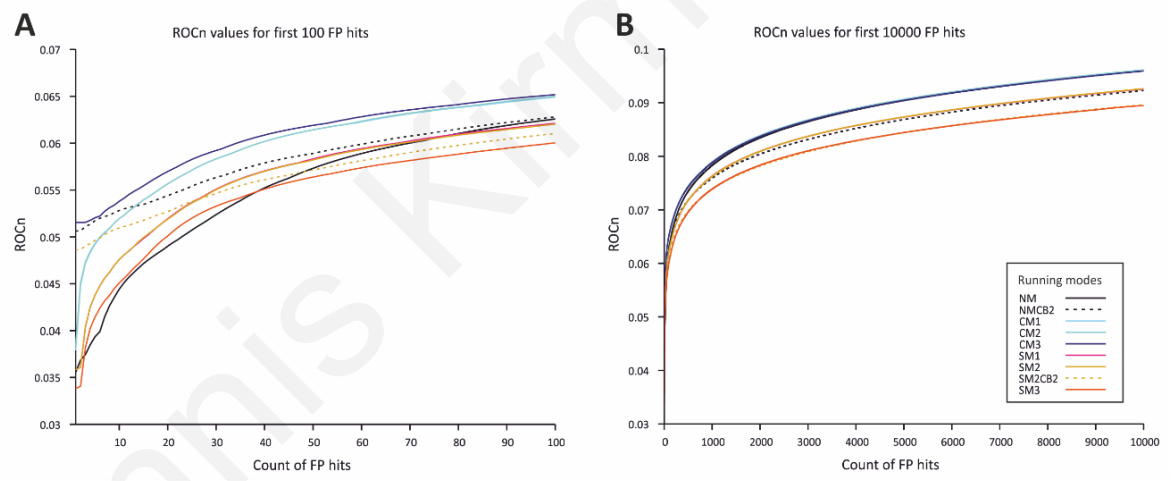


Figure 3-12. *Truncated ROC plots.*

Plots were calculated using the BLASTP output for the comparison of a subset of ASTRALNS$_{40}$ 1.73 data set against all sequences of ASTRAL$_{40}$ 1.73. (A) n=100 and (B) n=10000.

Table 5. *ROCn values and standard deviations for evaluated BLASTP modes.*

Running mode	ROC_{100}	ROC_{10000}
NM	0.0625 ± 0.0001	0.0959 ± 0.0001
NMCB2	0.0627 ± 0.0001	0.0922 ± 0.0001
CM1	0.0649 ± 0.0001	0.0961 ± 0.0001
CM2	0.0649 ± 0.0001	0.0961 ± 0.0001
CM3	0.0651 ± 0.0001	0.0959 ± 0.0001
SM1	0.0621 ± 0.0001	0.0925 ± 0.0001
SM2	0.0620 ± 0.0001	0.0926 ± 0.0001
SM2CB2	0.0610 ± 0.0001	0.0896 ± 0.0001
SM3	0.0600 ± 0.0001	0.0894 ± 0.0001

In order to address the effect of the presence of LCRs on our findings, we split the ASTRALNS$_{40}$ dataset into two disjoint subsets, based on the existence of LCRs, and repeated the comparisons for each subset. New types of graphical depictions (Figure 3-13 to Figure 3-16) help to gain a deeper understanding of how the various modes behave against sequences with and without LCRs. An expected observation (since ASTRAL is relatively poor on LCRs) came when using as queries sequences with no LCR content (Figure 3-15 & Figure 3-16 [3]) all methods perform in a strikingly similar manner to their performance on the overall data set (Figure 3-13 & Figure 3-16 [1]). However, despite the fact that the LCR containing subset is rather small, several different figures clearly indicate that the modes of choice from a biological point of view are CM2 and CM3 (Figures 3-14 & 3-15). These results suggest, that those should be the methods of choice when we perform searches against sequence databases such as NCBI nr[44] or UniProt/SwissProt[45], where LCRs are significantly more abundant. Importantly, our findings demonstrate that benchmarks using ASTRAL are biased against sequences containing LCRs, and this may have an impact on the trends followed for routine database searches.

[44] http://www.ncbi.nlm.nih.gov/
[45] http://www.uniprot.org/

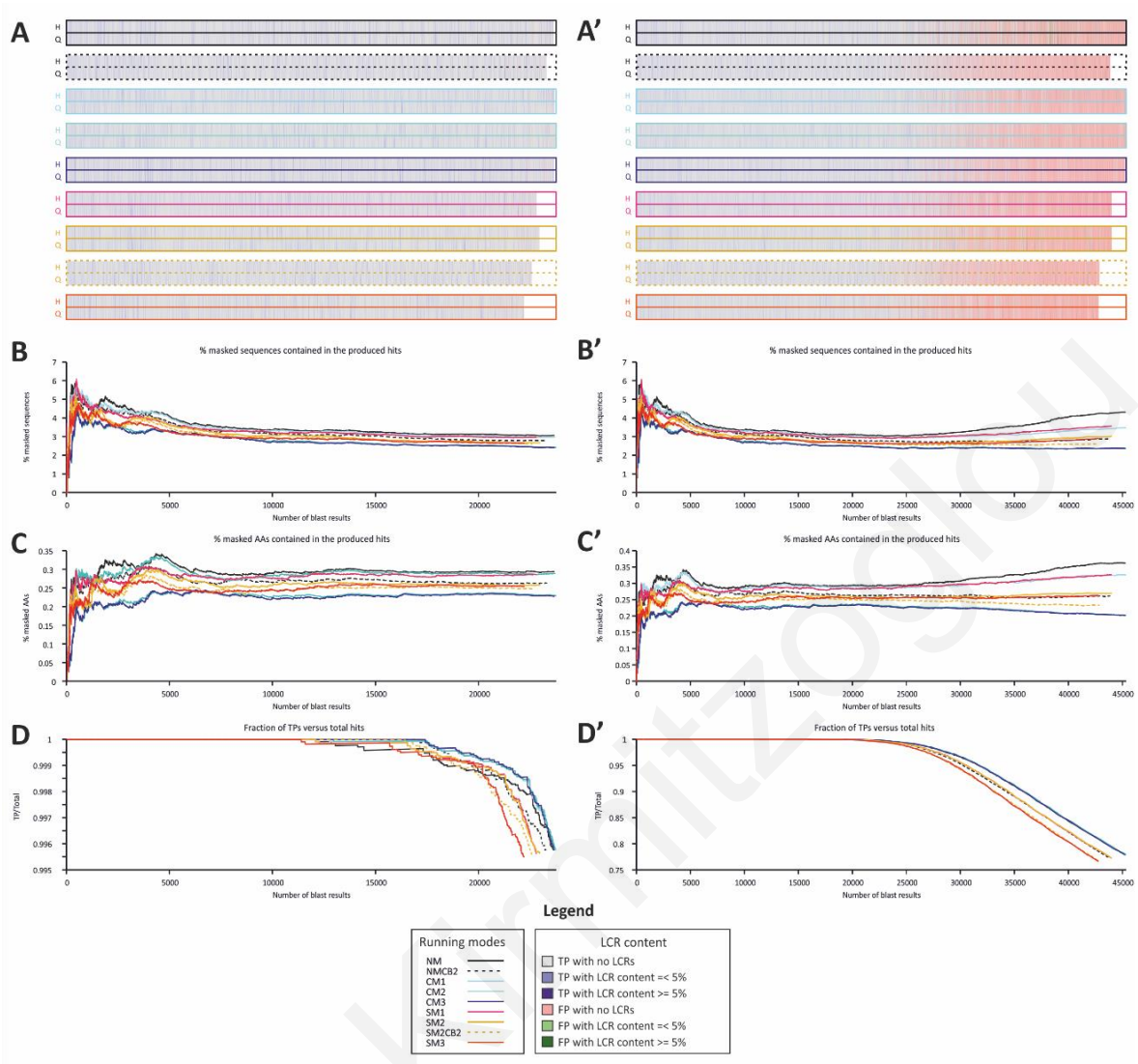


Figure 3-13. *Distributions of interesting properties versus the number of hits.*

Various characteristics for first hits up to rank N of FP produced when comparing ASTRALNS$_{40}$ 1.73 data set against all sequences of ASTRAL$_{40}$ 1.73. In this figure all sequences of ASTRALNS$_{40}$ 1.73 were used as queries, while in figures 3-14 & 3-15 only sequences with at least one LCR or no LCR (according to CAST) were used respectively. All the hits encountered up to FP N=100 (left), FP N=10000 (right) were plotted.

A & A': *Visualization of LCR content for each query-hit pair. H-Q (Hit-Query) pairs of rectangles correspond to a specific running mode (identified by the outline pattern - see legend). Each rectangle displays (from left to right) the LCR content (CAST) with respect to the state of the hit (TP or FP). Undetermined hits (see appendix) were ignored.*

B & B': *Percent masked sequences in the produced hits. Percentage of sequence hits having at least one LCR (CAST) versus the number of hits.*

C & C': *% masked amino acids contained in the produced hits. Percentage of amino acid residues masked by CAST in hit sequences versus the number of hits.*

D & D': *Fraction of TPs versus total hits.*

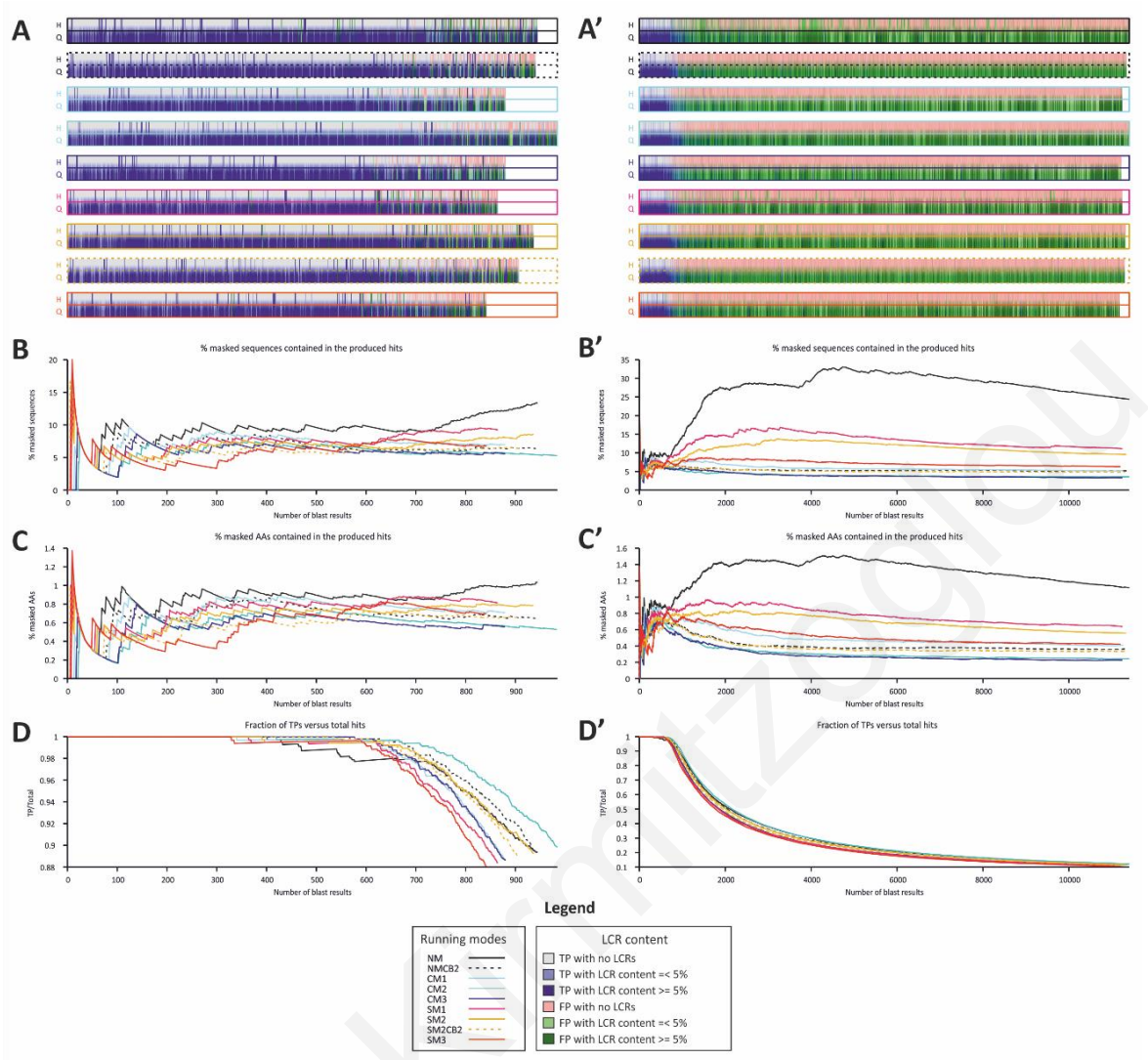


Figure 3-14. *Distributions of interesting properties versus the number of hits.*

Various characteristics for first hits up to rank N of FP produced when comparing ASTRALNS$_{40}$ 1.73 data set against all sequences of ASTRAL$_{40}$ 1.73. In this figure only sequences of ASTRALNS$_{40}$ 1.73 having at least one LCR (according to CAST) were used as queries, while in 3-13 & 3-15 all the sequences or only sequences with no LCR were used respectively. All the hits encountered up to FP N=100 (left), FP N=10000 (right) were plotted.

A & A': *Visualization of LCR content for each query-hit pair. H-Q (Hit-Query) pairs of rectangles correspond to a specific running mode (identified by the outline pattern - see legend). Each rectangle displays (from left to right) the LCR content (CAST) with respect to the state of the hit (TP or FP). Undetermined hits (see appendix) were ignored.*

B & B': *Percent masked sequences in the produced hits. Percentage of sequence hits having at least one LCR (CAST) versus the number of hits.*

C & C': *% masked amino acids contained in the produced hits. Percentage of amino acid residues masked by CAST in hit sequences versus the number of hits.*

D & D': *Fraction of TPs versus total hits.*

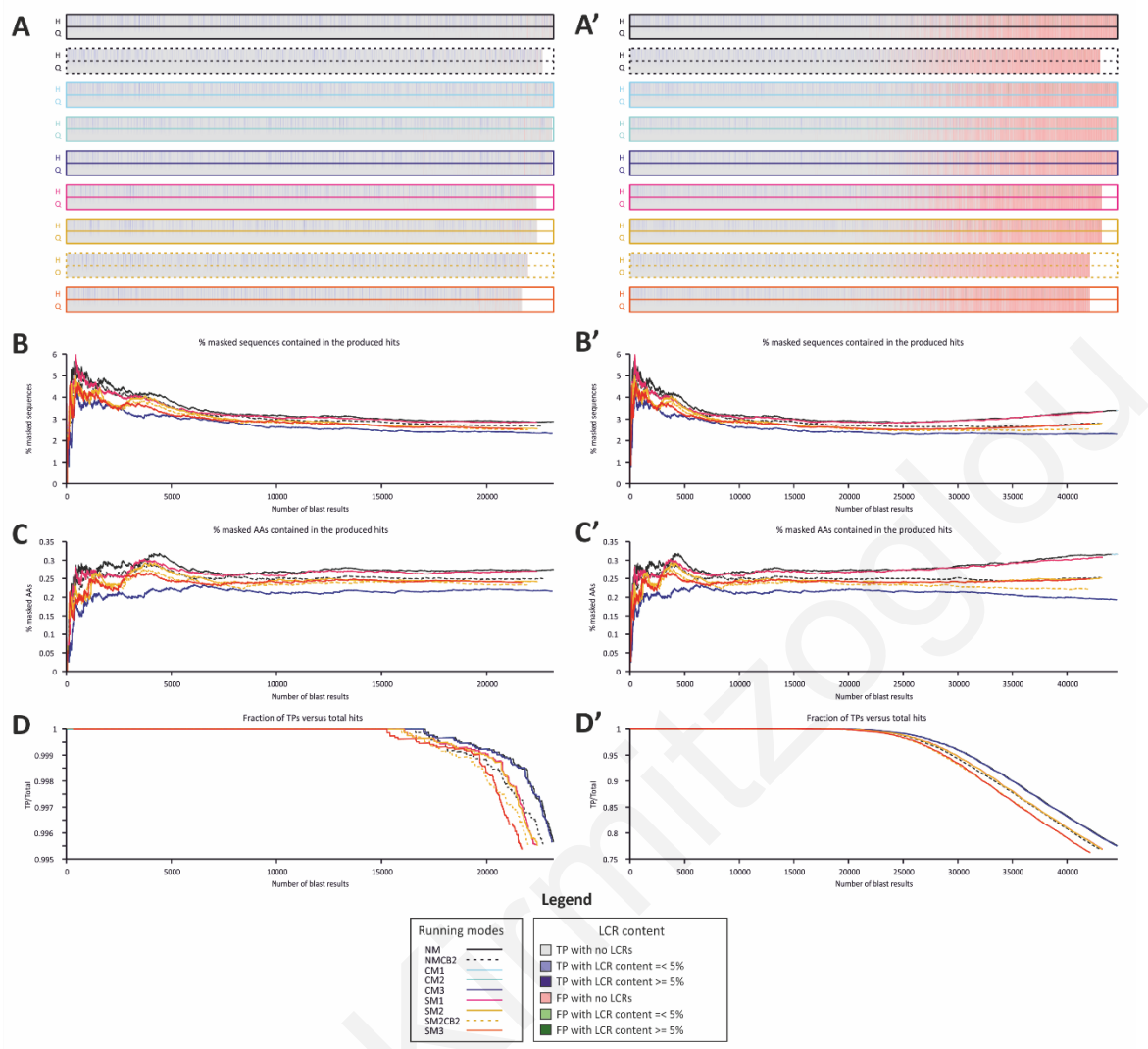


Figure 3-15. *Distributions of interesting properties versus the number of hits.*

Various characteristics for first hits up to rank N of FP produced when comparing ASTRALNS$_{40}$ 1.73 data set against all sequences of ASTRAL$_{40}$ 1.73. In this figure only sequences of ASTRALNS$_{40}$ 1.73 having no LCR (according to CAST) were used as queries, while in 3-13 & 3-14 all the sequences or only sequences with at least one LCR were used respectively. All the hits encountered up to FP N=100 (left), FP N=10000 (right) were plotted.

A & A': *Visualization of LCR content for each query-hit pair. H-Q (Hit-Query) pairs of rectangles correspond to a specific running mode (identified by the outline pattern - see legend). Each rectangle displays (from left to right) the LCR content (CAST) with respect to the state of the hit (TP or FP). Undetermined hits (see appendix) were ignored.*

B & B': *Percent masked sequences in the produced hits. Percentage of sequence hits having at least one LCR (CAST) versus the number of hits.*

C & C': *% masked amino acids contained in the produced hits. Percentage of amino acid residues masked by CAST in hit sequences versus the number of hits.*

D & D': *Fraction of TPs versus total hits.*

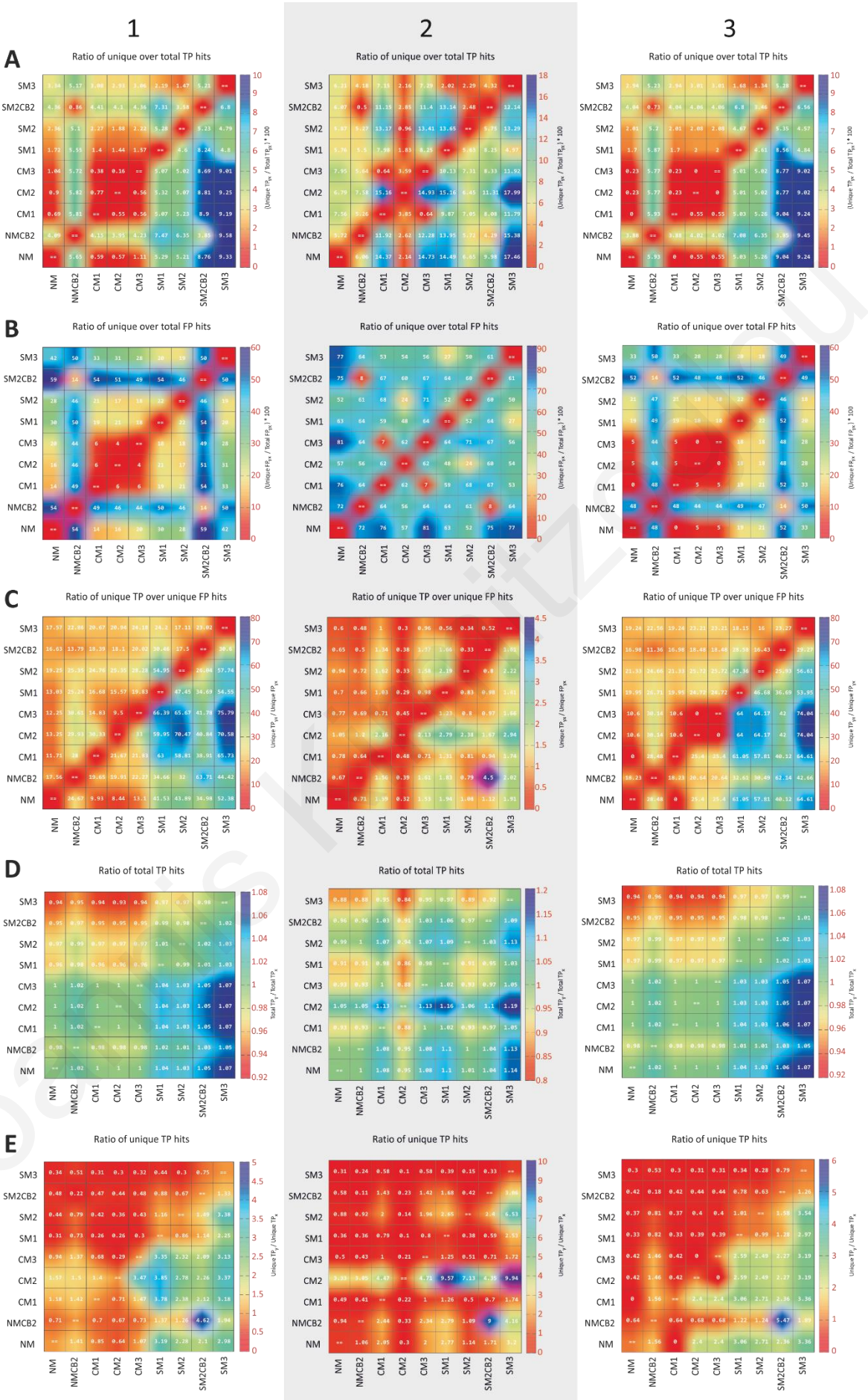

1
2
3
A
Ratio of unique over total TP hits
B
Ratio of unique over total FP hits
C
Ratio of unique TP over unique FP hits
D
Ratio of total TP hits
E
Ratio of unique TP hits
SM3
SM2CB2
SM2
SM1
CM3
CM2
CM1
NMCB2
NM

Figure 3-16. *Detailed comparisons of selected features between running modes.*

Squares in these heat map plots represent the ratios Fy/Fx, of features F for the modes on the y- and x-axis respectively. Only hits encountered up to FP N=100 were taken into account for the comparisons.

Features F correspond to:

A. *Ratio of unique over total TP hits*

The number of total TP hits for each y-axis mode is independent of the x-axis mode.

B. *Ratio of unique over total FP hits*

The number of total FP for all modes is equal to the FP limit N=100.

C. *Ratio of unique TP hits over unique FP hits*

D. *Ratio of total TP hits*

E. *Ratio of unique TP hits*

Data for sub-figures 1-3 are based on the same data sets used for figures 3-13 to 3-15 respectively. Colour scales are different and are depicted on the right of each heat map. An anti-aliasing filter has been employed for illustrative purposes.

3.2 LCR-eXXXplorer, a service to search, visualize and share low complexity regions in protein sequences.

3.2.1 Description of LCR-eXXXplorer

General Description

LCR-eXXXplorer (http://repeat.biol.ucy.ac.cy/mgb2/gbrowse/) is a web-service designed to assist biologists in displaying, searching and sharing low complexity regions contained within protein sequences. The service is built upon a modified instance of GBrowse (Stein *et al.*, 2002) and it currently contains 545,000 sequences (taken from the current version of UniProt/SwissProt) with more than 16 million LCR-related annotations. Biased regions are detected using two of the most commonly employed algorithms, namely SEG and CAST. Along with information about sequence complexity (SEG) or compositional bias (CAST), LCR-eXXXplorer displays external annotations from UniProt/SwissProt, as well as predicted disordered and binding regions by utilizing IUPRED (Dosztányi *et al.*, 2005a, 2005b) and ANCHOR (Dosztányi *et al.*, 2009; Mészáros *et al.*, 2009) respectively. LCR-related annotations are fully searchable through an advanced search form (which is a novel GBrowse plug-in specifically designed for this purpose, see paragraph "Search") and results can either be displayed in the browser or downloaded in a tab-delimited format for further processing. Users may also search for data in LCR-eXXXplorer using a local installation of BLASTP. Furthermore, users can initiate external BLASTP searches against the NCBI (http://www.ncbi.nlm.nih.gov/) or PDB (http://www.rcsb.org/) databases using as input query the currently displayed sequence after applying masking using any combination of amino acid residue types and detection algorithm.

Key functionality

While creating LCR-eXXXplorer we focused of four key issues: data, visualization, search and sharing/downloads. Each of these is described in detail in the following paragraphs.

Data

Features displayed through the LCR-eXXXplorer interface can be of 3 major types: (i) pre-calculated features, hosted on our webserver, (ii) features calculated on-the-fly and (iii) features from external sources, such as UniProt/SwissProt or generated by the user.

Pre-calculated features include low complexity regions detected by SEG and CAST (using default settings), as well as generic sequence-related information, such as sequence length, protein and gene name, organism and the sequence itself. These features are stored in a MySQL database for fast search and retrieval of data. The database schema is based on the SeqFeature schema that is used internally by GBrowse. We have adapted the schema to include one more table which contains –among others– counts of masked amino acid residues by type and masking algorithm (Figure 2-4). This change was necessary to allow us to implement the advanced search functionality.

With over half a million sequences stored in LCR-eXXXplorer, it makes sense to calculate some of the less requested features on-the-fly, i.e. just before displaying them to the user. Such features include the composition of each protein sequence per residue types, predicted disordered and binding regions (along with their scores) from IUPRED and ANCHOR, and GBrowse tracks that combine two or more stored features into one visually coherent piece of information (combo-tracks). The only downside of this approach is that the user cannot search for protein sequences containing such annotations, since there are not actually stored in the database (this is partly true for combo-tracks, since annotations for each sub-track remain searchable). On the other hand, it helps keep the database size smaller, thus guaranteeing that annotations requiring post-processing are always up-to-date with the current version of protein sequences in UniProt/SwissProt and LCR-eXXXplorer.

Displaying low complexity regions in a protein sequence might not be that useful unless there is a way to associate these LCRs with other regions that are linked to functional or structural features of the specific protein. By taking advantage of the underlying GBrowse capability to display features stored on a remote web accessible server, LCR-eXXXplorer incorporates selected annotations from UniProt/SwissProt into the main browser interface. UniProt/SwissProt annotations displayed in LCR-eXXXplorer are of two major types: (i) general annotations associated with the protein sequence (e.g. protein name, GO terms, PDB accession IDs) and (ii) position specific annotations, which may include domains, sites, secondary structure etc. (Table 6). These annotations are fetched from UniProt/SwissProt on demand for the protein sequence of interest, facilitated by a custom-designed cgi-bin script. The retrieved features are further post-processed to a format

suitable for the LCR-eXXXplorer. More details about how we handle and display UniProtKB-derived annotations are given in the following section.

Table 6. *UniProt/SwissProt annotations displayed in LCR-eXXXplorer and their organization in categories.*

More details on UniProt/SwissProt features are provided at the URL http://www.uniprot.org/manual/sequence_annotation. Category "Ignored features" contains annotation types that are excluded from LCR-eXXXplorer.

LCR-eXXXplorer category	UniProt/SwissProt tags
Amino acid modifications	Lipidation, Glycosylation, Disulfide bond, Cross-link
Molecule processing	Chain, Propeptide, Signal peptide
Repeats	Repeat, Compositional bias
Secondary structure	Helix, Beta strand, Turn
Sites	Active site, Metal binding, Binding site, Site
Topological Regions	Topological Domain, Transmembrane, Intramembrane
Others	Domain, Region, Coiled coil, Motif, Alternative sequence
Ignored features	Initiator methionine, Non-standard residue, Modified residue, Natural variant, Mutagenesis, Sequence uncertainty, Sequence conflict, Non-adjacent residues, Non-terminal residue

Using the same mechanism, LCR-eXXXplorer can display tracks generated by another instance of GBrowse, a Distributed Annotation System (DAS) server or valid GFF3 files generated by the user. The only requirement is that the remote tracks must use the same coordinate system, which in the case of LCR-eXXXplorer refers to the protein sequence itself.

Visualization

The main strength of LCR-eXXXplorer –setting it apart from similar services– is its visualization capabilities. The best way to explain the available visualization options is by describing the interface for an example protein sequence. Figure 3-17 shows the typical LCR-eXXXplorer entry for a protein sequence containing LCRs. Starting from the top we display the record summary in the form of a table that contains basic information regarding the example sequence, such as protein and gene names, organism, length, GO terms and links

to entries of associated 3D structures, if available in the Protein Data Bank. Some of these annotations are retrieved in real-time from UniProt/SwissProt while others are stored in the LCR-eXXXplorer database. Whenever a specific annotation type exists in both sources then we first try to obtain the entry from UniProt/SwissProt and if this fails we fall back to the internal entry. LCR-related annotations describing properties of the full sequence are also displayed here. Such annotations include the percentage of the sequence masked by SEG and CAST, as well as the amino acid composition of the protein sequence, which is displayed below the overall sequence composition for the LCRs-eXXXplorer database.

Just below the record summary, lies the main browser interface where a user can view location-specific annotations in a highly customisable manner. Annotations (or their groups) are displayed in horizontal strips, called tracks, which can be individually rearranged, modified, added or removed by the user. The very first track, called “Region” displays an overview of the protein sequence with residues masked by SEG and CAST marked as purple and magenta rectangles respectively. The “Region” track always displays the full length of the protein sequence and a user can click and drag on it to zoom and pan on a specific region of the sequence. All the other tracks always display the part of the sequence that is highlighted in the “Region”.

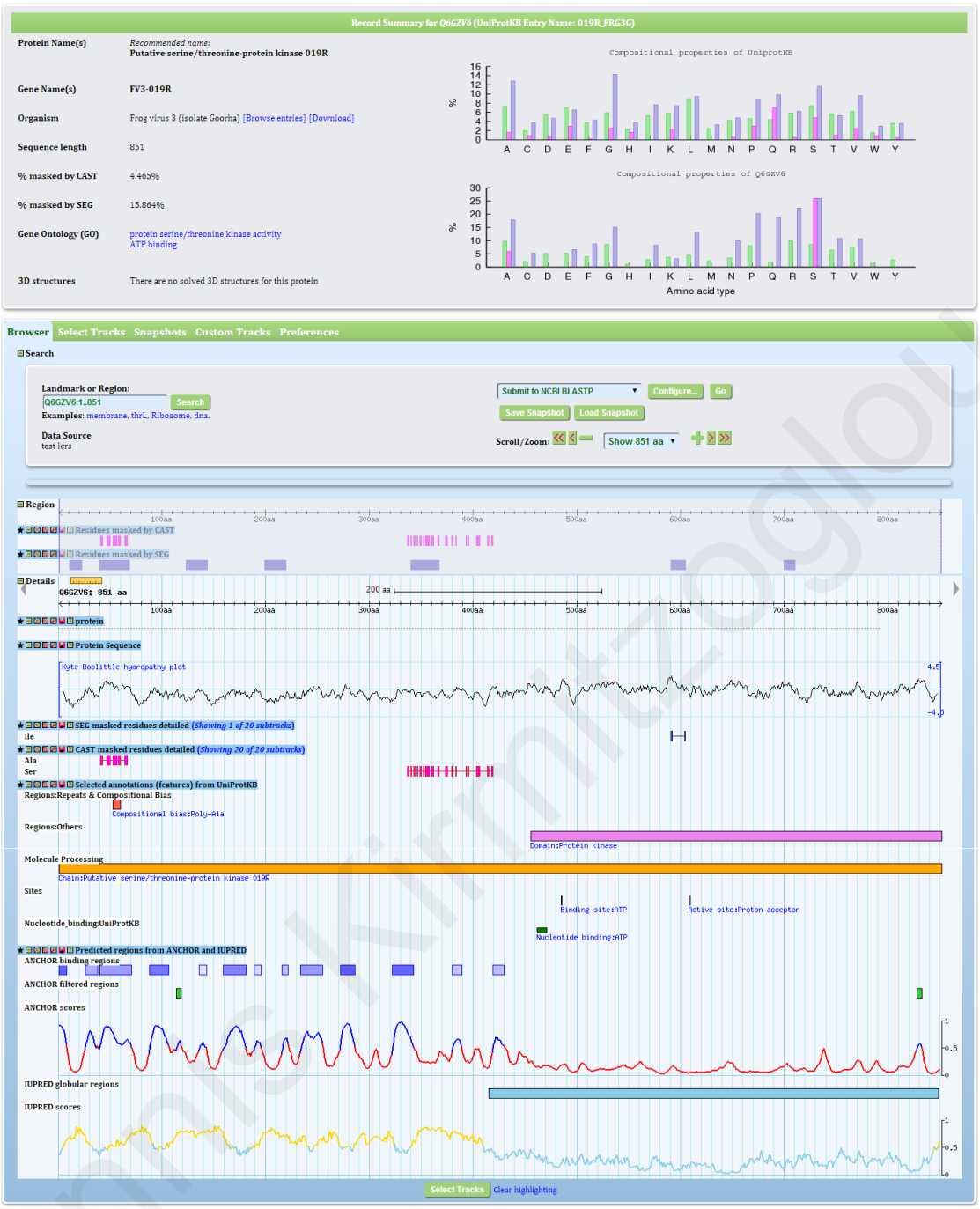


Figure 3-17. *The "protein details" view of LCR-eXXXplorer.*

On the top a "Record Summary" table provides general annotations about the protein sequence. On the right of the table a graph compares the amino acid composition of the sequence and the percentage of each residue type that is masked versus the whole database. On the bottom, the default tracks display annotations for LCRs-related properties of the sequence such as masked residues and regions predicted to be intrinsically disordered.

Default tracks

By default LCR-eXXXplorer displays the following tracks to a user:

1. Protein sequence: at low zoom levels this track displays a Kyte-Doolitle hydropathy plot, higher zoom levels reveal the amino acid sequence of the protein. Since

GBrowse was originally designed to display nucleotide sequences, we have created this special track type (called "glyph" internally) to make it compatible with protein sequences. Clicking on this track displays a pop-up window with the full protein sequence in text format.

2. Selected annotations (features) from UniProt/SwissProt: this track is generated on-the-fly using annotations pulled in real-time from UniProt/SwissProt. Using a cgi-bin script we group selected annotations in functional categories and generate a GFF3 file that is then served to GBrowse. Sequences stored in UniProt/SwissProt don't share the same types or number of annotations. For that reason, depending on the protein sequence, this track may contain one or more subtracks (in our example 5, Figure 3-17). Individual features are represented by coloured rectangles; clicking on a feature redirects the user to the relevant entry in the UniProt/SwissProt website.

3. Predicted regions from ANCHOR and IUPRED: since LCRs are commonly associated with intrinsically disorder regions in protein sequences (Dunker *et al.*, 2001; Romero *et al.*, 2001), having such information readily available can be very helpful. LCR-eXXXplorer utilizes IUPRED and ANCHOR, two algorithms that rely on pairwise energy estimation to predict disorder and binding regions respectively. IUPRED is designed to predict intrinsically disordered protein segments that are at least 30 residues long. For each residue in the protein sequence a disorder probability is given; residues with values greater than 0.5 are predicted to be disordered. These scores are displayed as a line graph in LCR-eXXXplorer; values greater than 0.5 are coloured gold while the rest are coloured cyan. IUPRED is also used to predict putative ordered (globular) regions. Neighbouring globular regions are merged together and if the final one is longer than 30 residues it is displayed as a cyan rectangle, just above the line graph.

4. SEG/CAST masked residues detailed: instead of displaying 20 different tracks with masked residues per amino-acid type, we group them together in a combo-track for each of the masking algorithms. Besides the aesthetical gain, this approach has a series of functional advantages, since the user has the option to control what residue types to display and in what order (Figure 3-18); at the same time, track

configuration options (such as glyph type or colour and position in the browser) apply to all the subtracks making customization fast and easy. Further customization of specific residue types is also possible by allowing the user to display each type as a separate track (see next paragraph).

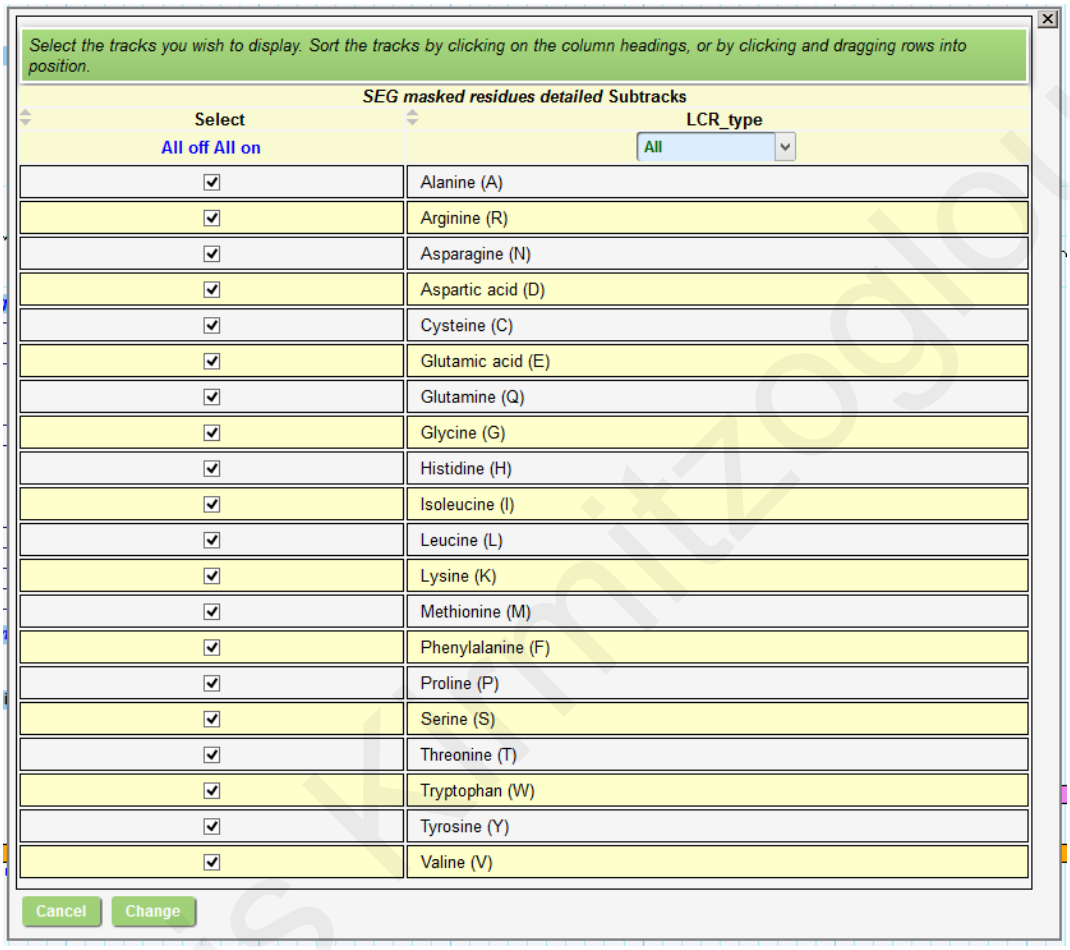


Figure 3-18. *Configuration options for a combo-track in LCR-eXXXplorer.*

The combo-tracks displays all the masked residues by SEG, grouped by residue type. Sub-tracks can be individually turned on or off, sorted by name or arranged manually in custom order.

Optional tracks

Besides the 5 default tracks, a LCR-eXXXplorer user has the option to display masked residues of a specific type as a separate track. These tracks are available in the track selection tab of LCR-eXXXplorer under the category “Masked residues per type”. Tracks representing LCRs rich in a specific amino acid residue type are available under the category “Low Complexity Regions”. These are different from masked residues, since a region rich in (e.g.) Proline most of the times contains other residue types as well. These tracks are generated using the output files produced by CAST when called with the –stat or –tab parameters. In summary, LCR-eXXXplorer provides 68 different types of annotations as

tracks plus the option to display an unlimited amount of custom, user-generated, tracks as described in section *"Data"* above.

Search

Search using sequence properties

Protein sequences stored in LCR-eXXXplorer are searchable by their general properties (such as accession number, organism, and protein or gene name) or LCR-specific properties. A search box is always available in the main browser tab where a user can search for protein sequences by their general properties. Searching for a UniProt/SwissProt Accession number or ID will instantly display the matching protein sequence, by automatically turning on wildcard suffixes. Using an incomplete Accession/ID will return in the results browser all sequences with a matching prefix in the respective field. A user may also search for keywords contained in the protein, gene or organism name. Wildcards are allowed and the number of the returned results is limited to 500. This simple search is powered by the built-in search of the GBrowse platform and works identically to every other service powered by GBrowse.

While the built-in GBrowse search works fine for searching protein sequences by their general properties, it is neither flexible nor fast enough to allow searches based on specific LCR-related properties of sequence entries in LCR-eXXXplorer. To overcome this issue we have incorporated a new table in the database schema and developed a brand new "Advanced Search" plugin that makes this table searchable (Figure 3-19). The "Advanced Search" is available in the plugins section of LCR-eXXXplorer. Selecting it from the drop-down menu and pressing "Configure" will reveal all the available search fields. The first four fields, (Accession, Gene name, Protein name and Organism) are similar to those available in simple search. The added functionality here is syntax checking (e.g. gene names must contain only letters, digits and "-"), targeted search against specific fields, and the ability to use the more advanced SQL-compatible wildcards for some of the fields.

"Advanced Search" also allows searching by sequence properties such as length and the percentage of the sequence length masked by CAST and/or SEG. Importantly, LCR-eXXXplorer allows a user to retrieve sequences with specific compositional bias type(s). Using a multiple selection list, a user can search for protein sequences locally enriched in one or more residue types. Complex search scenarios such as *"find all human protein se-*

quences between 200 and 800 residues long, rich in Proline and Leucine according to CAST and in Serine according to SEG" are also possible.

One limitation of the current "Advanced Search" implementation in LCR-eXXXplorer is that it does not allow for full Boolean searches, since it only returns results matching all search criteria. It also does not support searching based on other sequence properties such as the presence of specific domains. To compensate for these, LCR-eXXXplorer allows a user to input/upload a list of UniProt/SwissProt IDs and use them as search criteria. This way one can use other, more powerful, search engines –such as those available in UniProt/SwissProt or InterPro– and then perform LCR-specific searches within the results using LCR-eXXXplorer.

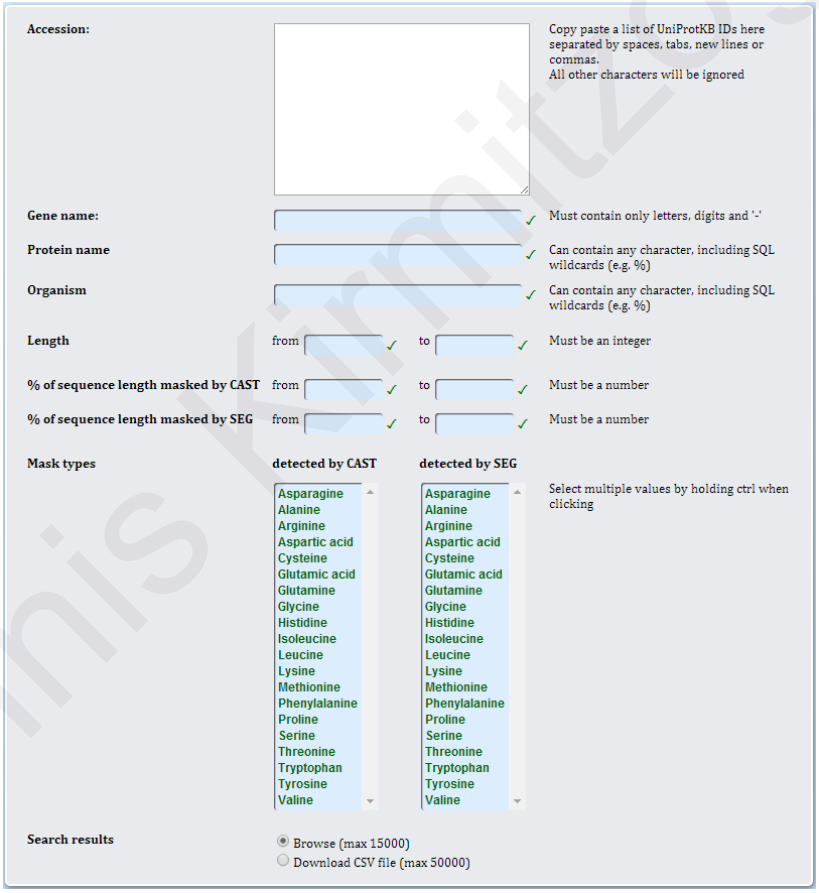


Figure 3-19. *The advanced search plugin*

BLAST search

LCRs-eXXXplorer supports BLASTP searches in two modes, internal and external. In internal mode a user can BLAST the LCR-eXXXplorer database for similar sequences (Figure XXX). Results are displayed in a simple and intuitive interface. Clicking on a single result, redirects the user to the main browser interface where the respective high-scoring seg-

ment part (HSP) is added as new track. What differentiates BLAST searches on LCR-eXXXplorer from a similar search in UniProt/SwissProt is the ability to search against masked databases (either with SEG or CAST) and with settings that are fine-tuned for spurious hits rejection. These settings are based on our results presented in section 3.1 above.

In external mode a user can use any sequence (or part of it) stored in the LCR-eXXXplorer as a query sequence for a BLASTP search on the NCBI servers. This capability is provided through a novel plugin specifically developed for LCR-eXXXplorer (Figure 3-20). To take full advantage of this functionality the user should first enable the desired annotation tracks in the tracks browser and select the part (or the whole length) of the sequence to be used as the query sequence for BLASTP. Selecting *"Submit to NCBI BLASTP"* in the plugins drop-down list and pressing configure will reveal the available options for external BLASTP searches. In the top section the user may soft mask the protein sequence using any of the enabled annotations tracks (only tracks showing internal LCR-eXXXplorer annotations are available). Tracks can be combined freely to create a soft mask that matches the user's needs (i.e. mask all positive charged [Arg, His, Lys] residues within CAST-detected LCRs and/or all negatively charged [Asp, Glu] residues masked by SEG).

The option to invert the applied soft mask is also available, essentially allowing the user to mask out all the non-LCRs residues. Doing so will enable searching against the NCBI databases for sequences having compositional bias patterns similar to the currently loaded sequence in LCR-eXXXplorer. This approach in BLAST search, to our knowledge, has never been documented before, but we have not experimented in detail with this option. For this type of search to work though, one should disable all types of LCR handling in NCBI BLASTP as discussed in the next paragraph.

In the lower section of the panel a series of LCRs-related BLASTP configuration options are available. These options are also available at the NCBI web-BLAST interface but are hidden by default. The exposed configuration settings are:

1. Filtering of low complexity regions: enabling this option will force BLASTP to apply SEG masking on the query sequence. This option is best to be kept off since it may

overlap with the applied soft mask. The default value for this option both in LCR-eXXXplorer and NCBI is OFF.

2. Mask lower case letters: by default BLASTP expects the query to be hard-masked, i.e. residues in LCRs to be replaced by character X. This option instructs BLAST to treat lower-case (i.e. soft-masked) characters in the sequence as masked residues. The default value for this option is ON in LCR-eXXXplorer and OFF in NCBI.

3. Mask for lookup table only: BLASTP is a two-step algorithm. In the first step candidate hits (seeds) are selected from a lookup table, which are then extended in the second step. When this option is selected, only the first step uses the masked query sequence and thus hits may potentially extend though low complexity regions. The default value is OFF as in NCBI BLASTP.

4. Compositional adjustments: As first demonstrated by Schaffer *et al.* (2001) amino acid substitution matrices can be adjusted to compensate for the different compositions of the sequences in comparison, resulting in more accurate E-values, especially in the cases of sequences with extreme composition. This approach has been repeatedly demonstrated (Altschul *et al.*, 2005; Schaffer *et al.*, 2001; Yu and Altschul, 2005; Yu *et al.*, 2003) to be more effective than SEG masking and is now the default option for LCRs handling in NCBI BLASTP. Normally, when searching against the NCBI databases, one would leave this option enabled; doing this when searching for sequences with specific compositional bias though, may limit the effectiveness of the search. For this reason the default option in LCR-eXXXplorer is *"No adjustment"*.

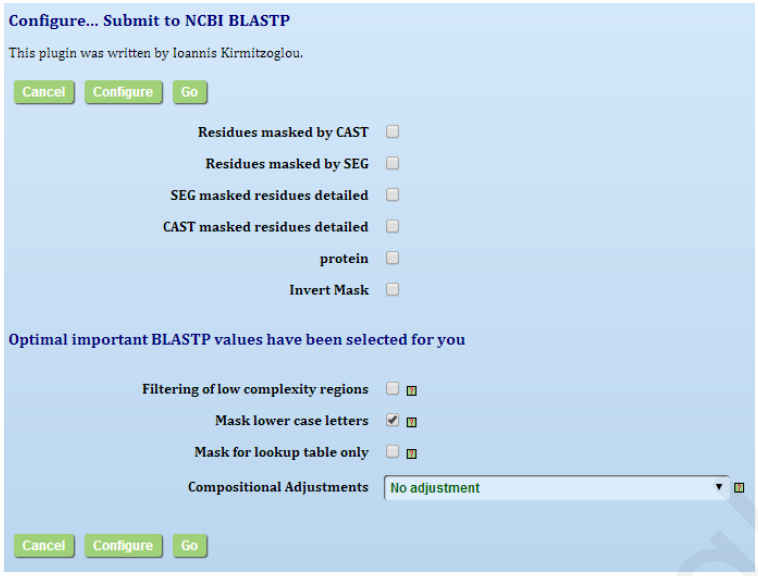


Figure 3-20. *The "Submit to NCBI BLASTP" plugin options*

Sharing/Downloads

An important aspect of LCR-eXXXplorer is that it provides users with the option to share and/or download LCR-related annotations for further processing. This can be achieved in several ways some of which are available on all GBrowse-based web-services, while others are unique to LCR-eXXXplorer. The easiest and simplest way to download all the detected LCRs of a protein in a Generic Feature Format (GFF) version 3 compliant file is by using the menu option *"File > Export as… > …GFF annotation table".* The same menu section also offers downloads in FASTA sequence format as well as a cleaner version of the tracks browser in Portable Network Graphics (PNG) format for use in publications (Figure 3-21). Downloads in FASTA, GFF3 & GenBank (actually GenPept) format for all of stored annotations or for each track separately are also available through the *"Download this track"* command, available at the header of each track (Diskette icon). Similarly, the option to *"Share"* (Radar icon) each track with other GBrowse-enabled web-services is also available.

Using either the *"Download decorated FASTA file"* or the *"Download masked FASTA file"* plugins a user has the option to download a FASTA file containing a custom selection of annotations. The first plugin can overlay any of the currently displayed tracks on top of the standard FASTA sequence with lower-case characters; depending on the output format, annotations can be represented as capital letters or with any combination of italics, bold, underlined and/or coloured characters (Figure 3-22A). The *"Download Masked FAS-*

TA file" plugin was designed and built specifically for LCR-eXXXplorer; it allows the user to download any combination of LCR-related annotation as a (soft or hard) masked FASTA file that can be used as input in other bioinformatics tools. Besides letting the user to apply any of the annotations tracks as either a soft or hard mask, it also offers the option to invert the mask. Such capability can help when searching for sequences having similar types of low complexity regions.

Another way to download files from LCR-eXXXplorer is through the *"Advanced Search"* plugin (Figure 3-19). Selecting *"Download CSV file"* will write search results on a tab-delimited text file. The text file contains 47 columns with annotations such as Accession, Name, Description, length and the portion of the sequence masked by SEG or CAST (Table 7). For each of the matching proteins, counts of masked amino acid residues grouped by type and algorithm are also provided. Using this mechanism, users can download such files for all the sequences matching specific criteria (e.g. all human protein sequences) and post-process them using other tools. Finally, a shortcut to download (or browse) all the sequences of a specific organism is also provided inside the record summary table for every sequence stored in LCR-eXXXplorer.

Table 7. *Description of the fields (columns) in the CSV files available for download in LCR-eXXXplorer.*

Column #	Header(s)	Description(s)
1	load_id	UniprotKB Accession
2	gene	Gene name
3	Note	Protein name
4	Organism	Organism name
5	length	Protein sequence length
6	perCAST	Percentage of protein sequence length masked by CAST
7	perSEG	Percentage of protein sequence length masked by SEG
8 - 27	cX*	Count of residues of type X masked by CAST
28 - 47	sX*	Counts of residues of type X masked by SEG

*** All of the 20 standard amino-acid types in ascending lexicographical order.**

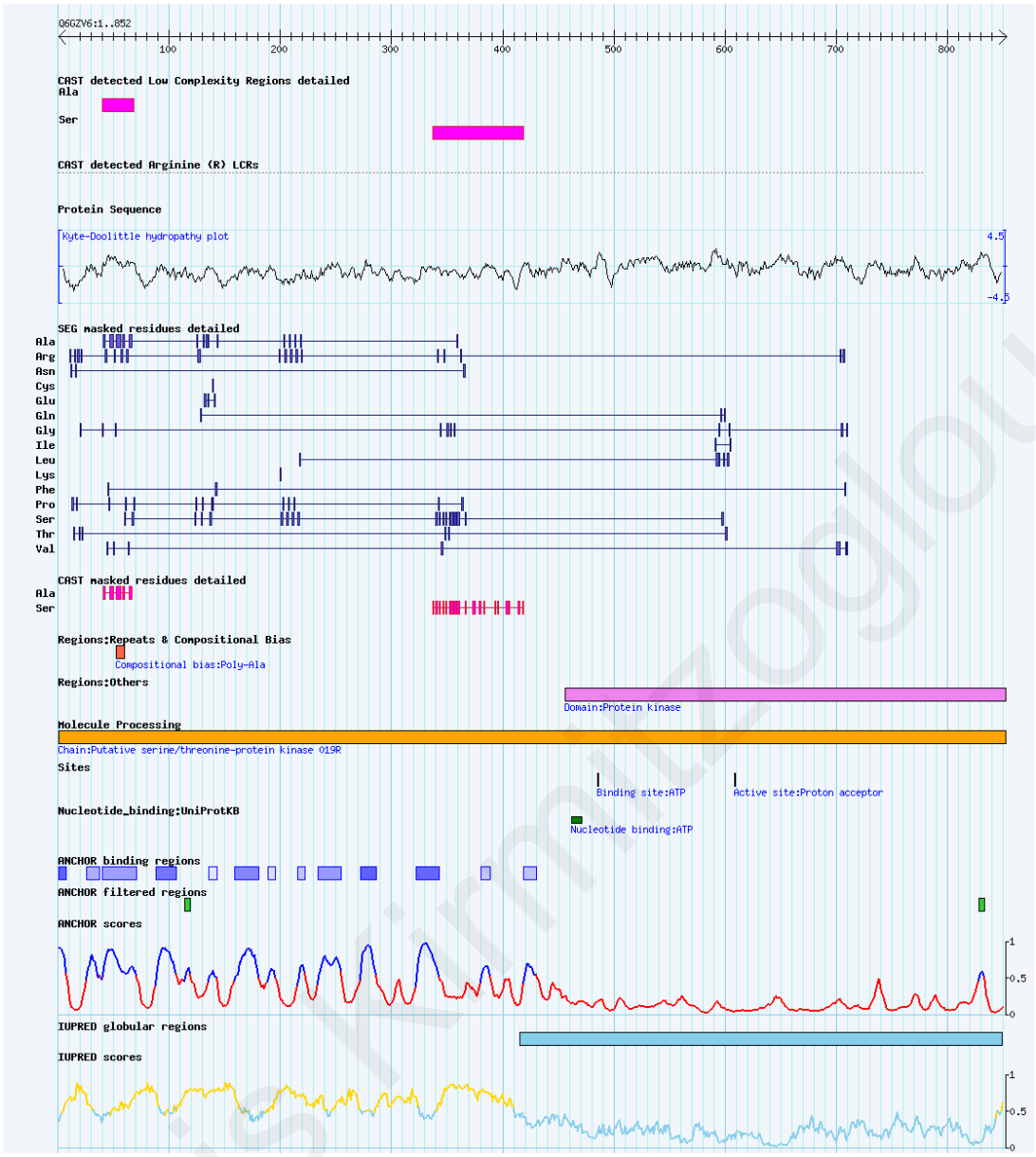


Figure 3-21. *Exporting to PNG in LCR-eXXXplorer.*

A PNG image of the tracks browser in LCR-eXXXplorer obtained using the "Export As low-res PNG format" command.

A

```
>Q6GZV6:1..851
matnycdeferntprtrprtgrtikrggpvfralerecsdgaarvfpaaavrgaaaaraas
prvaaaspcpefardptrnprtgrpikrggpvfralerecadyggasprrvsparafpnr
rvsparrqspaeaaeaspcpefardptrnprtgrtikrggptyraleaecadygrlspir
spwsdwsstglspfrshmrksparrsparrslaryteh1tsdsetevdydarnvi
rsqvgpggvcerfaadptrnpvtgsplsrndplytdlmeickgypdtpltksltgegtdd
dtceafcrdptrnpvtgqkmrrngieyqmfaeecdcSGISRPSGVSRTSGTSGSSGSSAS
SRPPNSFEAPGASSRPPNSFEASGAARVPGTPSVSRGEPRWMSSISTRHNYDESNPMSva
frlrhvkdirkflrtvrpgrsgfcatdkggwlgsaavsdnvigqgswgsvhmvkfrdfpe
efvvkeavlmsvsekhrykptvvwdewaagsvpdevvvnnmvteiaatgmtpfvpltaga
gacdscnpqllekaakvtkcylqameaadfsldrvlptmspdqaasalaqillglqslqt
tlgimhndikahnilvkrvppggywkvtdsfngqvfyipnegylcmladygvvrlvkpav
gmdtlygtrnarfvprdvgrwgkgagteyvvtpirskisvvvrggrfvgvepnkavrywk
ntdtskvgdvittnnvfymgydiepdmqvqlddtnsfpvwesrgdvadcvrtfvggkras
qpgfhrlfykktgsawekaaetvakqnplfsgftldgsglkyiraatacayifpgmavpr
pgereiesftm
```

B

```
>Q6GZV6:1..851  inverted_mask
matnycdefeRNPTRNPRTGRTikrggpvfralerecsdGAARVFPAAAVRGAAAARAAS
PRVAAASPcpefardptrnprtgrpikrggpvfralerecadyggasprrvsparafpnr
rvSPARRQSPAEAAEASPCPEFArdptrnprtgrtikrggptyraleaecadygrlspir
spwsdwsstglspfrshmRKSPARRSPARRSPARRSLARytehltsdsetevdydarnvi
rsqvgpggvcerfaadptrnpvtgsplsrndplytdlmeickgypdtpltksltgegtdd
dtceafcrdptrnpvtgqkmrrngieyqmfaeecdcsgiSRPSGVSRTSGTSGSSGSSAS
SRPPNSfeapgassrppnsfeasgaarvpgtpsvsrgeprwmssistrhnydesnpmsva
frlrhvkdirkflrtvrpgrsgfcatdkggwlgsaavsdnvigqgswgsvhmvkfrdfpe
efvvkeavlmsvsekhrykptvvwdewaagsvpdevvvnnmvteiaatgmtpfvpltaga
gacdscnpqllekaakvtkcylqameaadfsldrvlptmspdqaasalaqILLGLQSLQT
TLGlmhndikahnilvkrvppggywkvtdsfngqvfyipnegylcmladygvvrlvkpav
gmdtlygtrnarfvprdvgrwgkgagteyvvtpirskisVVVRGGRFVGVepnkavrywk
ntdtskvgdvittnnvfymgydiepdmqvqlddtnsfpvwesrgdvadcvrtfvggkras
qpgfhrlfykktgsawekaaetvakqnplfsgftldgsglkyiraatacayifpgmavpr
pgereiesftm
```

Figure 3-22. *Download (A) decorated & (B) masked FASTA files.*

3.3 Predicting the pathogenicity of *Escherichia* strains based on local and global amino acid compositional signatures

3.3.1 Clustering of *Escherichia* strains based on global and local compositional signatures

We masked the 22 verified genomes with CAST using 2 different score thresholds (15 and 40) and generated the global (GC), LCRs (LB) and masked residues (XB) compositional signatures as described in section 2.3.3 of *"Data & Methods".* Following that, and for each of the 22 strains, we generated 100 randomized sub-sampled artificial "proteomes" with varying numbers of proteins (100, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000)[46].

We then clustered each of the 900 sub-samples using both hierarchical (Ward) and k-means clustering. Following that, we calculated the similarity of each cluster to the simple grouping of the 22 strains based on their pathogenicity. Since in this study we only deal with two types of pathogenicity (i.e. pathogenic and non-pathogenic strains) we cut each tree generated by the Ward hierarchical clustering at the appropriate height needed to produce two main branches. In k-means clustering we achieved the same result by using 2 as the value of k. The measure used to calculate similarity was SIM as it was described by Promponas (2009). Following the approach of Promponas (2009), we calculated SIM twice for each cluster after altering the pathogenicity status of the environmental *E. coli* strain from non-pathogenic (SIM) to pathogenic (SIM+). SIM and SIM+ results averaged over gene numbers, gene-count classes, type of compositional signature and CAST score threshold are presented in Table 8.

Contrary to Promponas, SIM+ was found to be consistently smaller than SIM. This may be attributed to the fact that we excluded 2 strains (*E. coli* ATCC 8739 and *E. fergusonii ATCC 35469*) from our analysis since we couldn't verify their pathogenicity (both strains were marked as pathogenic by Promponas) (Promponas, 2009). In fact when taking these strains into account SIM+ was greater than SIM in most of the high gene-count classes (data not shown).

[46] For more details refer to section 2.3.5 of "Data & Methods".

Table 8. *Mean similarity (SIM) scores for all generated sub-samples.*

Following the approach of Promponas (2009) we calculated SIM twice for each class of gene-counts: (i) E. coli SMS-3-5 strain was considered as non-pathogenic (SIM) and (ii) the same strain was considered as pathogenic (SIM+). Standard deviation values for all scores are given in parentheses. Best scores for each gene numbers class as marked blue. GC: global composition; LB: LCRs composition; XB: Masked Residues composition; All: GC, LB and XB combined into a single vector.

Method	Signature	Genes count									
		100		500		1500		2500		4000	
		SIM	SIM+	SIM	SIM+	SIM	SIM+	SIM	SIM+	SIM	SIM+
CAST 40											
Ward	All	0.609 (0.052)	0.607 (0.056)	0.603 (0.056)	0.598 (0.073)	0.633 (0.079)	0.634 (0.082)	0.668 (0.076)	0.662 (0.084)	0.714 (0.086)	0.709 (0.052)
	GC	0.607 (0.059)	0.601 (0.079)	0.657 (0.090)	0.648 (0.101)	0.717 (0.100)	0.702 (0.100)	0.793 (0.078)	0.771 (0.071)	0.869 (0.033)	0.867 (0.059)
	LB	0.609 (0.051)	0.605 (0.067)	0.607 (0.061)	0.610 (0.068)	0.619 (0.090)	0.620 (0.083)	0.660 (0.102)	0.658 (0.099)	0.758 (0.094)	0.757 (0.051)
	XB	0.609 (0.049)	0.599 (0.058)	0.607 (0.052)	0.591 (0.067)	0.633 (0.069)	0.631 (0.070)	0.649 (0.064)	0.662 (0.081)	0.694 (0.080)	0.691 (0.049)
k-means	All	0.600 (0.054)	0.594 (0.066)	0.606 (0.061)	0.605 (0.078)	0.638 (0.075)	0.623 (0.087)	0.670 (0.080)	0.665 (0.084)	0.692 (0.078)	0.705 (0.054)
	GC	0.610 (0.069)	0.604 (0.084)	0.655 (0.085)	0.651 (0.092)	0.709 (0.098)	0.691 (0.097)	0.718 (0.090)	0.702 (0.081)	0.786 (0.084)	0.741 (0.069)
	LB	0.604 (0.055)	0.595 (0.058)	0.612 (0.077)	0.607 (0.072)	0.621 (0.091)	0.614 (0.092)	0.677 (0.098)	0.678 (0.090)	0.741 (0.093)	0.731 (0.055)
	XB	0.598 (0.055)	0.592 (0.065)	0.600 (0.056)	0.598 (0.076)	0.633 (0.083)	0.629 (0.073)	0.657 (0.070)	0.645 (0.077)	0.691 (0.085)	0.685 (0.055)

Method	Signature	Genes count									
		100		500		1500		2500		4000	
		SIM	SIM+	SIM	SIM+	SIM	SIM+	SIM	SIM+	SIM	SIM+
CAST 15											
Ward	All	0.613 (0.056)	0.609 (0.066)	0.640 (0.089)	0.635 (0.074)	0.693 (0.093)	0.691 (0.100)	0.742 (0.100)	0.741 (0.099)	0.771 (0.079)	0.772 (0.056)
	GC	0.608 (0.060)	0.604 (0.085)	0.651 (0.090)	0.642 (0.091)	0.720 (0.098)	0.696 (0.096)	0.803 (0.084)	0.793 (0.074)	0.862 (0.037)	0.866 (0.060)
	LB	0.608 (0.057)	0.603 (0.069)	0.630 (0.076)	0.619 (0.074)	0.668 (0.082)	0.656 (0.093)	0.709 (0.097)	0.695 (0.098)	0.773 (0.080)	0.746 (0.057)
	XB	0.615 (0.056)	0.606 (0.069)	0.628 (0.084)	0.629 (0.083)	0.697 (0.092)	0.701 (0.096)	0.738 (0.093)	0.743 (0.097)	0.761 (0.082)	0.775 (0.056)
k-means	All	0.614 (0.073)	0.604 (0.080)	0.636 (0.093)	0.628 (0.083)	0.708 (0.081)	0.701 (0.098)	0.741 (0.084)	0.736 (0.089)	0.748 (0.085)	0.730 (0.073)
	GC	0.608 (0.064)	0.611 (0.089)	0.651 (0.085)	0.647 (0.081)	0.715 (0.085)	0.686 (0.093)	0.755 (0.090)	0.718 (0.088)	0.786 (0.079)	0.732 (0.064)
	LB	0.609 (0.074)	0.595 (0.073)	0.628 (0.082)	0.608 (0.071)	0.682 (0.076)	0.662 (0.081)	0.713 (0.084)	0.683 (0.082)	0.732 (0.070)	0.697 (0.074)
	XB	0.603 (0.067)	0.598 (0.081)	0.641 (0.094)	0.623 (0.086)	0.706 (0.085)	0.702 (0.097)	0.747 (0.089)	0.750 (0.093)	0.745 (0.088)	0.774 (0.067)

Both Ward and k-means clustering demonstrated comparable performances, although Ward seems to perform slightly better in the higher gene-count classes (Figure 3-23 Top). When it comes to the type of compositional signatures, global composition (GB) was significantly better in all comparisons, an observation also noted by Promponas (2009). Lowering the CAST score threshold from 40 to 15 drastically improved the performance of the XB (masked residues composition) signature; the performance of LB (LCRs composition) was also improved but in to a lesser extent (Figure 3-23 Bottom). Both of these results can be easily explained since lowering the CAST threshold leads to a greater number of detected LCRs and masked residues, thus increasing the signal of the respective signatures. With a CAST threshold of 40, LB was consistently better than XB but also worse than LC (Figure 3-23 Bottom). Merging all signatures into one 60-dimensional vector didn't improve the overall performance for neither of the clustering methods (Table 8).

Based on these results, we decided to focus on a CAST threshold of 40 and the global and LCRs composition signature when trying to build a predictive model for the pathogenicity of *Escherichia* strains. The model is described in detail in the next section.

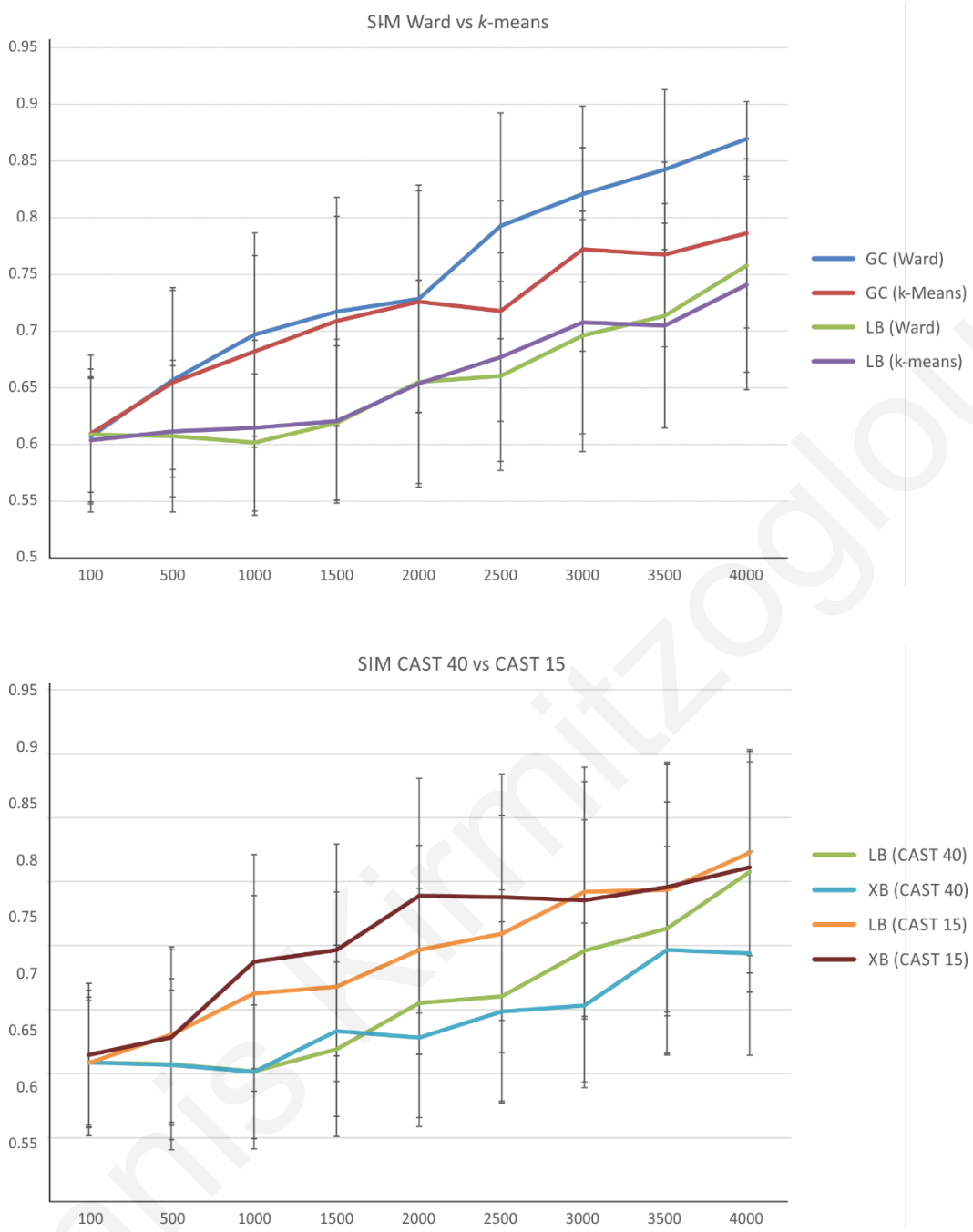


Figure 3-23. *Comparison of the SIM similarity measure for all the gene-count classes.*

Top: *Global (GC) and LCRs (LB) composition signatures when using hierarchical (Ward) or k-means clustering.*

Bottom: *Comparison of the LCRs (LB) and Masked residues (XB) with the Ward hierarchical clustering algorithm.*

3.3.2 Training and validation of binary classification models capable of prediction the pathogenicity of *Escherichia* strains based on global and local compositional signatures

Having confirmed the predictive power of compositional signatures derived from "proteomes" of *Escherichia* strains, originally reported by Promponas (2009), we set out to build a predictive model capable of performing the same task with an even higher accuracy. Furthermore, to test whether the model can accurately predict pathogenicity using only fractions of a genome's sequence (as is commonly the case when sequencing environmental samples) we validated its performance against randomly generated sub-samples of the original genomes.

The procedure followed to train and validate the model was described in detail in the section 2.3.6. At the end we built two different predictive models, one utilizing the Global composition signature (GC) and one utilizing the LCRs composition (LC). Both were build using the k-NN algorithm with the value of k (k=1) selected among 4 others (3, 5, 7, 9) after validating the performance of each implementation using leave-one-out cross-validation. Furthermore, we externally validated the performance of the models using as a test set 6 *Escherichia* strains –and their randomly generated sub-samples– (i.e. *E. coli* B str. REL606, *E. coli* O157:H7 str. TW14359, *E. coli* O103:H2 str. 12009, *E. coli* O26:H11 str. 11368, *E. coli* O111:H- str. 11128, *E. coli* O55:H7 str. CB9615) that were not included in the training set (Figure 3-24).

Both of the models have an accuracy of over 90% when using genome sub-samples that are close to the size of the complete genomes. Specifically, the model built based on Global composition (GC) has an accuracy of 84% even when used against sub-samples containing as few as 500 genes, which roughly correspond to about 10% of the average *Escherichia* genome size. For sub-samples with more than 3,000 genes the GC-based model always predicts the correct pathogenicity type. The performance of the LB-based model is not as impressive but it is still capable of correctly classifying such sub-samples 9 out of the 10 times.

Overall, our results showcase that compositional signatures can be used not only to accurately predict the phenotype of species based on their complete genomes [which has

been already demonstrated by others (e.g. Kreil and Ouzounis, 2001; Promponas, 2009)] but also on genome sub-samples, which could lead to the development of useful tools for handling partial metagenomic assemblies. Additionally our method is simple to implement and very fast, in contrast to other approaches that may utilize thousands (or even million) of features making them hard to implement and expensive in terms of computational resources (e.g. Sims and Kim, 2011). Even though some individual compositional features seem to have some correlation with the pathogenicity status (Figure 3-25) of *Escherichia* strains, further research is necessary in order to identify any possible biological relevance.

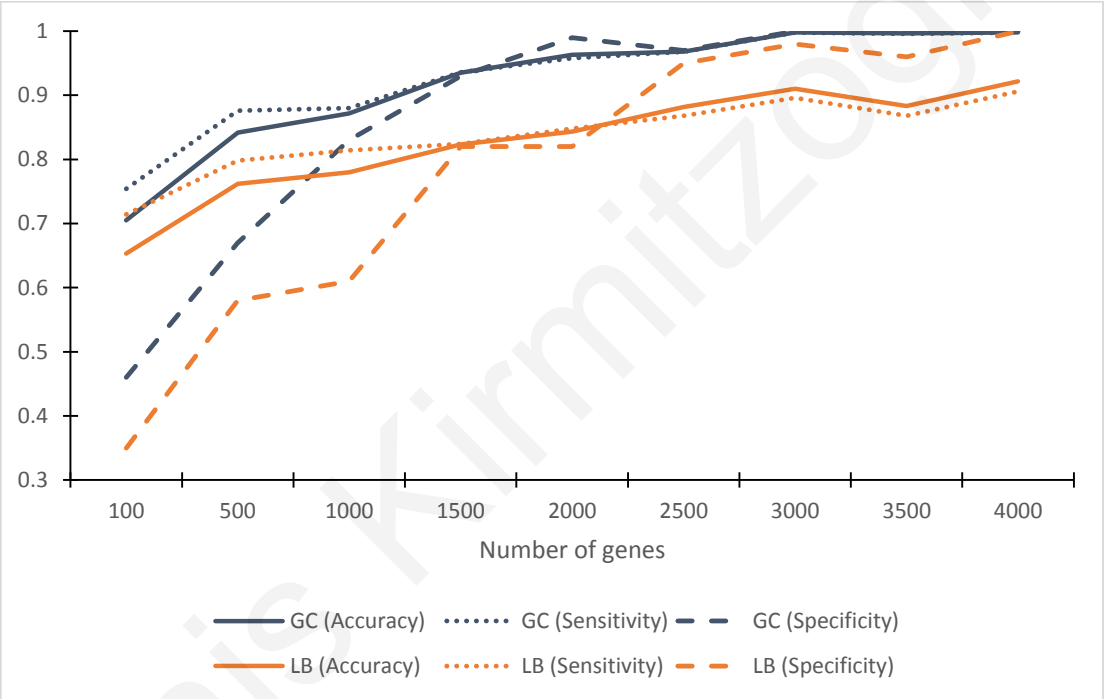


Figure 3-24. *Performance metrics for the final predictive models.*

GC: Global composition signature; LB: LCRs composition signature. For the definitions of the metrics refer to section 2.3.6.

Finally, we used the GC-based model to predict the pathogenicity of the 3 unknown strains in our dataset. The predictions are provided in Table 9.

Table 9. *Predicted pathogenicity for the 3 strains of our dataset that we were unable to verify their pathogenicity using the literature.*

The predictive model used was bases on the Global composition signature.

Species	Genbank Accession	Prediction
E. coli ATCC 8739	NC_010468	Non-pathogenic
E. coli SMS-3-5	NC_010498	Pathogenic
E. fergusonii ATCC 35469	NC_011740	Pathogenic

It is worth mentioning here that the environmental SMS-3-5 strain has been reported to be resistant to different antibiotics (Fricke *et al.*, 2008), an observation that may suggest that its origin could potentially be an infected individual.

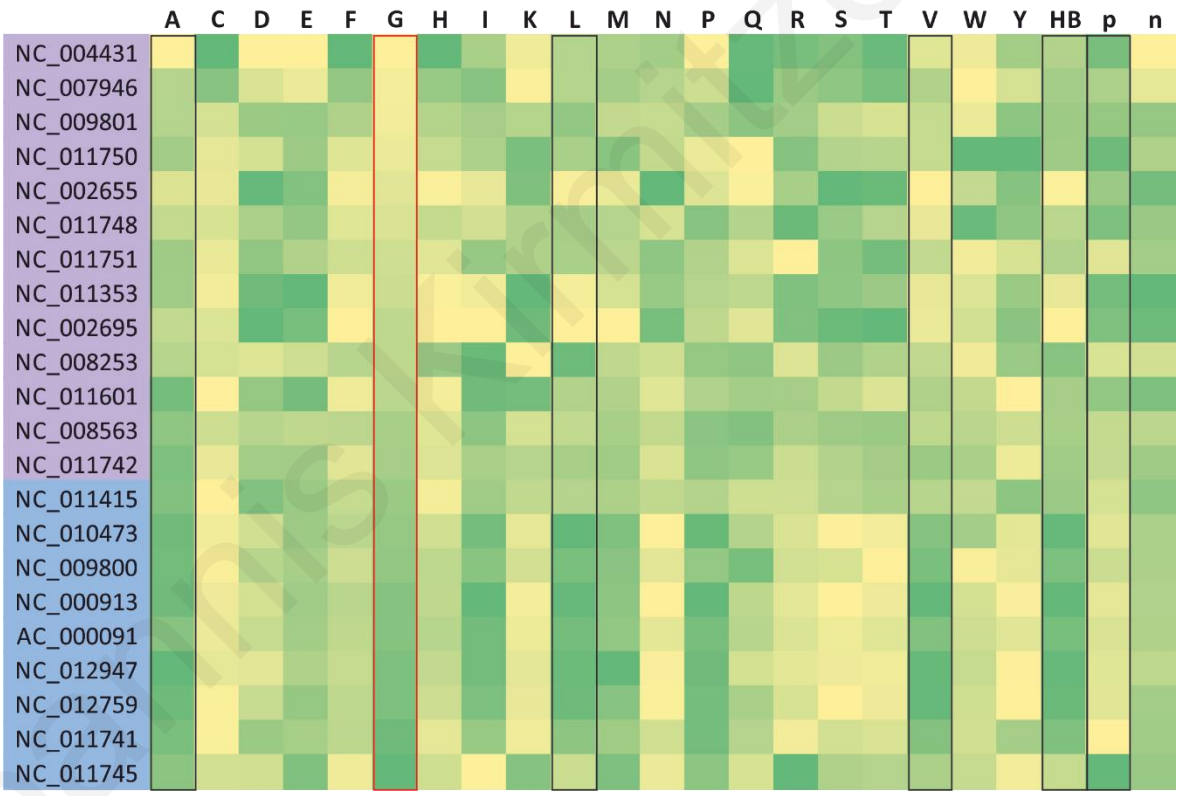


Figure 3-25. *Heatmap view of the Global composition signature (GC).*

Colour coded pathogenicity status: Pathogenic – Purple; Non-pathogenic – Blue. The colour scale maps values from yellow (smallest) to green (largest).

Columns correspond to the 20 standard amino acids; additional columns: ***HB*** *– Hydrophobic residues (A, F, I, L, M, V, W, Y);* ***p*** *– Positively charged residues (H, R, K);* ***n*** *– Negatively charged residues (D, E). We notice that Glycine composition correlates with pathogenicity status, with pathogenic Escherichia strains clearly depleted in Glycine (highlighted in red). Analogous correlations exist for Alanine, Leucine, Valine, Hydrophobic and Positively charged residues (highlighted in black).*

3.3.3 Exploration for the identification of proteomic subsets responsible for the predictive power of the final models

In an effort to identify subsets of specific proteins responsible for the predictive power of the final model we first clustered all protein sequences belonging to the 22 genomes of the training set using `blastclust`[47] from the NCBI-BLAST package (Altschul *et al.*, 1990) with default settings. Then, we classified every protein sequence into one of four different classes based on the properties of their parent clusters[48]: (i) Core-proteins, (ii) P-only proteins, (iii) NP-only proteins and (iv) Mixed-proteins. The results of the classification process are summarized in Table 10.

Keeping only the respective protein sequences from each strain we generated 3 different versions for each genome having only (i) Core-proteins (ii) P- or NP-only proteins (depending on the pathogenicity of the strain) and (iii) Mixed-proteins. Next, we trained and validated predictive models for each class of genomes, using the same procedure described earlier.

As it is obvious from Figure 3-26 all six predictive models (3 based on GC and 3 on LB signatures) had no predictive value, as they were over-trained. Especially the GC-based ones predicted every sub-sample of the test set as pathogenic. The LB-based models were less biased in their predictions but still wrong most of the times.

[47] ftp://ftp.ncbi.nih.gov/blast/documents/blastclust.html.

[48] For more details refer to section 2.3.7 of "Data & Methods".

Table 10. *Sequence classes for every strain of the training set.*
Classification of the sequences was based on the properties of their belonging clusters generated using blustclust. The average number of sequences for each class per pathogenicity type are also given.

Species	GenBank Accession	P	Counts of proteins			
			Total	Core	NP/P	Mixed
E. coli str. K-12 substr. W3110	AC_000091	NP	4226	2176	249	1801
E. coli str. K-12 substr. MG1655	NC_000913	NP	4144	2176	190	1778
E. coli HS	NC_009800	NP	4378	2176	315	1887
E. coli str. K-12 substr. DH10B	NC_010473	NP	4126	2228	235	1663
E. coli SE11	NC_011415	NP	4679	2176	371	2132
E. coli IAI1	NC_011741	NP	4351	2176	208	1967
E. coli ED1a	NC_011745	NP	4915	2176	616	2123
E. coli BW2952	NC_012759	NP	4084	2177	173	1734
E. coli BL21(DE3)	NC_012947	NP	4228	2176	186	1866
E. coli O157:H7 EDL933	NC_002655	P	5298	2176	1345	1777
E. coli O157:H7 str. Sakai	NC_002695	P	5229	2176	1208	1845
E. coli CFT073	NC_004431	P	5338	2176	1316	1846
E. coli UTI89	NC_007946	P	5021	2176	1001	1844
E. coli 536	NC_008253	P	4619	2176	796	1647
E. coli APEC O1	NC_008563	P	4428	2176	753	1499
E. coli E24377A	NC_009801	P	4749	2176	546	2027
E. coli O157:H7 str. EC4115	NC_011353	P	5315	2176	1323	1816
E. coli O127:H6 str. E2348/69	NC_011601	P	4552	2176	719	1657
E. coli S88	NC_011742	P	4692	2176	535	1981
E. coli 55989	NC_011748	P	4759	2176	478	2105
E. coli IAI39	NC_011750	P	4730	2176	688	1866
E. coli UMN026	NC_011751	P	4825	2176	753	1896
Average NP			4347.9	2181.9	282.6	1883.4
Average P			4888.8	2176.0	881.6	1831.2
Average All			4667.5	2178.4	636.5	1852.6

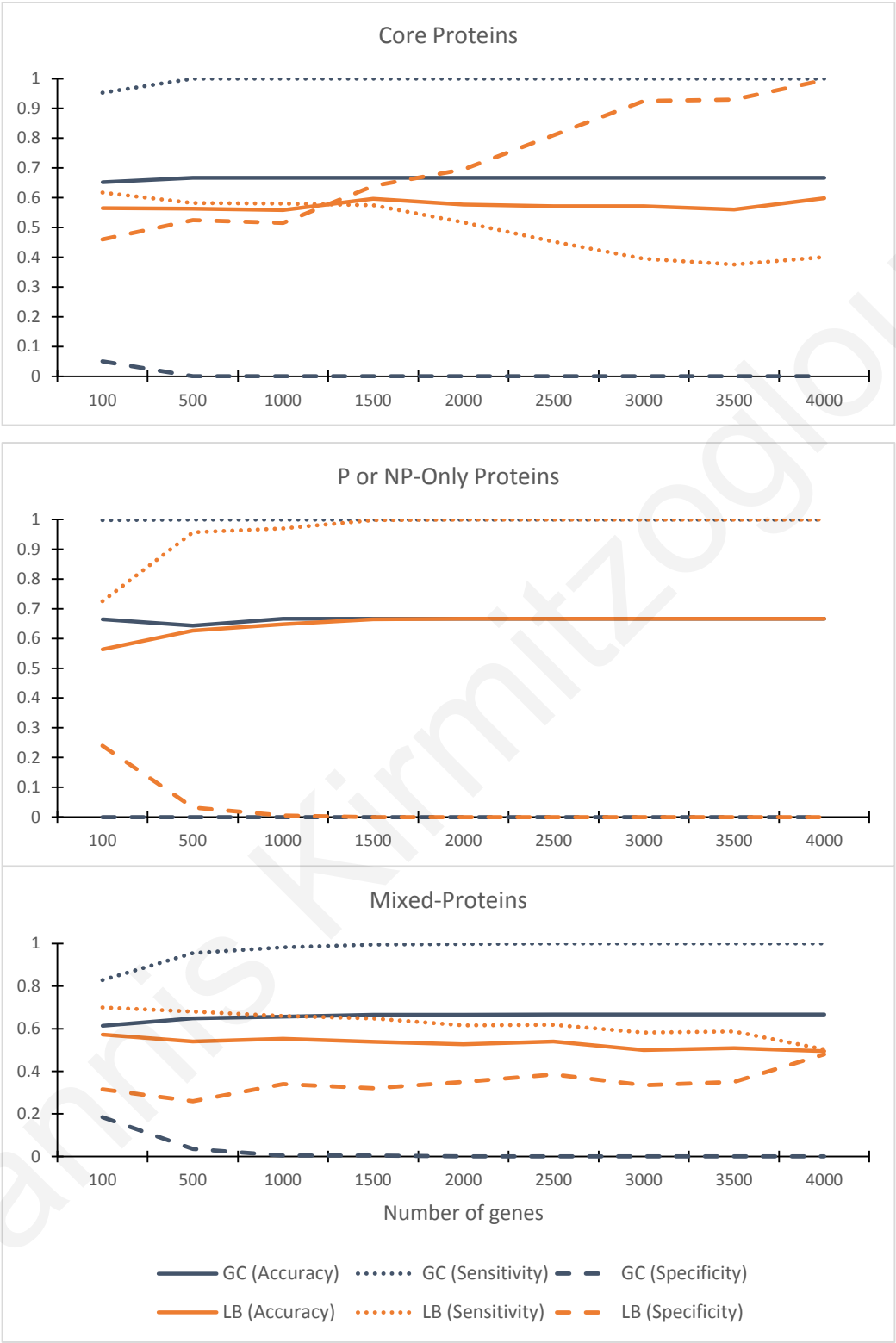


Figure 3-26. *Performance metrics of the predictive models built from genomes containing only specific classes of sequences.*
Core-proteins are sequences with homologs in every Escherichia strain; P- and NP-only have homologs only in pathogenic and non-pathogenic strains respectively; Mixed-proteins don't belong to any of the above classes. GC: Global composition signature; LB: LCRs composition signature. For the definitions of the metrics refer to section 2.3.6.

In a different strategy we generated chimeric genomes having (i) P-only, (ii) NP-only and (iii) P & NP-only genes in equal numbers. We also generated a fourth category of chimeras containing proteins randomly sampled from all 22 genomes. Using the full GC and LB-based models we presented in the previous section, we predicted the pathogenicity of each bootstrap/chimeric genome. Results were grouped together per chimeric class and subclass and are presented in Figure 3-27.

Our results indicate that P and NP-only sequences contain enough information to characterize a chimeric genome as non-pathogenic or pathogenic respectively, based solely on compositional measures (Figure 3-27 Bottom). The prediction rate for these chimera classes, with a gene count close to the average gene count of *Escherichia* strains, is similar to the prediction rate of the predictive models when validated against our test set (Figure 3-24). In addition, it seems that the compositional signal contained by the P-only class of sequences is considerably stronger compared to the NP-only sequences; prediction rates for the chimeras containing P-only sequences are consistently higher than 70% even for "genomes" containing as few as 100 genes. On the contrary, both predictive models struggle to accurately predict the NP-only chimeric "genomes" containing less than 2000 genes.

As it was the case with the validation against complete genomes, the GC-based model performs better than the LB-based one, especially in the case of NP-only chimeras. Still, when used against the P-only chimeras, the performance of the LB-based model is much closer to the GC-based one, indicating that perhaps P-only genes carry a stronger local composition signature. This finding suggests that local compositional bias is worth further investigating for more subtle associations to pathogenicity.

Prediction rates against the chimeras containing NP- and P-only sequences in equal amount, as well as against chimeras with sequences randomly sampled from all genomes, are remarkably similar. As expected, predictions rates are close to 50% although the GC-based model shows a preference in predicting chimeric "genomes" as pathogenic. This may be due to the seemingly stronger compositional signal of P-only genes but can also be attributed to the unbalanced composition of the training set which contained only 9 non-pathogenic compared to 13-pathogenic *Escherichia* strains.

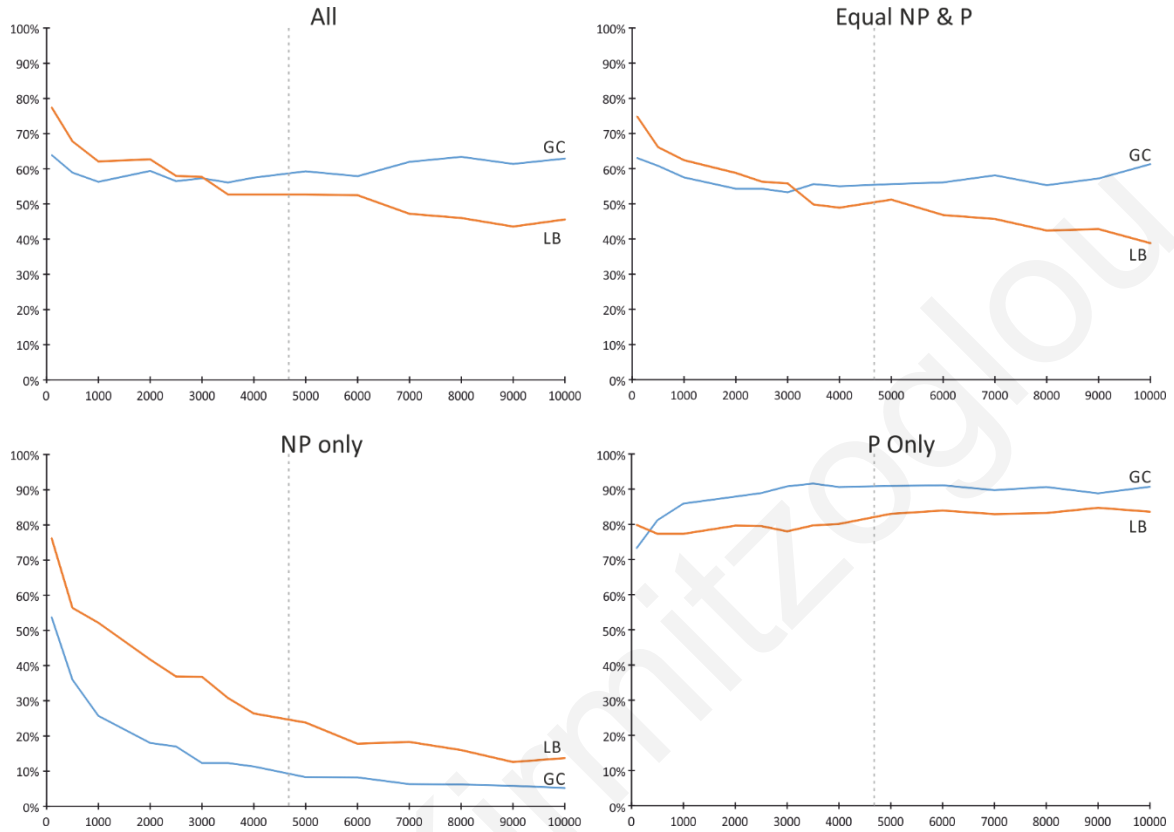


Figure 3-27. *Percentage of chimeric "genomes" predicted as pathogenic plotted against the "genome" size.*
All: chimeras with sequences randomly sampled from the pool of all sequences contained in every genome; P- and NP-only have sequences randomly sampled from genes that have homologs only in pathogenic and non-pathogenic strains respectively; Equal NP & P contain sequences from both classes in equal amounts.
GC: Global composition signature; LB: LCRs composition signature. For the definitions of the metrics refer to section 2.3.6. The dotted grey line represents mean genome size.

3.4 A new generation of the CAST algorithm

While CAST (version 1.0) was shown to be a superior choice for masking low complexity regions compared to SEG (Kreil and Ouzounis, 2003; also this study; Promponas *et al.*, 2000) it is also considerably slower (Figure 3-28; Figure 3-29; Table 12). Furthermore, it lacks certain features that made it harder to use, both manually and as part of a pipeline. During this work we developed a new version of CAST with added features and significant speed improvements. We also mention the results of the collaboration with researchers from the Research Center for Intelligent Systems & Networks (KIOS) of the University of Cyprus to port the optimized CAST versions to new hardware platforms.

In summary the latest version of CAST (v2.1) incorporates the following changes, which were added incrementally during the past 5 years:

i. General speed optimizations to the implementation of the algorithm and bug fixes.
ii. We added an option (`-skip X` parameter switch) to stop calculating the alignment scores for amino acid residue types that failed to produce a score higher than the threshold after X iterations. X can be any non-negative integer; the default value (0) maintains complete backwards compatibility with the results of the original CAST implementation.
iii. We added an option (`-tab` parameter switch) to save the statistics information to a tab delimited file. The original implementation saved statistics information (using the `-stat` switch) in a text format that was more suitable for humans. As a result, a post processing step was often necessary to load the output file in software tools (such as spreadsheet editors) that natively support the tab delimited format. We also kept the `-stat` option to ensure backwards compatibility.
iv. Added an option (`-soft` parameter switch) to apply soft- instead of hard-masking on the output sequence. When the original implementation of CAST was published (Promponas *et al.*, 2000) soft-masking was not supported by BLAST but it was added in later versions (and is now the default option for `segmasker`, the C++ implementation of seg distributed through the NCBI-BLAST suite). CAST v2.1 can now produce such files, which also makes it directly compatible with the latest

versions of the BLAST database creation tool that is able to store soft-masks as part of the BLAST database.

The optimizations incorporated to the latest version of CAST provide a significant boost to its performance, independently of the LCR content of the data-set to which CAST is applied (Table 12). Specifically, by benchmarking CAST against the complete protein sets of 7 organisms with varying levels of LCR-content (percentage of masked residues ranging from 1.37% for *Chlamydophila pneumoniae AR39* to 14.42% for *Plasmodium falciparum 3D7*) we observe a consistent speed-up of 1.8-2.0x (Figure 3-28).

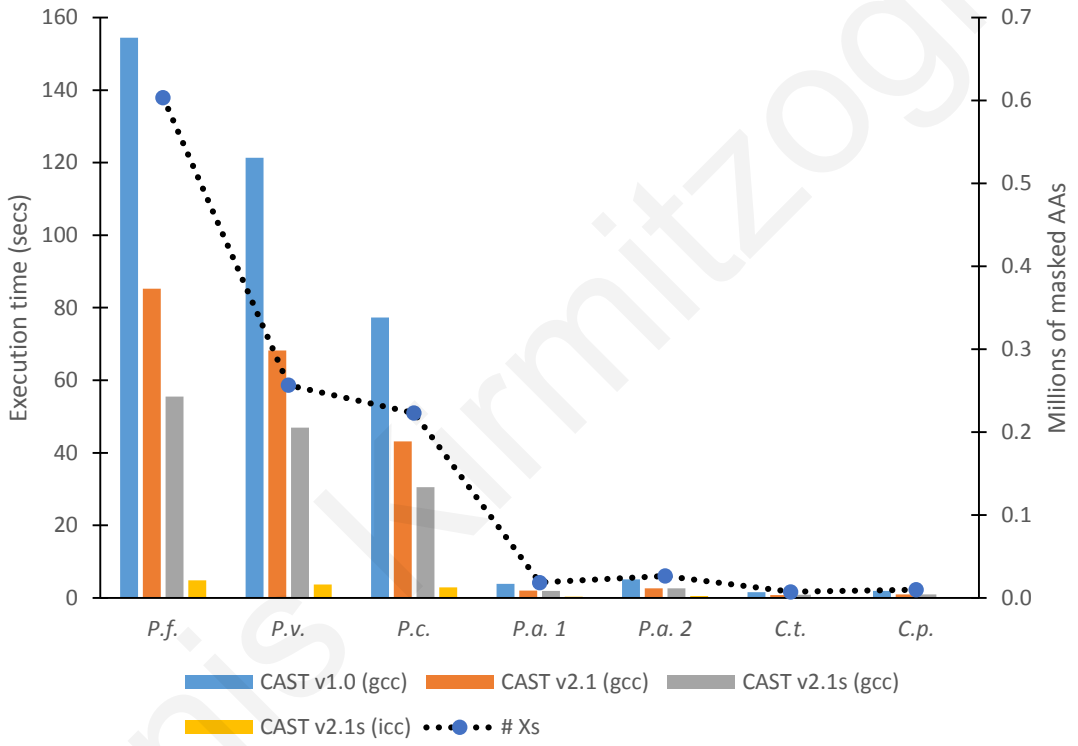


Figure 3-28. *Computational performance of CAST running against 7 genomes with varying levels of LCR content.*
Execution times for all versions of CAST heavily rely on the LCR content (**# Xs**) of the data-set. ***P.f.:*** *Plasmodium falciparum 3D7;* ***P.v.:*** *Plasmodium vivax SaI1;* ***P.c.:*** *Plasmodium cynomolgi B;* ***P.a. 1:*** *Pseudomonas aeruginosa PAO1;* ***P.a. 2:*** *Pseudomonas aeruginosa PA14;* ***C.t.:*** *Chlamydia trachomatis D-EC;* ***C.p.:*** *Chlamydophila pneumoniae AR39;* ***gcc:*** *CAST binaries compiled using the GNU Compiler Collection;* ***icc:*** *CAST binaries compiled using the Intel C++ compiler;* ***CAST v2.1s****: CAST v2.1 with the* `-skip 5` *option.*

By enabling the `-skip` parameter we observe speed-ups that are much more LCR content-dependent. This is to be expected since this parameter instructs CAST to stop checking for LCRs of residue types that failed to produce any significant scores after a user-specified amount of iterations. Protein sequences rich in LCRs, usually require many itera-

tions of the Smith-Waterman (Smith and Waterman, 1981) algorithm implementation in CAST before every LCR is detected. Instructing CAST to ignore specific residue types after a number of iterations can improve performance by a wide margin for data-sets rich in LCRs (e.g. execution times are reduced by about 35% in the case of *P. falciparum*; Figure 3-29). On the other hand, the speed-up for data-sets poor in LCRs is marginal.

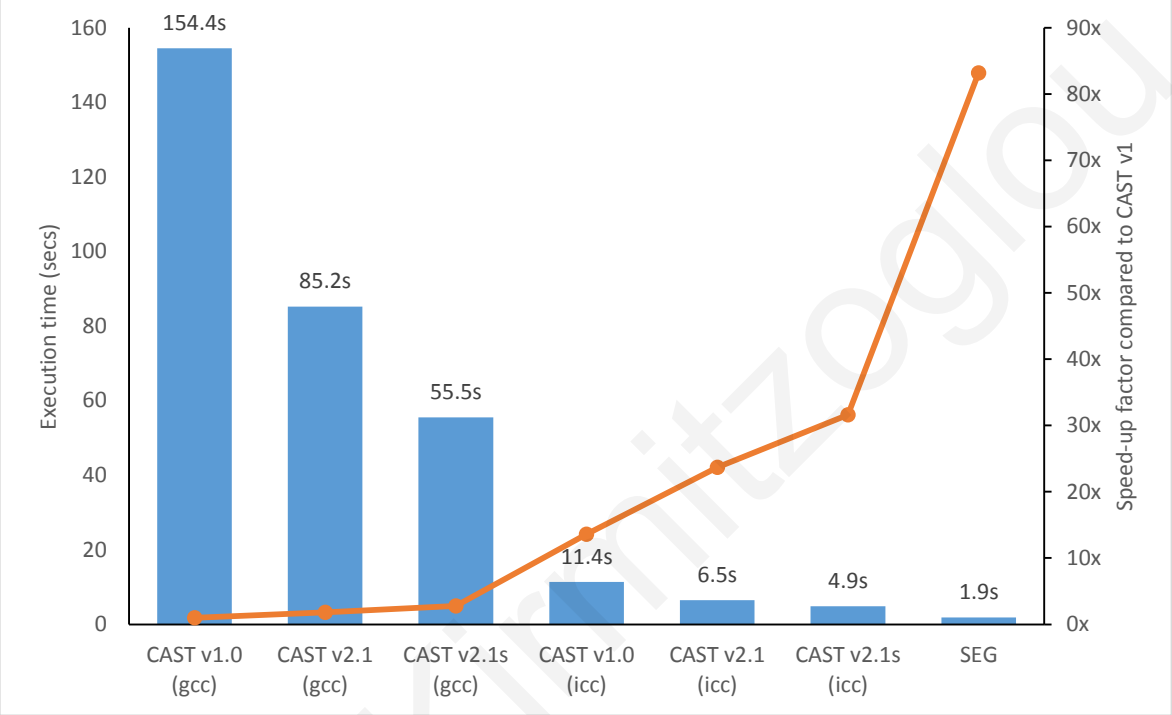


Figure 3-29. *Computational performance of CAST running against P. falciparum* and the observed speed-up factor compared to CAST v.1.0 (gcc).
gcc: *CAST binaries compiled using the GNU Compiler Collection;* ***icc:*** *CAST binaries compiled using the Intel C++ compiler.*

One side-effect of using the `-skip` parameter is that the produced output files may differ compared to those created by CAST v1.0. This usually happens when there are overlapping LCRs of several residue types in a protein sequence. The amount of differences heavily depends on the value of the skip parameter and the composition of the dataset. By empirical means we discovered that a value of 5 offers a decent speed-up for data-sets rich in LCRs with minimal alterations to the output files. For example, in the case of the *P. falciparum* genome only 6 sequences are affected; for comparison, the number of affected proteins for the whole UniProt/SwissProt database (release 2014_05) is 7. On the other hand, as an added benefit of this new option, a researcher can experiment with the score and skip thresholds to gain tighter control to the way CAST detects and masks LCRs.

Table 11. *P. falciparum proteins affected using the `-skip 5` option of CAST v2.1. Out of a total of 13,143 LCRs detected by the original implementation of CAST, only 6 are not detected using this option. By definition the use of the `-skip` option will never affect the number of proteins detected having at least one LCR.*

Protein ID	AA type	From	To	CAST score
PF3D7_0405400	M	118	241	46
PF3D7_1472200	C	1621	1657	50
PF3D7_0704600	Y	3044	3118	69
PF3D7_1410300	Y	706	765	45
PF3D7_1230000	Y	1672	1736	45
PF3D7_0827600	Y	777	816	61

Despite our efforts to optimize CAST, the biggest improvements occurred serendipitously when we compiled the source code using the proprietary Intel C++ compiler[49] (`icc`). Apparently, the Intel compiler was able to apply low-level optimizations during the translation of the source to machine code that the GNU Compiler Collection[50] (`gcc`) (the open source alternative used when compiling the vast majority of software meant for Linux) did not. It is possible that further optimization could be achieved by a skilled programmer simply by applying similar optimizations using `gcc`, e.g. SSE optimizations written directly into the source code. For the record, using the default settings, `icc` produces binaries of CAST that are consistently 5 to 14 times faster than the `gcc` alternatives, especially when used against data-sets rich in LCRs (Figure 3-28).

Other enhancements with regards to CAST performance have been performed in collaborative work that employed techniques for (i) multi-threaded parallelization, (ii) general purpose Graphical Processing Units (gp-GPUs), and (iii) systolic hardware implementation of the CAST version 2 algorithm on Field Programmable Gate Arrays (FPGAs) (Chrysos *et al.*, 2014; Papadopoulos *et al.*, 2012). These approaches demonstrated similar improvements with the FPGAs reaching up to 1000x speed-up. However, hardware based acceleration still faces some limitations with regards to input/output operations and data transfer to/from the special hardware.

[49] https://software.intel.com/en-us/c-compilers
[50] http://gcc.gnu.org/

Table 12. *Computational performance of CAST and SEG running against 7 genomes with varying levels of LCR content.*
We report running times (in seconds) of different CAST binaries. ***gcc:*** *CAST binaries compiled using the GNU Compiler Collection;* ***icc:*** *CAST binaries compiled using the Intel C++ compiler;* ***# SEQ:*** *number of sequences;* ***# AAs:*** *number of (thousands) of amino acid residues;* ***% SEQ:*** *percentage of sequences containing at least one LCR;* ***% AAs:*** *percentage of amino acid residues masked by CAST v2.1 using default settings.*

		compiled with gcc			compiled with icc			SEG					
Species		v1.0	v2.1	v2.1s	v1.0i	v2.1i	v2.1is	vs 1.0	vs 2.1is	# SEQ	# AAs (k)	% SEQ	% AAs
	avg	154.44	85.21	55.47	11.35	6.50	4.89	1.86					
P. falciparum	sd	0.48	0.15	0.04	0.01	0.01	0.01	0.01		5,538	4,186	72.84%	14.42%
	accl	1.0x	1.8x	2.8x	13.6x	23.7x	31.6x	83.2x	2.6x				
	avg	121.36	68.22	46.95	8.19	4.77	3.74	1.08					
P. vivax	sd	1.67	0.05	0.04	0.01	0.01	0.00	0.01		5,393	3,749	59.39%	6.85%
	accl	1.0x	1.8x	2.6x	14.8x	25.5x	32.4x	112.3x	3.5x				
	avg	77.32	43.17	30.53	6.37	3.64	2.94	0.91					
P. cynomolgi	sd	0.33	0.04	0.06	0.03	0.01	0.01	0.01		5,716	3,285	52.26%	6.79%
	accl	1.0x	1.8x	2.5x	12.1x	21.2x	26.3x	84.9x	3.2x				
	avg	3.86	1.98	1.97	0.72	0.37	0.38	0.28					
P. aeruginosa PAO1	sd	0.01	0.01	0.01	0.00	0.00	0.01	0.01		4,139	1,373	13.19%	1.35%
	accl	1.0x	1.9x	2.0x	5.4x	10.4x	10.3x	13.8x	1.7x				
	avg	5.15	2.64	2.68	1.02	0.52	0.53	0.40					
P. aeruginosa PA14	sd	0.01	0.00	0.02	0.01	0.01	0.00	0.01		5,892	1,942	13.42%	1.37%
	accl	1.0x	2.0x	1.9x	5.1x	9.8x	9.7x	13.0x	1.3x				
	avg	1.61	0.83	0.84	0.33	0.17	0.17	0.10					
C. trachomatis	sd	0.00	0.00	0.02	0.00	0.00	0.00	0.01		1,758	626	12.51%	1.19%
	accl	1.0x	1.9x	1.9x	4.8x	9.5x	9.5x	15.8x	1.7x				
	avg	1.91	0.98	0.99	0.41	0.21	0.21	0.12					
C. pneumoniae	sd	0.02	0.01	0.01	0.01	0.01	0.00	0.00		2,283	748	14.63%	1.33%
	accl	1.0x	2.0x	1.9x	4.7x	9.2x	9.0x	15.9x	1.8x				

4 Conclusions

4.1 Effects of LCRs and LCRs detection tools in protein database search

Overall, our results clearly indicate that low complexity regions can and will cause issues in all methods relying heavily on the calculation of sequence similarity. This is especially true when the sequence similarity software used does not employ built-in ways to handle LCRs. At the same time even BLAST, the first sequence database search tool that incorporated LCR-handling mechanisms, struggles against datasets rich in LCRs. In this work we tested and verified the effectiveness of all LCR-handling methods offered by BLAST with the addition of database masking with CAST or SEG; an approach that has never been discussed before in such detail.

Our results clearly indicate for the first time that filtering both the query sequence and the database may improve the accuracy of retrieving the true best-hit, which is crucial –among others– in homolog detection using the best bidirectional hit approach in computational comparative genomics. Utilizing extra features of the BLASTP output (bit score, percent identities and alignment length) is shown to enhance the method's performance (Figure 3-7). Additionally, as a side effect of two-way masking, we observed a significant speedup of BLASTP (in the range of 1.3-40x), especially in cases of heavily compositionally biased datasets. This is mainly attributed to the notable reduction of the number of HSPs identified and, consequently, input/output operations, along with a reduction of file sizes, depending on the proportion of compositionally biased proteins and the frequency of incidence of low complexity regions.

Furthermore, two-way masking with CAST was consistently the method returning the most biologically relevant results. Composition-based statistics –the default LCR-handling method employed by BLAST– while clearly superior to SEG, struggled to keep-up with all the masking schemes that employed CAST and in some cases was even worse than bare BLASTP (i.e. with no LCR-handling employed). This observation is in agreement with Forslund & Sonnhammer (2009), the only other study we are aware of comparing CBS with any other method besides SEG-masking.

We propose that two-way CAST masking should be adopted in large-scale computational comparative genomics studies for (i) reducing necessary computational resources (CPU time, storage space) and (ii) increasing the sensitivity of homolog detection. To facilitate the adoption of CAST, we also developed a new version with significant speed-up improvements and pipeline-friendly features. We plan to release the source code of CAST version 2.1 along with optimized binaries to the public in the near future.

Another issue worth raising is the suitability of ASTRAL (and its subsets derived by removing redundancy) as a data-set for evaluating the performance of LCR-handling tools, mainly because of its deficit in LCRs compared to the databases routinely searched by biologists. Forslund & Sonnhammer have expressed similar concerns [concurrently to our work (Kirmitzoglou and Promponas, 2009)] and described a computational pipeline to produce PFAM-based datasets capable of exposing the weaknesses of the aforementioned methods (Forslund and Sonnhammer, 2009). Unfortunately, their method to produce such datasets requires many days of computational time on a typical PC and –contrary to ASTRAL– there is currently no way to download precompiled sets. Until a better data-set is described researchers should be very cautious when trying to evaluate LCR-handling tools with ASTRAL.

Future work along this line of research may include (i) benchmarking other sequence search algorithms [e.g. PSI-BLAST (Altschul *et al.*, 1997), HMMER (Mistry *et al.*, 2013)], and (ii) a more detailed investigation of how the different LCR-handling approaches perform on specific protein families (e.g. transmembrane proteins).

4.2 Tools for searching and visualizing low complexity regions in protein sequences

In this work, we developed LCR-eXXXplorer, a novel web-based system with unique properties and features for researchers interested in LCRs. Given the multifaceted importance of low complexity regions reported in the literature (Benita *et al.*, 2006; Dunker and Obradovic, 2001; Dunker *et al.*, 2001, 2002; Echols *et al.*, 2002; Fogel *et al.*, 2005; Lovell, 2003; Miskinyte *et al.*, 2013; e.g. Mitas, 1997; Müller *et al.*, 2002; Nandi *et al.*, 2003b; Pizzi and Frontali, 2001; Radivojac *et al.*, 2006; Romov *et al.*, 2006; Stern *et al.*, 2001; Tress *et*

al., 2006) the lack of such a service impedes efforts for the more detailed elucidation of the biological relevance of LCRs in terms of function, structure and evolution.

LPS-annotate (Harbi *et al.*, 2011), the only service comparable to LCR-eXXXplorer, has several drawbacks since it mainly provides a search functionality. Furthermore, it only contains LCRs detected by a single algorithm (LPS, which is not widely used by the community) and its database has not been updated since 2009[51]. Commonly used services with strong visual capabilities, such as UniProtKB (The UniProt Consortium, 2013), contain annotations of LCRs-related features but they are oftentimes incomplete, hard to search and inconsistent. As of May 1st 2014, less than 8% of proteins in UniProt/SwissProt were annotated with at least one LCR type, when the respective figure for the same data-set in LCR-eXXXplorer is close to 60%.

The LCR-eXXXplorer system is designed in a modular fashion which enables future addition of LCRs detected by other algorithms (e.g. BIAS, LPS) or LCRs detected with different software options. Additions of other datasets (e.g. UniProt/TrEMBL, or complete genomes) is trivial since it requires configuration of a separate database under the GBrowse hierarchy; computational load with this amount of data can be handled using the inherent capability of GBrowse (also LCR-eXXXplorer) to work with distributed databases. However, the major priority for future development is to automate the database update procedure.

4.3 Compositional properties of protein sequences in complete genomes: is there a signal out there?

Previous studies demonstrated that a global view of amino acid compositional properties of proteins encoded within complete genomes contain a signal related to phenotypic traits of the respective species, e.g. reflecting the niche in which an organism thrives (Campbell *et al.*, 1999; e.g. Karlin *et al.*, 1997; Kreil and Ouzounis, 2001). In all these works, only global amino acid composition was taken into account.

[51] Information obtained from http://cedra.biol.mcgill.ca/LPS/help_lps1.pdf, accessed June 2014.

In this work we went one step further and, using as a target trait the pathogenicity of strains/species belonging to the *Escherichia* genus, we examined whether local compositional properties contain comparable signals using the same holistic (i.e., genome-wide) approach.

Our results show that while global compositional properties seem to contain more information that could be suitable for successful pathogenicity prediction compared to local ones, it is worth mentioning that the information content of the latter (as encoded using different measures of local compositional bias content) is not negligible.

Moreover, an important (and clearly innovative) contribution of the work presented in this thesis, rests on the systematic simulations performed on artificially generated "genomes" encoding proteins of desired origin and/or "pathogenicity profile" (i.e., distribution with regards to pathogenic/non-pathogenic strains). With this experimental setup we were able to demonstrate that

- it is in principle feasible to use relatively incomplete genome data (e.g. incomplete metagenome supercontigs with a few hundred genes) to assess with good precision whether such data originate from the genome of a pathogenic species/strain, and
- it is the overall protein coding complement of the genome that contains the most useful signal and not only a subclass of "virulence genes".

Regarding the former issue, it is worth mentioning that this finding shows a potential for useful applications. More specifically, given the fact that next generation (meta)-genome sequencing is now becoming a commodity and increasingly accessible to researchers, it can be envisioned that it may be routinely used for clinical or environmental monitoring applications within the near future. In environmental samples (note: environment may refer to man-made environments or even the human body itself) where meta-genome sequencing is currently the only way to go, our approach could rapidly identify with high precision cases where pathogenic stains may require to set up an alert.

5 References

Albà, M.M., Laskowski, R.A., Hancock, J.M., 2002. Detecting cryptically simple protein sequences using the SIMPLE algorithm. Bioinformatics 18, 672–678. doi:10.1093/bioinformatics/18.5.672

Altschul, S.F., Boguski, M.S., Gish, W., Wootton, J.C., 1994. Issues in searching molecular sequence databases. Nat. Genet. 6, 119–129. doi:10.1038/ng0294-119

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., 1990. Basic local alignment search tool. J. Mol. Biol. 215, 403–410. doi:10.1016/S0022-2836(05)80360-2

Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J., 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25, 3389–3402.

Altschul, S.F., Wootton, J.C., Gertz, E.M., Agarwala, R., Morgulis, A., Schaffer, A.A., Yu, Y.-K., 2005. Protein Database Searches Using Compositionally Adjusted Substitution Matrices. FEBS J. 272, 5101–5109. doi:10.1111/j.1742-4658.2005.04945.x

Andrade, M.A., Brown, N.P., Leroy, C., Hoersch, S., Daruvar, A. de, Reich, C., Franchini, A., Tamames, J., Valencia, A., Ouzounis, C., Sander, C., 1999. Automated genome sequence analysis and annotation. Bioinformatics 15, 391–412. doi:10.1093/bioinformatics/15.5.391

Andrade, M.A., Ponting, C.P., Gibson, T.J., Bork, P., 2000. Homology-based method for identification of protein repeats using statistical significance estimates. J. Mol. Biol. 298, 521–537. doi:10.1006/jmbi.2000.3684

Andreeva, A., Howorth, D., Chandonia, J.-M., Brenner, S.E., Hubbard, T.J.P., Chothia, C., Murzin, A.G., 2007. Data growth and its impact on the SCOP database: new developments. Nucleic Acids Res. doi:10.1093/nar/gkm993

Antonets, K.S., Nizhnikov, A.A., 2013. SARP: A Novel Algorithm to Assess Compositional Biases in Protein Sequences. Evol. Bioinforma. Online 9, 263–273. doi:10.4137/EBO.S12299

Anurag, M., Singh, G.P., Dash, D., 2011. Location of disorder in coiled coil proteins is influenced by its biological role and subcellular localization: a GO-based study on human proteome. Mol. Biosyst. 8, 346–352. doi:10.1039/C1MB05210A

Bailey, T.L., 2008. Discovering Sequence Motifs, in: Keith, J.M. (Ed.), Bioinformatics, Methods in Molecular Biology™. Humana Press, pp. 231–251.

Bairoch, A., Boeckmann, B., 1992. The SWISS-PROT protein sequence data bank. Nucleic Acids Res. 20, 2019–2022.

Bamber, D., 1975. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. J. Math. Psychol. 12, 387–415. doi:10.1016/0022-2496(75)90001-2

Benita, Y., Wise, M.J., Lok, M.C., Humphery-Smith, I., Oosting, R.S., 2006. Analysis of High Throughput Protein Expression in Escherichia coli. Mol. Cell. Proteomics 5, 1567–1580. doi:10.1074/mcp.M600140-MCP200

Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E., 2000. The Protein Data Bank. Nucleic Acids Res. 28, 235–242. doi:10.1093/nar/28.1.235

Blum, G., Falbo, V., Caprioli, A., Hacker, J., 1995. Gene clusters encoding the cytotoxic necrotizing factor type 1, Prs-fimbriae and alpha-hemolysin form the pathogenicity island II of the uropathogenic Escherichia coli strain J96. FEMS Microbiol. Lett. 126, 189–195.

Bradshaw, C.R., Surendranath, V., Habermann, B., 2006. ProFAT: a web-based tool for the functional annotation of protein sequences. BMC Bioinformatics 7, 466. doi:10.1186/1471-2105-7-466

Brendel, V., Bucher, P., Nourbakhsh, I.R., Blaisdell, B.E., Karlin, S., 1992. Methods and algorithms for statistical analysis of protein sequences. Proc. Natl. Acad. Sci. 89, 2002–2006.

Buchan, D.W.A., Ward, S.M., Lobley, A.E., Nugent, T.C.O., Bryson, K., Jones, D.T., 2010. Protein annotation and modelling servers at University College London. Nucleic Acids Res. 38, W563–W568. doi:10.1093/nar/gkq427

Campbell, A., Mrázek, J., Karlin, S., 1999. Genome signature comparisons among prokaryote, plasmid, and mitochondrial DNA. Proc. Natl. Acad. Sci. 96, 9184–9189. doi:10.1073/pnas.96.16.9184

Carlos, C., Pires, M.M., Stoppe, N.C., Hachich, E.M., Sato, M.I., Gomes, T.A., Amaral, L.A., Ottoboni, L.M., 2010. Escherichia coli phylogenetic group determination and its application in the identification of the major animal source of fecal contamination. BMC Microbiol. 10, 161. doi:10.1186/1471-2180-10-161

Chandonia, J.-M., Hon, G., Walker, N.S., Lo Conte, L., Koehl, P., Levitt, M., Brenner, S.E., 2004. The ASTRAL Compendium in 2004. Nucleic Acids Res. 32, D189–D192. doi:10.1093/nar/gkh034

Chart, H., Smith, H. r., La Ragione, R. m., Woodward, M. j., 2000. An investigation into the pathogenic properties of Escherichia coli strains BLR, BL21, DH5α and EQ1. J. Appl. Microbiol. 89, 1048–1058. doi:10.1046/j.1365-2672.2000.01211.x

Chrysos, G., Sotiriades, E., Rousopoulos, C., Pramataris, K., Papaefstathiou, I., Dollas, A., Papadopoulos, A., Kirmitzoglou, I., Promponas, V.J., Theocharides, T., Petihakis, G., Lagnel, J., 2014. Reconfiguring the Bioinformatics Computational Spectrum: Challenges and Opportunities of FPGA-Based Bioinformatics Acceleration Platforms. IEEE Des. Test 31, 62–73. doi:10.1109/MDAT.2013.2284191

Claverie, J.-M., States, D.J., 1993. Information enhancement methods for large scale sequence analysis. Comput. Chem. 17, 191–201. doi:10.1016/0097-8485(93)85010-A

Cohen, C., Parry, D.A.D., 1990. α-Helical coiled coils and bundles: How to design an α-helical protein. Proteins Struct. Funct. Bioinforma. 7, 1–15. doi:10.1002/prot.340070102

Coletta, A., Pinney, J.W., Solis, D.Y.W., Marsh, J., Pettifer, S.R., Attwood, T.K., 2010. Low-complexity regions within protein sequences have position-dependent roles. BMC Syst. Biol. 4, 43. doi:10.1186/1752-0509-4-43

Coronado, J.E., Attie, O., Epstein, S.L., Qiu, W.-G., Lipke, P.N., 2006. Composition-Modified Matrices Improve Identification of Homologs of Saccharomyces cerevisiae Low-Complexity Glycoproteins. Eukaryot. Cell 5, 628–637. doi:10.1128/EC.5.4.628-637.2006

Daily, K.M., Radivojac, P., Dunker, A.K., 2005. Intrinsic Disorder and Prote in Modifications: Building an SVM Predictor for Methylation, in: Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, 2005. CIBCB '05. Presented at the Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, 2005. CIBCB '05, pp. 1–7. doi:10.1109/CIBCB.2005.1594957

Di Domenico, T., Potenza, E., Walsh, I., Parra, R.G., Giollo, M., Minervini, G., Piovesan, D., Ihsan, A., Ferrari, C., Kajava, A.V., Tosatto, S.C.E., 2014. RepeatsDB: a database of tandem repeat protein structures. Nucleic Acids Res. 42, D352–357. doi:10.1093/nar/gkt1175

Dosztányi, Z., Csizmok, V., Tompa, P., Simon, I., 2005a. IUPred: web server for the prediction of intrinsically unstructured regions of proteins based on estimated energy content. Bioinformatics 21, 3433–3434. doi:10.1093/bioinformatics/bti541

Dosztányi, Z., Csizmók, V., Tompa, P., Simon, I., 2005b. The Pairwise Energy Content Estimated from Amino Acid Composition Discriminates between Folded and Intrinsically Unstructured Proteins. J. Mol. Biol. 347, 827–839. doi:10.1016/j.jmb.2005.01.071

Dosztányi, Z., Mészáros, B., Simon, I., 2009. ANCHOR: web server for predicting protein binding regions in disordered proteins. Bioinformatics 25, 2745–2746. doi:10.1093/bioinformatics/btp518

Dunker, A.K., Lawson, J.D., Brown, C.J., Williams, R.M., Romero, P., Oh, J.S., Oldfield, C.J., Campen, A.M., Ratliff, C.M., Hipps, K.W., Ausio, J., Nissen, M.S., Reeves, R., Kang, C., Kissinger, C.R., Bailey, R.W., Griswold, M.D., Chiu, W., Garner, E.C., Obradovic, Z., 2001. Intrinsically disordered protein. J. Mol. Graph. Model. 19, 26–59. doi:10.1016/S1093-3263(00)00138-8

Dunker, A.K., Obradovic, Z., 2001. The protein trinity—linking function and disorder. Nat. Biotechnol. 19, 805–806. doi:10.1038/nbt0901-805

Dunker, K.A., Brown, C.J., Obradovic, Z., 2002. Identification and functions of usefully disordered proteins, in: Rose, G.D. (Ed.), Advances in Protein Chemistry, Unfolded Proteins. Academic Press, pp. 25–49.

Echols, N., Harrison, P., Balasubramanian, S., Luscombe, N.M., Bertone, P., Zhang, Z., Gerstein, M., 2002. Comprehensive analysis of amino acid and nucleotide composition in eukaryotic genomes, comparing genes and pseudogenes. Nucleic Acids Res. 30, 2515–2523. doi:10.1093/nar/30.11.2515

Eddy, S.R., 2004. Where did the BLOSUM62 alignment score matrix come from? Nat. Biotechnol. 22, 1035–1036. doi:10.1038/nbt0804-1035

Ferenci, T., Zhou, Z., Betteridge, T., Ren, Y., Liu, Y., Feng, L., Reeves, P.R., Wang, L., 2009. Genomic Sequencing Reveals Regulatory Mutations and Recombinational Events in the Widely Used MC4100 Lineage of Escherichia coli K-12. J. Bacteriol. 191, 4025–4029. doi:10.1128/JB.00118-09

Fielding, R.T., Kaiser, G., 1997. The Apache HTTP Server Project. IEEE Internet Comput. 1, 88–90. doi:10.1109/4236.612229

Finn, R.D., Mistry, J., Tate, J., Coggill, P., Heger, A., Pollington, J.E., Gavin, O.L., Gunasekaran, P., Ceric, G., Forslund, K., Holm, L., Sonnhammer, E.L.L., Eddy, S.R., Bateman, A., 2010. The Pfam protein families database. Nucleic Acids Res. 38, D211–D222. doi:10.1093/nar/gkp985

Fogel, G.B., Weekes, D.G., Varga, G., Dow, E.R., Craven, A.M., Harlow, H.B., Su, E.W., Onyia, J.E., Su, C., 2005. A statistical analysis of the TRANSFAC database. Biosystems 81, 137–154. doi:10.1016/j.biosystems.2005.03.003

Forslund, K., Sonnhammer, E.L., 2009. Benchmarking homology detection procedures with low complexity filters. Bioinformatics 25, 2500–2505. doi:10.1093/bioinformatics/btp446

Fricke, W.F., Wright, M.S., Lindell, A.H., Harkins, D.M., Baker-Austin, C., Ravel, J., Stepanauskas, R., 2008. Insights into the Environmental Resistance Gene Pool from the Genome Sequence of the Multidrug-Resistant Environmental Isolate Escherichia coli SMS-3-5. J. Bacteriol. 190, 6779–6794. doi:10.1128/JB.00661-08

Golding, G.B., 1999. Simple sequence is abundant in eukaryotic proteins. Protein Sci. Publ. Protein Soc. 8, 1358–1361.

Grantham, R., Gautier, C., Gouy, M., Mercier, R., Pave, A., 1980. Codon catalog usage and the genome hypothesis. Nucleic Acids Res. 8, r49–r62.

Green, R.E., Brenner, S.E., 2002. Bootstrapping and normalization for enhanced evaluations of pairwise sequence comparison. Proc. IEEE 90, 1834–1847. doi:10.1109/JPROC.2002.805303

Gribskov, M., Robinson, N.L., 1996. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. Comput. Chem. 20, 25–33.

Harbi, D., Kumar, M., Harrison, P.M., 2011. LPS-annotate: complete annotation of compositionally biased regions in the protein knowledgebase. Database J. Biol. Databases Curation 2011. doi:10.1093/database/baq031

Harrison, P.M., 2006. Exhaustive assignment of compositional bias reveals universally prevalent biased regions: analysis of functional associations in human and Drosophila. BMC Bioinformatics 7, 441. doi:10.1186/1471-2105-7-441

Harrison, P.M., Gerstein, M., 2003. A method to assess compositional bias in biological sequences and its application to prion-like glutamine/asparagine-rich domains in eukaryotic proteomes. Genome Biol. 4, R40. doi:10.1186/gb-2003-4-6-r40

He, D., Parkinson, J., 2008. SubSeqer: a graph-based approach for the detection and identification of repetitive elements in low-complexity sequences. Bioinformatics 24, 1016–1017. doi:10.1093/bioinformatics/btn073

Henikoff, S., Henikoff, J.G., 1992. Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. U. S. A. 89, 10915–10919.

Hoersch, S., Leroy, C., Brown, N.P., Andrade, M.A., Sander, C., 2000. The GeneQuiz Web server: protein functional analysis through the Web. Trends Biochem. Sci. 25, 33–35. doi:10.1016/S0968-0004(99)01510-8

Hunter, S., Jones, P., Mitchell, A., Apweiler, R., Attwood, T.K., Bateman, A., Bernard, T., Binns, D., Bork, P., Burge, S., de Castro, E., Coggill, P., Corbett, M., Das, U., Daugherty, L., Duquenne, L., Finn, R.D., Fraser, M., Gough, J., Haft, D., Hulo, N., Kahn, D., Kelly, E., Letunic, I., Lonsdale, D., Lopez, R., Madera, M., Maslen, J., McAnulla, C., McDowall, J., McMenamin, C., Mi, H., Mutowo-Muellenet, P., Mulder, N., Natale, D., Orengo, C., Pesseat, S., Punta, M., Quinn, A.F., Rivoire, C., Sangrador-Vegas, A., Selengut, J.D., Sigrist, C.J.A., Scheremetjew, M., Tate, J., Thimmajanarthanan, M., Thomas, P.D., Wu, C.H., Yeats, C., Yong, S.-Y., 2012. InterPro in 2011: new developments in the family and domain prediction database. Nucleic Acids Res. 40, D306–D312. doi:10.1093/nar/gkr948

Huntley, M.A., Golding, G.B., 2002. Simple sequences are rare in the Protein Data Bank. Proteins Struct. Funct. Bioinforma. 48, 134–140. doi:10.1002/prot.10150

Iguchi, A., Thomson, N.R., Ogura, Y., Saunders, D., Ooka, T., Henderson, I.R., Harris, D., Asadulghani, M., Kurokawa, K., Dean, P., Kenny, B., Quail, M.A., Thurston, S., Dougan, G., Hayashi, T., Parkhill, J., Frankel, G., 2009. Complete Genome Sequence and Comparative Genome Analysis of Enteropathogenic Escherichia coli O127:H6 Strain E2348/69. J. Bacteriol. 191, 347–354. doi:10.1128/JB.01238-08

Jeong, H., Barbe, V., Lee, C.H., Vallenet, D., Yu, D.S., Choi, S.-H., Couloux, A., Lee, S.-W., Yoon, S.H., Cattolico, L., Hur, C.-G., Park, H.-S., Ségurens, B., Kim, S.C., Oh, T.K., Lenski, R.E., Studier, F.W., Daegelen, P., Kim, J.F., 2009. Genome Sequences of Escherichia coli B strains REL606 and BL21(DE3). J. Mol. Biol. 394, 644–652. doi:10.1016/j.jmb.2009.09.052

Johnson, M., Zaretskaya, I., Raytselis, Y., Merezhuk, Y., McGinnis, S., Madden, T.L., 2008. NCBI BLAST: a better web interface. Nucleic Acids Res. 36, W5–W9. doi:10.1093/nar/gkn201

Kajava, A.V., 2012. Tandem repeats in proteins: From sequence to structure. J. Struct. Biol., Structural Bioinformatics 179, 279–288. doi:10.1016/j.jsb.2011.08.009

Karlin, S., Altschul, S.F., 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Proc. Natl. Acad. Sci. U. S. A. 87, 2264–2268.

Karlin, S., Dembo, A., Kawabata, T., 1990. Statistical Composition of High-Scoring Segments from Molecular Sequences. Ann. Stat. 18, 571–581.

Karlin, S., Mrázek, J., Campbell, A.M., 1997. Compositional biases of bacterial genomes and evolutionary implications. J. Bacteriol. 179, 3899–3913.

Kinsella, R.J., Kahari, A., Haider, S., Zamora, J., Proctor, G., Spudich, G., Almeida-King, J., Staines, D., Derwent, P., Kerhornou, A., Kersey, P., Flicek, P., 2011. Ensembl BioMarts: a hub for data retrieval across taxonomic space. Database 2011, bar030–bar030. doi:10.1093/database/bar030

Kirmitzoglou, I., Promponas, V.J., 2007. Effects of different masking strategies on protein sequence database searches with BLASTP.

Kirmitzoglou, I., Promponas, V.J., 2009. On the quality of established datasets for benchmarking sequence database search and low-complexity handling tools: the ASTRAL compendium test case.

Koo, E.H., Lansbury, P.T., Kelly, J.W., 1999. Amyloid diseases: Abnormal protein aggregation in neurodegeneration. Proc. Natl. Acad. Sci. U. S. A. 96, 9989–9990.

Koonin, E.V., Tatusov, R.L., Rudd, K.E., 1996. Protein sequence comparison at genome scale. Methods Enzymol. 266, 295–322.

Kreil, D.P., Ouzounis, C.A., 2001. Identification of thermophilic species by the amino acid compositions deduced from their genomes. Nucleic Acids Res. 29, 1608–1615.

Kreil, D.P., Ouzounis, C.A., 2003. Comparison of sequence masking algorithms and the detection of biased protein sequence regions. Bioinformatics 19, 1672–1681. doi:10.1093/bioinformatics/btg212

Kriško, A., Smole, Z., Debret, G., Nikolić, N., Radman, M., 2010. Unstructured Hydrophilic Sequences in Prokaryotic Proteomes Correlate with Dehydration Tolerance and Host Association. J. Mol. Biol. 402, 775–782. doi:10.1016/j.jmb.2010.08.012

Kuhn, M., 2008. Building Predictive Models in R Using the caret Package. J. Stat. Softw. 28, 1–26.

Kuhn, M., Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Team, the R.C., 2014. caret: Classification and Regression Training.

Kulasekara, B.R., Jacobs, M., Zhou, Y., Wu, Z., Sims, E., Saenphimmachak, C., Rohmer, L., Ritchie, J.M., Radey, M., McKevitt, M., Freeman, T.L., Hayden, H., Haugen, E., Gillett, W., Fong, C., Chang, J., Beskhlebnaya, V., Waldor, M.K., Samadpour, M., Whittam, T.S., Kaul, R., Brittnacher, M., Miller, S.I., 2009. Analysis of the Genome of the Escherichia coli O157:H7 2006 Spinach-Associated Outbreak Isolate Indicates Candidate Genes That May Enhance Virulence. Infect. Immun. 77, 3713–3721. doi:10.1128/IAI.00198-09

Kuznetsov, I.B., Hwang, S., 2006. A novel sensitive method for the detection of user-defined compositional bias in biological sequences. Bioinformatics 22, 1055–1063. doi:10.1093/bioinformatics/btl049

Lee, C.A., 1996. Pathogenicity islands and the evolution of bacterial pathogens. Infect. Agents Dis. 5, 1–7.

Lees, J., Yeats, C., Redfern, O., Clegg, A., Orengo, C., 2010. Gene3D: merging structure and function for a Thousand genomes. Nucleic Acids Res. 38, D296–D300. doi:10.1093/nar/gkp987

Li, L., Stoeckert, C.J., Roos, D.S., 2003. OrthoMCL: Identification of Ortholog Groups for Eukaryotic Genomes. Genome Res. 13, 2178–2189. doi:10.1101/gr.1224503

Li, X., Kahveci, T., 2006. A Novel algorithm for identifying low-complexity regions in a protein sequence. Bioinformatics 22, 2980–2987. doi:10.1093/bioinformatics/btl495

Lima, T., Auchincloss, A.H., Coudert, E., Keller, G., Michoud, K., Rivoire, C., Bulliard, V., Castro, E. de, Lachaize, C., Baratin, D., Phan, I., Bougueleret, L., Bairoch, A., 2009. HAMAP: a database of completely sequenced microbial proteome sets and manually curated microbial protein families in UniProtKB/Swiss-Prot. Nucleic Acids Res. 37, D471–D478. doi:10.1093/nar/gkn661

Lobanov, M.Y., Furletova, E.I., Bogatyreva, N.S., Roytberg, M.A., Galzitskaya, O.V., 2010. Library of Disordered Patterns in 3D Protein Structures. PLoS Comput Biol 6, e1000958. doi:10.1371/journal.pcbi.1000958

Lobanov, M.Y., Sokolovskiy, I.V., Galzitskaya, O.V., 2014. HRaP: database of occurrence of HomoRepeats and patterns in proteomes. Nucleic Acids Res. 42, D273–D278. doi:10.1093/nar/gkt927

Lovell, S.C., 2003. Are non-functional, unfolded proteins (“junk proteins”) common in the genome? FEBS Lett. 554, 237–239. doi:10.1016/S0014-5793(03)01223-7

Makino, K., Yokoyama, K., Kubota, Y., Yutsudo, C.H., Kimura, S., Kurokawa, K., Ishii, K., Hattori, M., Tatsuno, I., Abe, H., Iida, T., Yamamoto, K., Onishi, M., Hayashi, T., Yasunaga, T., Honda, T., Sasakawa, C., Shinagawa, H., 1999. Complete nucleotide sequence of the prophage VT2-Sakai carrying the verotoxin 2 genes of the enterohemorrhagic Escherichia coli O157:H7 derived from the Sakai outbreak. Genes Genet. Syst. 74, 227–239. doi:10.1266/ggs.74.227

Martinez-Medina, M., Aldeguer, X., Lopez-Siles, M., Gonzalez-Huix, F., Lopez-Oliu, C., Dahbi, G., Blanco, J.E., Blanco, J., Garcia-Gil, J.L., Darfeuille-Michaud, A., 2009. Mo-

lecular diversity of Escherichia coli in the human gut: New ecological evidence supporting the role of adherent-invasive E. coli (AIEC) in Crohn's disease. Inflamm. Bowel Dis. June 2009 15, 872–882. doi:10.1002/ibd.20860

Merhej, V., Georgiades, K., Raoult, D., 2013. Postgenomic analysis of bacterial pathogens repertoire reveals genome reduction rather than virulence factors. Brief. Funct. Genomics 12, 291–304. doi:10.1093/bfgp/elt015

Mészáros, B., Simon, I., Dosztányi, Z., 2009. Prediction of Protein Binding Regions in Disordered Proteins. PLoS Comput Biol 5, e1000376. doi:10.1371/journal.pcbi.1000376

Michelitsch, M.D., Weissman, J.S., 2000. A census of glutamine/asparagine-rich regions: Implications for their conserved function and the prediction of novel prions. Proc. Natl. Acad. Sci. U. S. A. 97, 11910–11915.

Miskinyte, M., Sousa, A., Ramiro, R.S., de Sousa, J.A.M., Kotlinowski, J., Caramalho, I., Magalhães, S., Soares, M.P., Gordo, I., 2013. The Genetic Basis of Escherichia coli Pathoadaptation to Macrophages. PLoS Pathog 9, e1003802. doi:10.1371/journal.ppat.1003802

Mistry, J., Finn, R.D., Eddy, S.R., Bateman, A., Punta, M., 2013. Challenges in homology search: HMMER3 and convergent evolution of coiled-coil regions. Nucleic Acids Res. 41, e121–e121. doi:10.1093/nar/gkt263

Mitas, M., 1997. Trinucleotide repeats associated with human disease. Nucleic Acids Res. 25, 2245–2253. doi:10.1093/nar/25.12.2245

Morgulis, A., Gertz, E.M., Schäffer, A.A., Agarwala, R., 2006. A Fast and Symmetric DUST Implementation to Mask Low-Complexity DNA Sequences. J. Comput. Biol. 13, 1028–1040. doi:10.1089/cmb.2006.13.1028

Moriel, D.G., Bertoldi, I., Spagnuolo, A., Marchi, S., Rosini, R., Nesta, B., Pastorello, I., Corea, V.A.M., Torricelli, G., Cartocci, E., Savino, S., Scarselli, M., Dobrindt, U., Hacker, J., Tettelin, H., Tallon, L.J., Sullivan, S., Wieler, L.H., Ewers, C., Pickard, D., Dougan, G., Fontana, M.R., Rappuoli, R., Pizza, M., Serino, L., 2010. Identification of protective and broadly conserved vaccine antigens from the genome of extraintestinal pathogenic Escherichia coli. Proc. Natl. Acad. Sci. U. S. A. 107, 9072–9077. doi:10.1073/pnas.0915077107

Morschhäuser, J., Vetter, V., Emödy, L., Hacker, J., 1994. Adhesin regulatory genes within large, unstable DNA regions of pathogenic Escherichia coli: cross-talk between different adhesin gene clusters. Mol. Microbiol. 11, 555–566. doi:10.1111/j.1365-2958.1994.tb00336.x

Müller, A., MacCallum, R.M., Sternberg, M.J.E., 2002. Structural Characterization of the Human Proteome. Genome Res. 12, 1625–1641. doi:10.1101/gr.221202

Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C., 1995. SCOP: a structural classification of proteins database for the investigation of sequences and structures. J. Mol. Biol. 247, 536–540. doi:10.1006/jmbi.1995.0159

Nandi, T., Dash, D., Ghai, R., B-Rao, C., Kannan, K., Brahmachari, S.K., Ramakrishnan, C., Ramachandran, S., 2003a. A Novel Complexity Measure for Comparative Analysis of Protein Sequences from Complete Genomes. J. Biomol. Struct. Dyn. 20, 657–667. doi:10.1080/07391102.2003.10506882

Nandi, T., Kannan, K., Ramachandran, S., 2003b. The Low Complexity Proteins from Enteric Pathogenic Bacteria: Taxonomic Parallels Embedded in Diversity. In Silico Biol. 3, 277–285.

Nataro, J.P., Kaper, J.B., 1998. Diarrheagenic Escherichia coli. Clin. Microbiol. Rev. 11, 142–201.

Needleman, S.B., Wunsch, C.D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443–453. doi:10.1016/0022-2836(70)90057-4

Ogura, Y., Ooka, T., Iguchi, A., Toh, H., Asadulghani, M., Oshima, K., Kodama, T., Abe, H., Nakayama, K., Kurokawa, K., Tobe, T., Hattori, M., Hayashi, T., 2009. Comparative genomics reveal the mechanism of the parallel evolution of O157 and non-O157 enterohemorrhagic Escherichia coli. Proc. Natl. Acad. Sci. U. S. A. 106, 17939–17944. doi:10.1073/pnas.0903585106

Oshima, K., Toh, H., Ogura, Y., Sasamoto, H., Morita, H., Park, S.-H., Ooka, T., Iyoda, S., Taylor, T.D., Hayashi, T., Itoh, K., Hattori, M., 2008. Complete Genome Sequence and Comparative Analysis of the Wild-type Commensal Escherichia coli Strain SE11 Isolated from a Healthy Adult. DNA Res. 15, 375–386. doi:10.1093/dnares/dsn026

Overbeek, R., Fonstein, M., D'Souza, M., Pusch, G.D., Maltsev, N., 1999. The use of gene clusters to infer functional coupling. Proc. Natl. Acad. Sci. U. S. A. 96, 2896–2901.

Papadopoulos, A., Kirmitzoglou, I., Promponas, V.J., Theocharides, T., 2012. FPGA-based hardware acceleration for local complexity analysis of massive genomic data. Integr. VLSI J.

Pearson, W.R., 1990. Rapid and sensitive sequence comparison with FASTP and FASTA. Methods Enzymol. 183, 63–98.

Pearson, W.R., Lipman, D.J., 1988. Improved tools for biological sequence comparison. Proc. Natl. Acad. Sci. 85, 2444–2448.

Perna, N.T., Plunkett, G., Burland, V., Mau, B., Glasner, J.D., Rose, D.J., Mayhew, G.F., Evans, P.S., Gregor, J., Kirkpatrick, H.A., Pósfai, G., Hackett, J., Klink, S., Boutin, A., Shao, Y., Miller, L., Grotbeck, E.J., Davis, N.W., Lim, A., Dimalanta, E.T., Potamousis, K.D., Apodaca, J., Anantharaman, T.S., Lin, J., Yen, G., Schwartz, D.C., Welch, R.A., Blattner, F.R., 2001. Genome sequence of enterohaemorrhagic Escherichia coli O157:H7. Nature 409, 529–533. doi:10.1038/35054089

Pietrokovski, S., Hirshon, J., Trifonov, E.N., 1990. Linguistic Measure of Taxonomic and Functional Relatedness of Nucleotide Sequences. J. Biomol. Struct. Dyn. 7, 1251–1268. doi:10.1080/07391102.1990.10508563

Pizzi, E., Frontali, C., 2001. Low-Complexity Regions in Plasmodium falciparum Proteins. Genome Res. 11, 218–229. doi:10.1101/gr.152201

Popov, O., Segal, D.M., Trifonov, E.N., 1996. Linguistic complexity of protein sequences as compared to texts of human languages. Biosystems 38, 65–74. doi:10.1016/0303-2647(95)01568-X

Priyam, A., Woodcroft, B.J., Wurm, Y., n.d. SequenceServer: BLAST searching made easy. in preparation.

Promponas, V.J., 2009. A simple clustering approach for pathogenic strain identification based on local and global amino acid compositional signatures from genomic sequences: the Escherichia genus case, in: 9th International Conference on Information Technology and Applications in Biomedicine, 2009. ITAB 2009. Presented at the 9th International Conference on Information Technology and Applications in Biomedicine, 2009. ITAB 2009, IEEE, Larnaca, Cyprus, pp. 1–4. doi:10.1109/ITAB.2009.5394396

Promponas, V.J., Enright, A.J., Tsoka, S., Kreil, D.P., Leroy, C., Hamodrakas, S., Sander, C., Ouzounis, C.A., 2000. CAST: an iterative algorithm for the complexity analysis of sequence tracts. Bioinformatics 16, 915–922. doi:10.1093/bioinformatics/16.10.915

R Core Team, 2014. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Radivojac, P., Vucetic, S., O'Connor, T.R., Uversky, V.N., Obradovic, Z., Dunker, A.K., 2006. Calmodulin signaling: Analysis and prediction of a disorder-dependent molecular recognition. Proteins Struct. Funct. Bioinforma. 63, 398–410. doi:10.1002/prot.20873

Rakin, A., Urbitsch, P., Heesemann, J., 1995. Evidence for two evolutionary lineages of highly pathogenic Yersinia species. J. Bacteriol. 177, 2292–2298.

Rasko, D.A., Rosovitz, M.J., Myers, G.S.A., Mongodin, E.F., Fricke, W.F., Gajer, P., Crabtree, J., Sebaihia, M., Thomson, N.R., Chaudhuri, R., Henderson, I.R., Sperandio, V., Ravel, J., 2008. The Pangenome Structure of Escherichia coli: Comparative Genomic Analysis of E. coli Commensal and Pathogenic Isolates. J. Bacteriol. 190, 6881–6893. doi:10.1128/JB.00619-08

Romero, P., Obradovic, Z., Li, X., Garner, E.C., Brown, C.J., Dunker, A.K., 2001. Sequence complexity of disordered protein. Proteins Struct. Funct. Bioinforma. 42, 38–48. doi:10.1002/1097-0134(20010101)42:1<38::AID-PROT50>3.0.CO;2-3

Romov, P.A., Li, F., Lipke, P.N., Epstein, S.L., Qiu, W.-G., 2006. Comparative Genomics Reveals Long, Evolutionarily Conserved, Low-Complexity Islands in Yeast Proteins. J. Mol. Evol. 63, 415–425. doi:10.1007/s00239-005-0291-0

Rost, B., Yachdav, G., Liu, J., 2004. The PredictProtein server. Nucleic Acids Res. 32, W321–W326. doi:10.1093/nar/gkh377

Salamon, P., Konopka, A.K., 1992. A maximum entropy principle for the distribution of local complexity in naturally occurring nucleotide sequences. Comput. Chem. 16, 117–124. doi:10.1016/0097-8485(92)80038-2

Schaffer, A.A., Aravind, L., Madden, T.L., Shavirin, S., Spouge, J.L., Wolf, Y.I., Koonin, E.V., Altschul, S.F., 2001. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. Nucleic Acids Res. 29, 2994–3005.

Schlessinger, A., Schaefer, C., Vicedo, E., Schmidberger, M., Punta, M., Rost, B., 2011. Protein disorder — a breakthrough invention of evolution? Curr. Opin. Struct. Biol. 21, 412–418. doi:10.1016/j.sbi.2011.03.014

Schnell, S., Fortunato, S., Roy, S., 2007. Is the intrinsic disorder of proteins the cause of the scale-free architecture of protein–protein interaction networks? PROTEOMICS 7, 961–964. doi:10.1002/pmic.200600455

Shannon, C., 1948. A Mathematical Theory of Communication.

Shin, S.W., Kim, S.M., 2005. A new algorithm for detecting low-complexity regions in protein sequences. Bioinformatics 21, 160–170. doi:10.1093/bioinformatics/bth497

Sigrist, C.J.A., Cerutti, L., Castro, E. de, Langendijk-Genevaux, P.S., Bulliard, V., Bairoch, A., Hulo, N., 2010. PROSITE, a protein domain database for functional characterization and annotation. Nucleic Acids Res. 38, D161–D166. doi:10.1093/nar/gkp885

Sim, K.L., Creamer, T.P., 2004. Protein simple sequence conservation. Proteins Struct. Funct. Bioinforma. 54, 629–638. doi:10.1002/prot.10623

Sims, G.E., Kim, S.-H., 2011. Whole-genome phylogeny of Escherichia coli/Shigella group by feature frequency profiles (FFPs). Proc. Natl. Acad. Sci. 108, 8329–8334. doi:10.1073/pnas.1105168108

Smith, T.F., Waterman, M.S., 1981. Identification of common molecular subsequences. J. Mol. Biol. 147, 195–197. doi:10.1016/0022-2836(81)90087-5

Sonnhammer, E.L.L., Wootton, J.C., 2001. Integrated graphical analysis of protein sequence features predicted from sequence composition. Proteins Struct. Funct. Bioinforma. 45, 262–273. doi:10.1002/prot.1146

Stajich, J.E., Block, D., Boulez, K., Brenner, S.E., Chervitz, S.A., Dagdigian, C., Fuellen, G., Gilbert, J.G.R., Korf, I., Lapp, H., Lehvaslaiho, H., Matsalla, C., Mungall, C.J., Osborne, B.I., Pocock, M.R., Schattner, P., Senger, M., Stein, L.D., Stupka, E., Wilkinson, M.D., Birney, E., 2002. The Bioperl Toolkit: Perl Modules for the Life Sciences. Genome Res. 12, 1611–1618. doi:10.1101/gr.361602

Stein, L.D., Mungall, C., Shu, S., Caudy, M., Mangone, M., Day, A., Nickerson, E., Stajich, J.E., Harris, T.W., Arva, A., Lewis, S., 2002. The Generic Genome Browser: A Building Block for a Model Organism System Database. Genome Res. 12, 1599–1610. doi:10.1101/gr.403602

Stern, L., Allison, L., Coppel, R.L., Dix, T.I., 2001. Discovering patterns in Plasmodium falciparum genomic DNA. Mol. Biochem. Parasitol. 118, 175–186. doi:10.1016/S0166-6851(01)00388-7

Tan, J.C., Tan, A., Checkley, L., Honsa, C.M., Ferdig, M.T., 2010. Variable Numbers of Tandem Repeats in Plasmodium falciparum Genes. J. Mol. Evol. 71, 268–278. doi:10.1007/s00239-010-9381-8

Tatusov, R.L., Lipman, D.J., n.d. unpublished manuscript.

The UniProt Consortium, 2013. UniProt manual [WWW Document]. UniProt Man. URL http://www.uniprot.org/manual (accessed 5.29.14).

The UniProt Consortium, 2014. Activities at the Universal Protein Resource (UniProt). Nucleic Acids Res. 42, D191–D198. doi:10.1093/nar/gkt1140

Touchon, M., Hoede, C., Tenaillon, O., Barbe, V., Baeriswyl, S., Bidet, P., Bingen, E., Bonacorsi, S., Bouchier, C., Bouvet, O., Calteau, A., Chiapello, H., Clermont, O., Cruveiller, S., Danchin, A., Diard, M., Dossat, C., Karoui, M.E., Frapy, E., Garry, L., Ghigo, J.M., Gilles, A.M., Johnson, J., Le Bouguenec, C., Lescat, M., Mangenot, S., Martinez-Jehanne, V., Matic, I., Nassif, X., Oztas, S., Petit, M.A., Pichon, C., Rouy, Z., Ruf, C.S., Schneider, D., Tourret, J., Vacherie, B., Vallenet, D., Medigue, C., Rocha, E.P.C., Denamur, E., 2009. Organised Genome Dynamics in the Escherichia coli Species Results in Highly Diverse Adaptive Paths. PLoS Genet. 5. doi:10.1371/journal.pgen.1000344

Tress, M.L., Cozzetto, D., Tramontano, A., Valencia, A., 2006. An analysis of the Sargasso Sea resource and the consequences for database composition. BMC Bioinformatics 7, 213. doi:10.1186/1471-2105-7-213

Troyanskaya, O.G., Arbell, O., Koren, Y., Landau, G.M., Bolshoy, A., 2002. Sequence complexity profiles of prokaryotic genomic sequences: A fast algorithm for calculating linguistic complexity. Bioinformatics 18, 679–688. doi:10.1093/bioinformatics/18.5.679

Tsoka, S., Promponas, V., Ouzounis, C.A., 1999. Reproducibility in genome sequence annotation: the Plasmodium falciparum chromosome 2 case. FEBS Lett. 451, 354–355. doi:10.1016/S0014-5793(99)00599-2

Uversky, V.N., Oldfield, C.J., Dunker, A.K., 2005. Showing your ID: intrinsic disorder as an ID for recognition, regulation and cell signaling. J. Mol. Recognit. 18, 343–384. doi:10.1002/jmr.747

Walsh, I., Sirocco, F.G., Minervini, G., Domenico, T.D., Ferrari, C., Tosatto, S.C.E., 2012. RAPHAEL: recognition, periodicity and insertion assignment of solenoid protein structures. Bioinformatics 28, 3257–3264. doi:10.1093/bioinformatics/bts550

Wan, H., Li, L., Federhen, S., Wootton, J.C., 2003. Discovering Simple Regions in Biological Sequences Associated with Scoring Schemes. J. Comput. Biol. 10, 171–185. doi:10.1089/106652703321825955

Waterhouse, R.M., Tegenfeldt, F., Li, J., Zdobnov, E.M., Kriventseva, E.V., 2012. OrthoDB: a hierarchical catalog of animal, fungal and bacterial orthologs. Nucleic Acids Res. 41, D358–D365. doi:10.1093/nar/gks1116

Wise, M.J., 2001. 0j.py: a software tool for low complexity proteins and protein domains. Bioinformatics 17, S288–S295. doi:10.1093/bioinformatics/17.suppl_1.S288

Wootton, J.C., 1994a. Non-globular domains in protein sequences: Automated segmentation using complexity measures. Comput. Chem. 18, 269–285. doi:10.1016/0097-8485(94)85023-2

Wootton, J.C., 1994b. Sequences with "unusual" amino acid compositions. Curr. Opin. Struct. Biol. 4, 413–421. doi:10.1016/S0959-440X(94)90111-2

Wootton, J.C., Federhen, S., 1993. Statistics of local complexity in amino acid sequences and sequence databases. Comput. Chem. 17, 149–163. doi:10.1016/0097-8485(93)85006-X

Ye, J., McGinnis, S., Madden, T.L., 2006. BLAST: improvements for better sequence analysis. Nucleic Acids Res. 34, W6–W9. doi:10.1093/nar/gkl164

Yu, Y.-K., Altschul, S.F., 2005. The construction of amino acid substitution matrices for the comparison of proteins with non-standard compositions. Bioinformatics 21, 902–911. doi:10.1093/bioinformatics/bti070

Yu, Y.-K., Gertz, E.M., Agarwala, R., Schaffer, A.A., Altschul, S.F., 2006. Retrieval accuracy, statistical significance and compositional similarity in protein sequence database searches. Nucleic Acids Res. 34, 5966–5973. doi:10.1093/nar/gkl731

Yu, Y.-K., Wootton, J.C., Altschul, S.F., 2003. The compositional adjustment of amino acid substitution matrices. Proc. Natl. Acad. Sci. U. S. A. 100, 15688–15693. doi:10.1073/pnas.2533904100

Zhou, Z., Li, X., Liu, B., Beutin, L., Xu, J., Ren, Y., Feng, L., Lan, R., Reeves, P.R., Wang, L., 2010. Derivation of Escherichia coli O157:H7 from Its O55:H7 Precursor. PLoS ONE 5. doi:10.1371/journal.pone.0008700

6 Supplement

6.1 Services Similar to LCR-eXXXplorer

6.1.1 NCBI BLAST

NCBI BLAST (Altschul *et al.*, 1990; Johnson *et al.*, 2008) is undoubtedly the most commonly employed bioinformatics application. It was designed to efficiently search huge public sequence databases and has built-in support for the detection of LCRs and the elimination of spurious hits caused by their existence. If the user chooses to by-pass the default LCR handling method of compositionally-based statistics (Schaffer *et al.*, 2001) and employ SEG masking instead, detected LCRs are presented in the protein sequence as lower-case letters (Figure 6-1A). Although BLAST is set-up by default to avoid LCRs, once composition-based statistics and SEG masking are turned off (Figure 6-1B), one can use it to search for proteins containing specific LCRs. Nevertheless, this ability is not immediately obvious, especially to the non-technical user[52].

[52] Current online versions of BLASTP contain a bug/feature that prevents users from searching for homopolymeric sequences of A, T, C, G because they are erroneously identified as nucleotide sequences. However, command line versions of BLASTP are not affected.

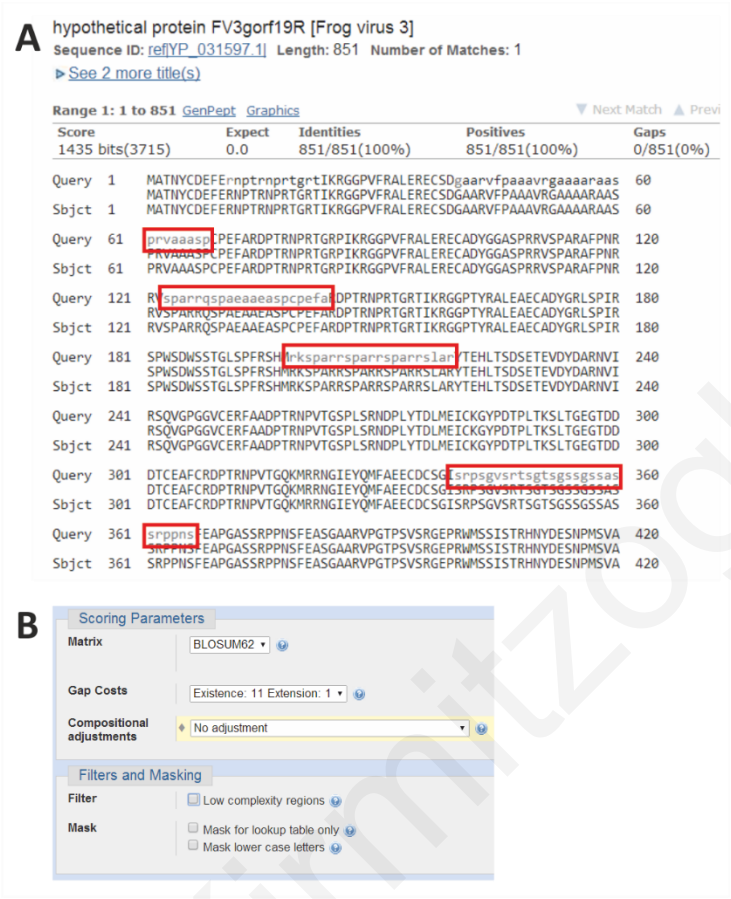


Figure 6-1. *NCBI BLASTP presentation and handling of LCRs.*

A. *Detected LCRs are displayed as lowercase characters in the alignment view.*

B. *Options related with LCRs are hidden by default.*

6.1.2 UniProtKB/SwissProt

The Protein Knowledgebase (UniProtKB) (The UniProt Consortium, 2014) aims to provide "*a comprehensive, high-quality and freely accessible resource of protein sequence and functional annotation*". UniProtKB is divided into two sections: SwissProt (UniProt/SwissProt from now on) which contains around 550,000 manually annotated and reviewed (by expert curators) sequences and TrEMBL with 56 million automatically annotated –and not reviewed– protein sequences (statistics for UniProt release 2014_05). As such it integrates and provides data from the literature, computational tools and other online resources. Localized sequence features are graphically displayed as rectangles covering parts of the sequence and are fully searchable through a simple, yet powerful, interface (Figure 6-2).

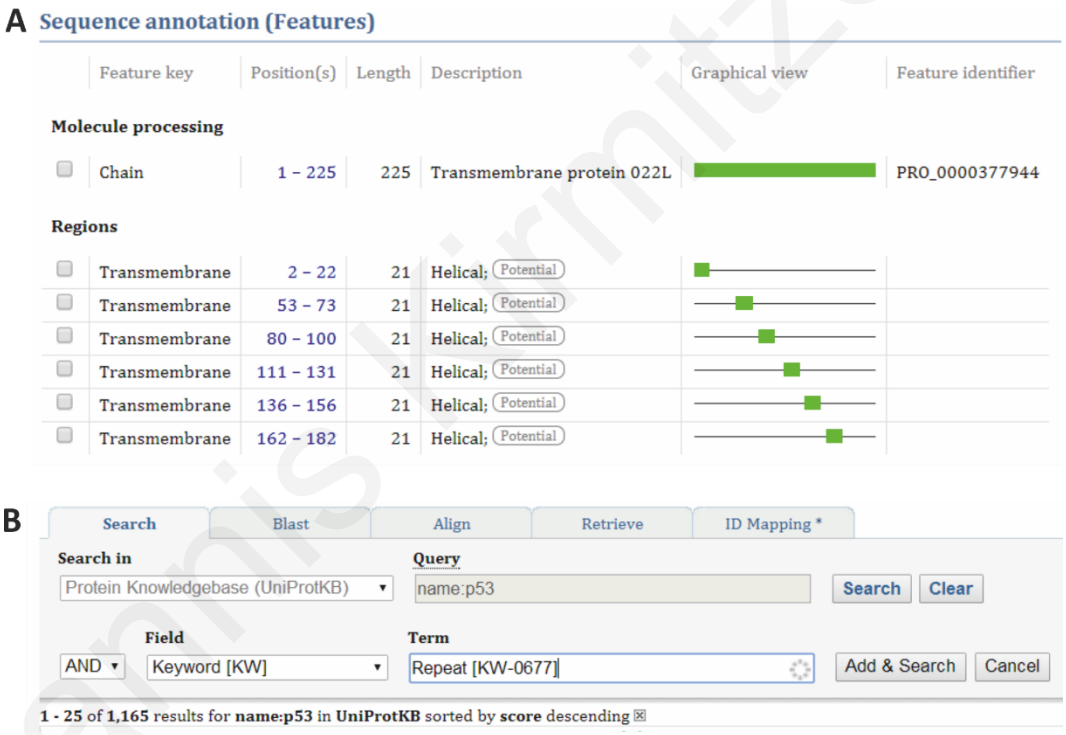


Figure 6-2. *Annotations in UniProt/SwissProt.*

A. *Sequence features depicted in UniProt/SwissProt*

B. *The search form.*

UniProtKB provides annotations for 2 major LCR-related feature types, namely regions of compositional bias and repeats (The UniProt Consortium, 2013). It should be noted here that repeats, especially the long ones or those associated with structural elements, do not

always lower the complexity of a region enough to be considered as LCRs. However, we consider them as features that might be of potential interest for a researcher searching for LCRs. Annotations of type "*Compositional bias*" cover two categories of LCRs; *homo-polymeric stretches* of at least 4 residues in length and long *regions of compositional bias*. It is worth mentioning that although the UniProtKB manual clearly describes how *homo-polymeric stretches* are detected, the same doesn't apply for the *regions of compositional bias*. Repeats typical of a specific protein or protein family are annotated as "*Repeats*", while repeats that don't have a standardized name are annotated using the "*Region*" type, accompanied by a description and the pattern of each repeat (Figure 6-3). A large fraction of the annotated repeats in UniProtKB are either modelled by InterPro (Hunter *et al.*, 2012) or predicted by the software REP (Andrade *et al.*, 2000), which detects common repeat families such as HEAT and WD40 repeats. In total, as for May 1st 2014, there are 40,492 (7.43% of the total number of sequences) proteins in UniProt/SwissProt annotated with at least one LCR type, an unexpectedly low figure. By contrast, the percentage of protein sequences in LCR-eXXXplorer annotated with at least one LCR is 58.87% (57.85% of all the sequences have SEG annotations, 18.38% have CAST annotations).

Q03763 (DSG1_BOVIN) Reviewed, UniProtKB/Swiss-Prot

Regions

☐	Topological domain	50 – 551	502	Extracellular (Potential)
☐	Transmembrane	552 – 572	21	Helical; (Potential)
☐	Topological domain	573 – 1043	471	Cytoplasmic (Potential)
☐	Domain	50 – 158	109	Cadherin 1
☐	Domain	159 – 270	112	Cadherin 2
☐	Domain	271 – 385	115	Cadherin 3
☐	Domain	386 – 498	113	Cadherin 4
☐	Repeat	819 – 845	27	Desmoglein repeat 1
☐	Repeat	846 – 875	30	Desmoglein repeat 2
☐	Repeat	876 – 905	30	Desmoglein repeat 3
☐	Repeat	906 – 933	28	Desmoglein repeat 4
☐	Repeat	934 – 962	29	Desmoglein repeat 5
☐	Compositional bias	963 – 1012	50	Gly/Ser-rich

Q96NY7 (CLIC6_HUMAN) Reviewed, UniProtKB/Swiss-Prot

Regions

☐	Transmembrane	489 – 509	21	Helical; Note=After insertion into the membrane; (Potential)
☐	Repeat	157 – 166	10	1
☐	Repeat	167 – 176	10	2
☐	Repeat	177 – 186	10	3
☐	Repeat	187 – 196	10	4
☐	Repeat	197 – 206	10	5
☐	Repeat	207 – 216	10	6
☐	Repeat	217 – 226	10	7
☐	Repeat	227 – 236	10	8
☐	Repeat	237 – 246	10	9
☐	Repeat	247 – 256	10	10
☐	Repeat	257 – 266	10	11
☐	Repeat	267 – 276	10	12
☐	Repeat	277 – 286	10	13
☐	Domain	556 – 704	149	GST C-terminal
☐	Region	157 – 282	126	13 X 10 AA tandem repeat of G-D-[SNG]-[VIM]-[DEQ]-A-[EAG]-[GDVE]-[PRG]-[LAVP]

Figure 6-3. *Examples of repeats annotated in UniProt/SwissProt.*

Repeats with standard (top) versus non-standard (bottom) names in UniProt/SwissProt.

6.1.3 InterPro

InterPro (Hunter *et al.*, 2012) is a web-service hosted by the European Bioinformatics Institute that aims to provide a unified interface for fast and easy access to information about protein families, domains and functional sites. Annotations originate from one of the (currently 12) protein-related databases indexed by InterPro, including Pfam (Finn *et al.*, 2010), PROSITE (Sigrist *et al.*, 2010), CATH-Gene3D (Lees *et al.*, 2010) and HAMAP (Lima *et al.*, 2009). All InterPro annotations are manually curated and, importantly, annotations from multiple databases describing the same feature are cross-referenced whenever possible.

InterPro provides four entry types (*families, domains, repeats* and *sites*) and while any of them may be associated with regions of low complexity, most of the annotated LCRs belong to the *repeats* type. Table 13 presents examples of InterPro annotations related to LCRs.

Table 13. *LCR-related entries in InterPro may belong to any of the four entry types.*

InterPro ID	Type	Description
IPR029394	Domain	AP-3 complex subunit beta 1, serine-rich domain
IPR009408	Domain	Formin Homology 1
IPR005899	Family	Sodium ion-translocating decarboxylase
IPR014247	Family	Sporulation lipoprotein YhcN/YlaJ
IPR022263	Site	KxYKxGKxW signal peptide
IPR000357	Repeat	HEAT
IPR006553	Repeat	Leucine-rich repeat, cysteine-containing subtype

InterPro offers various ways of searching for annotations, from a simple search box to a BioMart instance (Kinsella *et al.*, 2011) allowing a user to construct elaborate queries and download the results in several formats for further analysis. A unique way of searching InterPro is by means of domain organization (Figure 6-4) where a user can discover protein sequences with a particular set (or order) of domains. Since several of the annotated domains are LCR-related, this provides a powerful way of searching for sequences with specific combinations of such regions or with LCRs that coexist with selected functional or structural domains.

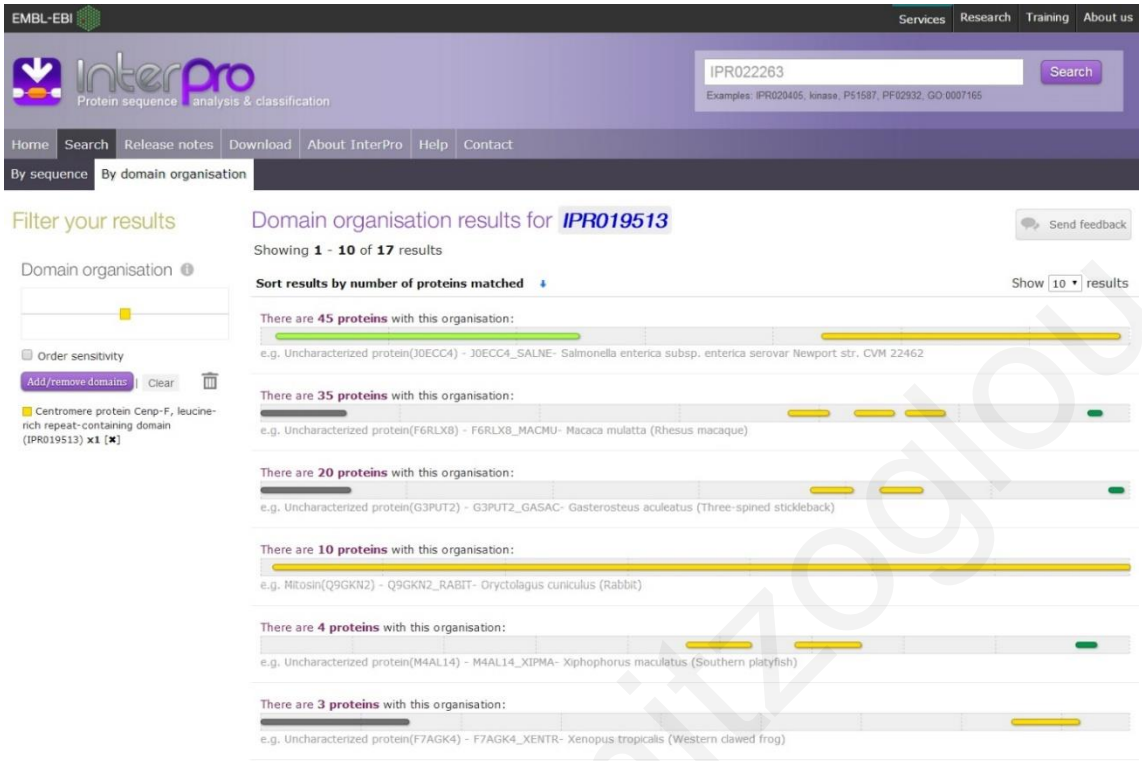


Figure 6-4. *Searching by domain organisation in InterPro.*

Searching IntePro entries by domain organisation provides a powerful way of searching for sequences with specific combinations of LCRs or with LCRs that coexist with selected functional or structural domains.

Despite all these features, InterPro is not a tool particularly suited to discover, display and study low complexity regions. Its main culprit is the structured and curated nature of the database; most stored features must be common enough and well understood to end up in InterPro. For example, searching for “Proline-rich” will return repeats or domains rich in proline that are known to be associated with a specific function or structure; it will also display all families containing proline-rich proteins although it is possible that not every family member will be proline-rich (which in the context of an LCRs search can be considered as false positive results). This is adequate if one wants to study the biological relevance of known repeat types but is limiting for discovering novel functional or structural properties of LCRs.

6.1.4 RepeatsDB

RepeatsDB (http://repeatsdb.bio.unipd.it/) is a database of annotated tandem repeats in protein structures. Its primary aim is to address the lack of a universal resource for structural protein repeats. Di Domenico *et al.* (2014) applied RAPHAEL (Walsh *et al.*, 2012), a software for the detection of tandem repeats based on protein structural data, to all structures stored in PDB and came up with 10745 predicted repeats. Following manual annotation and homology detection, 2797 of these repeats where classified in 18 classes and subclasses based on the classification scheme proposed by Kajava (2012); detailed annotations –such as the location of the repeats along the sequence– are provided for 321 proteins.

A user can browse a tree-like structure following the classification scheme and acquire a detailed list of proteins belonging to each class in various formats. Furthermore, users can search RepeatsDB using PDB IDs or Uniprot/SwissProt Accession numbers or even perform advanced searches using the already established and powerful UniProtKB search engine. Details for the 2797 classified sequences are limited to general information, such as the name of the protein, PFAM annotations and cross-links to other popular classifications of 3D structures (such as CATH and SCOP). For the 321 proteins having detailed annotations, useful information (such as the start and end positions of the repeated units and insertions) is also available, through a functional and attractive web interface (Figure 6-5). Available PFAM annotations are overlaid on the protein sequence along with repeat units and insertions. Detected repeats and insertions are mapped on an interactive 3D structure display of the protein that allows a user to carefully study the significance of the repeats on the molecular and structural level (Figure 57).

Overall, RepeatsDB has the potential to become a very useful resource for the study of tandem repeats in general as well as LCR-related repeats, but is currently limited by the very small number of protein structures annotated in detail. Still, the authors have clearly expressed that RepeatsDB will continue to be updated and we expect its usefulness to increase as more proteins are manually annotated into the database.

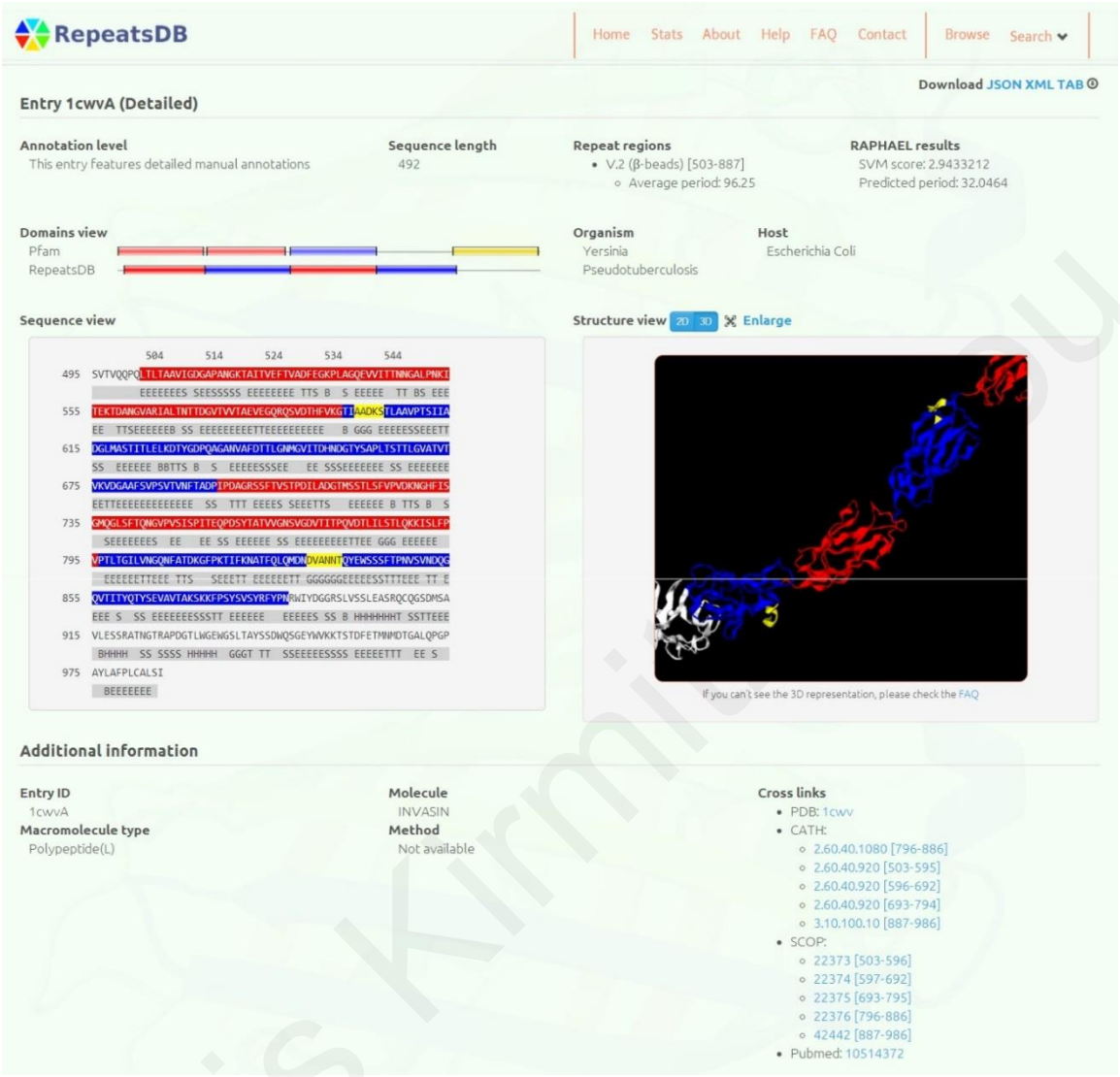


Figure 6-5. *The user interface of RepeatsDB.*

Detailed view for the protein Invasin from Yersinia pseudotuberculosis (PDB ID 1cwvA) in RepeatsDB. Annotated repeats are overlaid on the protein sequence (left panel) as well as on an interactive 3D view of the protein structure (right panel).

6.1.5 HRaP

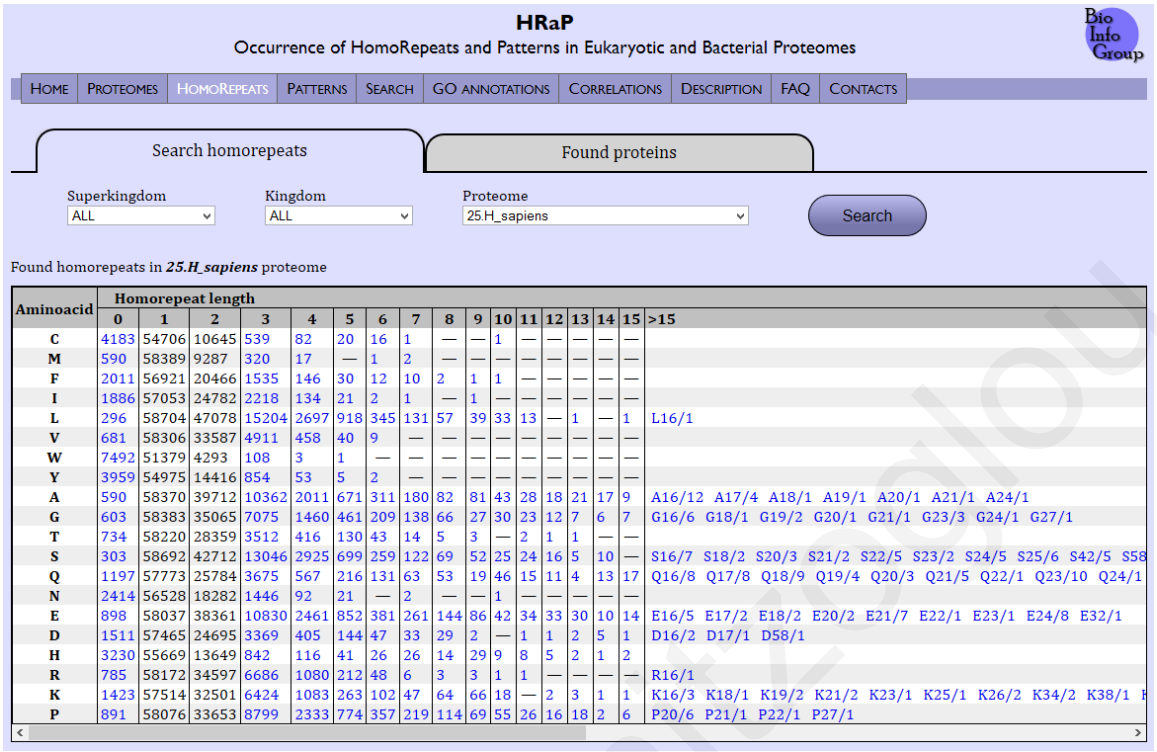
HRaP (http://bioinfo.protres.ru/hrap/) (Lobanov *et al.*, 2014) is a recently developed database of homorepeats and disordered patterns. Homorepeats (or homopolymeric repeats) comprise a special case of tandem repeats, where a single amino acid type is repeated several times and are usually associated with low complexity. Disordered patterns, in the context of HRaP, are a list of 171 short patterns that have been found to commonly occur within disordered regions of proteins with experimentally determined

3D structures in a previous study of the same group (Lobanov *et al.*, 2010). HRaP allows a user to search and download all detected occurrences of homorepeats and disordered patterns in 97 eukaryotic and 25 bacterial proteomes taken from UniProtKB. A user may initiate a search by selecting a specific Superkingdom/Kingdom/proteome; alternatively they may search HRaP by protein name.

In the case of homorepeats, results are displayed in a cross-tabulated format with the number of the occurrences for each of the 20 different homorepeat types grouped by length (Figure 6-6). Clicking on a specific homorepeat (e.g. H6 or HHHHHH) will display all protein sequences of the selected proteome having the pattern, along with the option to display or download all the results. Selecting any of the proteins will display its sequence with the searched homorepeat highlighted (Figure 6-7) along with any other homorepeats, disordered patterns or GO terms associated with the protein. Searching for proteins with specific disordered patterns works in a similar way.

HRaP also provides some interesting statistics, such as the associations of homorepeats/disordered patterns with function (based on GO term annotations) or the occurrence of the 20 homorepeat types of specific length along the proteomes.

Overall, HRaP is a useful tool for any researcher aiming to study the importance of LCRs in protein sequences, despite the fact that it only covers homorepeats and disordered patterns and ignores other known LCR types. Another drawback is the lack of a more informative *"protein details"* view where additional info, such as annotations from UniProt/SwissProt or PFAM, could be displayed along the detected patterns.



Aminoacid	Homorepeat length															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	>15
C	4183	54706	10645	539	82	20	16	1	—	—	1	—	—	—	—	—	
M	590	58389	9287	320	17	—	1	2	—	—	—	—	—	—	—	—	
F	2011	56921	20466	1535	146	30	12	10	2	1	1	—	—	—	—	—	
I	1886	57053	24782	2218	134	21	2	1	—	1	—	—	—	—	—	—	
L	296	58704	47078	15204	2697	918	345	131	57	39	33	13	—	1	—	1	L16/1
V	681	58306	33587	4911	458	40	9	—	—	—	—	—	—	—	—	—	
W	7492	51379	4293	108	3	1	—	—	—	—	—	—	—	—	—	—	
Y	3959	54975	14416	854	53	5	2	—	—	—	—	—	—	—	—	—	
A	590	58370	39712	10362	2011	671	311	180	82	81	43	28	18	21	17	9	A16/12 A17/4 A18/1 A19/1 A20/1 A21/1 A24/1
G	603	58383	35065	7075	1460	461	209	138	66	27	30	23	12	7	6	7	G16/6 G18/1 G19/2 G20/1 G21/1 G23/3 G24/1 G27/1
T	734	58220	28359	3512	416	130	43	14	5	3	—	2	1	1	—	—	
S	303	58692	42712	13046	2925	699	259	122	69	52	25	24	16	5	10	—	S16/7 S18/2 S20/3 S21/2 S22/5 S23/2 S24/5 S25/6 S42/5 S58
Q	1197	57773	25784	3675	567	216	131	63	53	19	46	15	11	4	13	17	Q16/8 Q17/8 Q18/9 Q19/4 Q20/3 Q21/5 Q22/1 Q23/10 Q24/1
N	2414	56528	18282	1446	92	21	—	2	—	—	1	—	—	—	—	—	
E	898	58037	38361	10830	2461	852	381	261	144	86	42	34	33	30	10	14	E16/5 E17/2 E18/2 E20/2 E21/7 E22/1 E23/1 E24/8 E32/1
D	1511	57465	24695	3369	405	144	47	33	29	2	—	1	1	2	5	1	D16/2 D17/1 D58/1
H	3230	55669	13649	842	116	41	26	26	14	29	9	8	5	2	1	2	
R	785	58172	34597	6686	1080	212	48	6	3	3	1	1	—	—	—	—	R16/1
K	1423	57514	32501	6424	1083	263	102	47	64	66	18	—	2	3	1	1	K16/3 K18/1 K19/2 K21/2 K23/1 K25/1 K26/2 K34/2 K38/1 K
P	891	58076	33653	8799	2333	774	357	219	114	69	55	26	16	18	2	6	P20/6 P21/1 P22/1 P27/1

Figure 6-6. *Homorepeats found in Homo sapiens according to HRaP.*

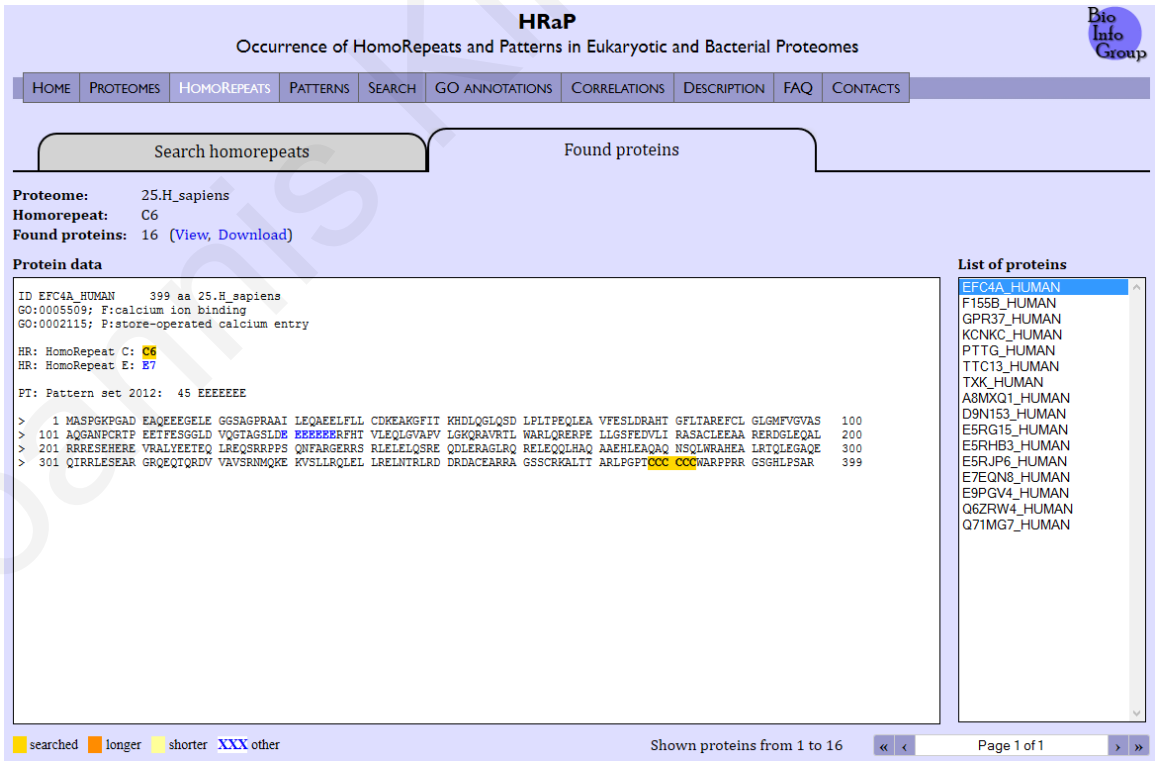


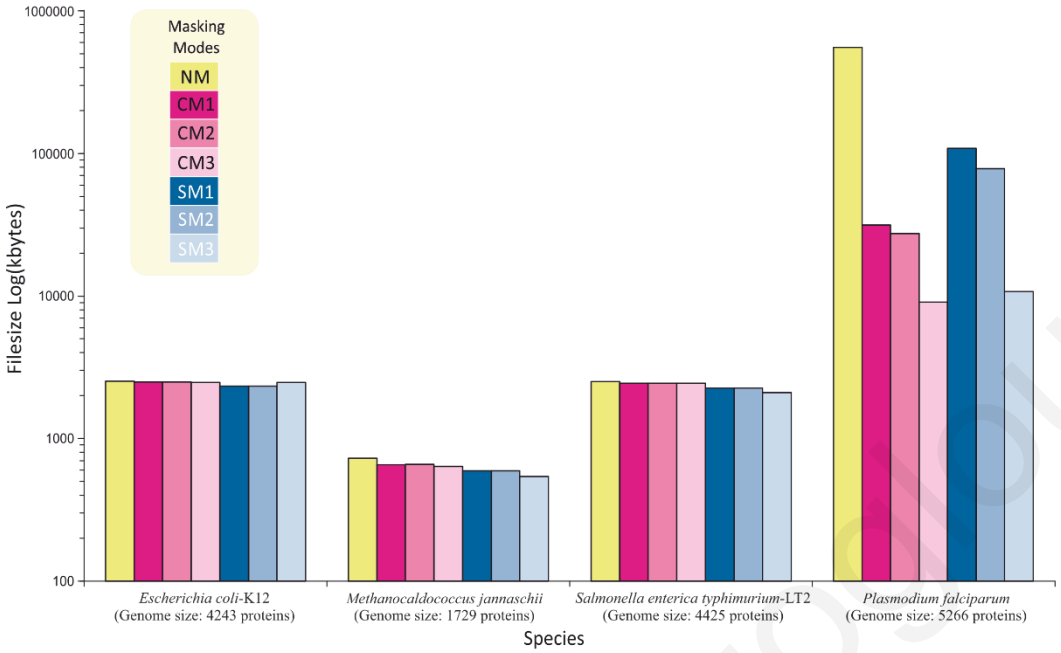
Figure 6-7. *Detailed view of a protein matching search criteria in HRaP.*

6.2 Supplementary Figures & Tables

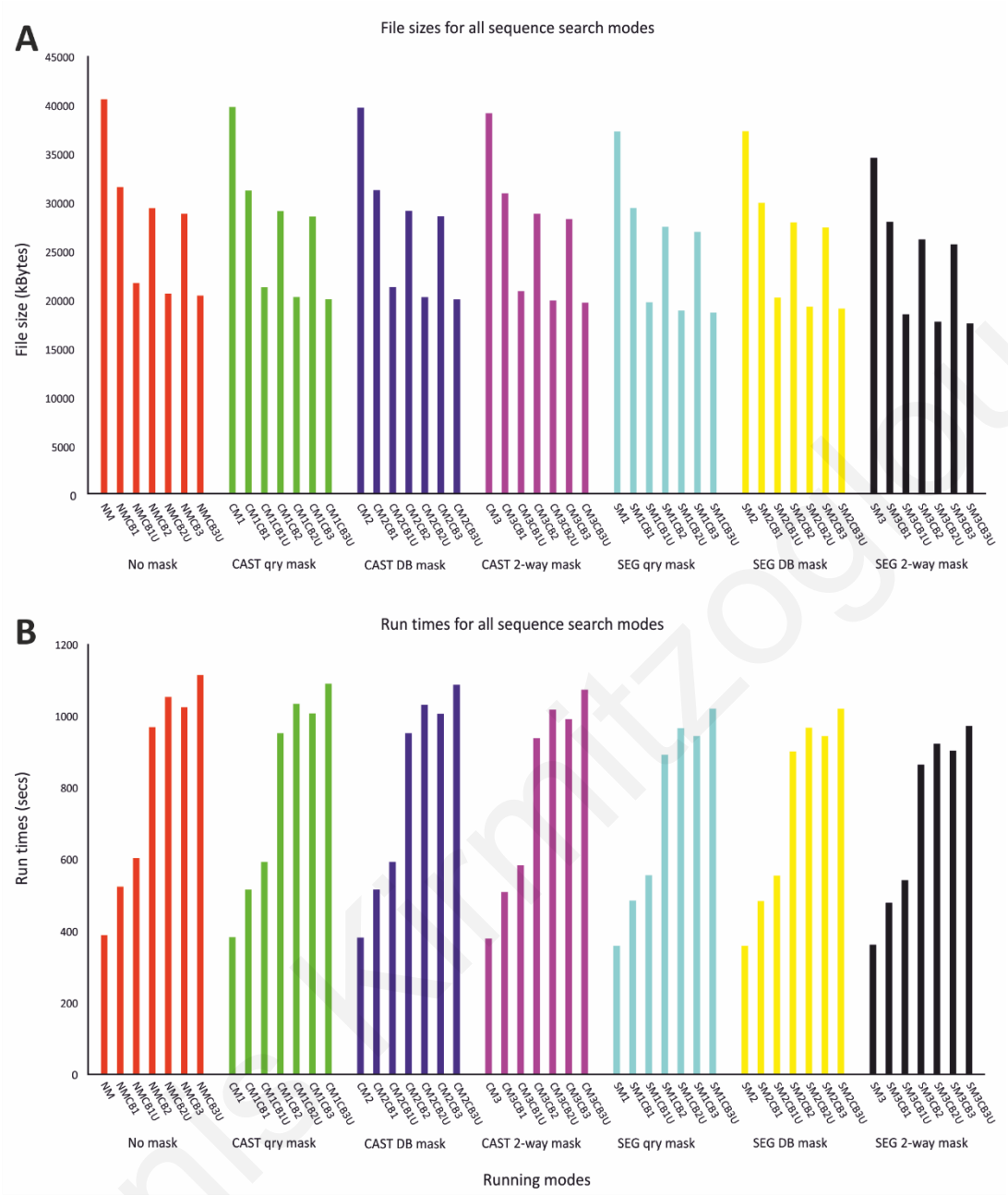
Supplementary Figure 1. *The BLOSUM62 substitution matrix.*

Taken from (Eddy, 2004).

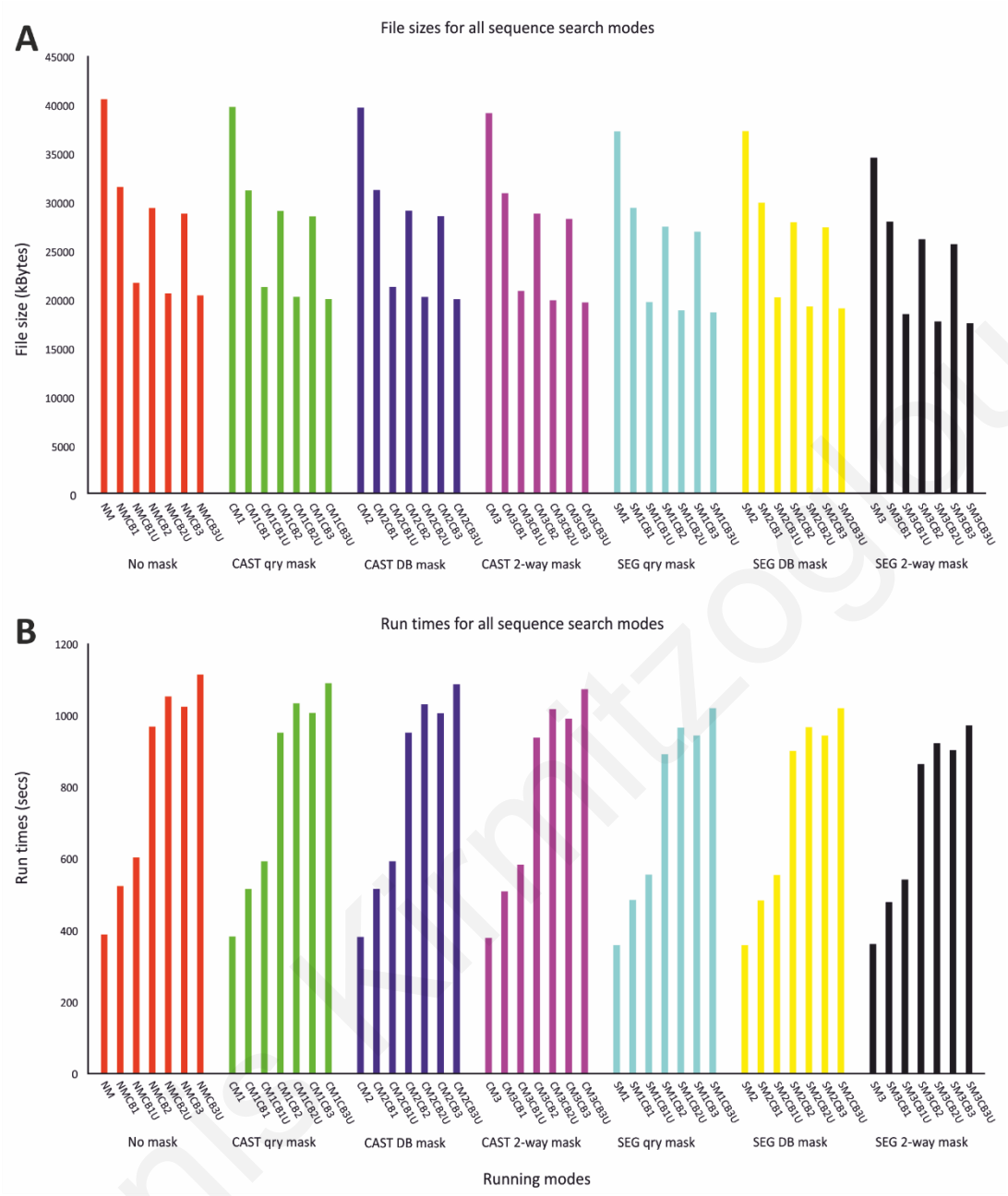
```
#  Matrix made by matblas from blosum62.iij
#  * column uses minimum score
#  BLOSUM Clustered Scoring Matrix in 1/2 Bit Units
#  Blocks Database = /data/blocks_5.0/blocks.dat
#  Cluster Percentage: >= 62
#  Entropy =   0.6979, Expected =  -0.5209
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V  B  Z  X  *
A  4 -1 -2 -2  0 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -3 -2  0 -2 -1  0 -4
R -1  5  0 -2 -3  1  0 -2  0 -3 -2  2 -1 -3 -2 -1 -1 -3 -2 -3 -1  0 -1 -4
N -2  0  6  1 -3  0  0  0  1 -3 -3  0 -2 -3 -2  1  0 -4 -2 -3  3  0 -1 -4
D -2 -2  1  6 -3  0  2 -1 -1 -3 -4 -1 -3 -3 -1  0 -1 -4 -3 -3  4  1 -1 -4
C  0 -3 -3 -3  9 -3 -4 -3 -3 -1 -1 -3 -1 -2 -3 -1 -1 -2 -2 -1 -3 -3 -2 -4
Q -1  1  0  0 -3  5  2 -2  0 -3 -2  1  0 -3 -1  0 -1 -2 -1 -2  0  3 -1 -4
E -1  0  0  2 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
G  0 -2  0 -1 -3 -2 -2  6 -2 -4 -4 -2 -3 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -4
H -2  0  1 -1 -3  0  0 -2  8 -3 -3 -1 -2 -1 -2 -1 -2 -2  2 -3  0  0 -1 -4
I -1 -3 -3 -3 -1 -3 -3 -4 -3  4  2 -3  1  0 -3 -2 -1 -3 -1  3 -3 -3 -1 -4
L -1 -2 -3 -4 -1 -2 -3 -4 -3  2  4 -2  2  0 -3 -2 -1 -2 -1  1 -4 -3 -1 -4
K -1  2  0 -1 -3  1  1 -2 -1 -3 -2  5 -1 -3 -1  0 -1 -3 -2 -2  0  1 -1 -4
M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1 -3 -1 -1 -4
F -2 -3 -3 -3 -2 -3 -3 -3 -1  0  0 -3  0  6 -4 -2 -2  1  3 -1 -3 -3 -1 -4
P -1 -2 -2 -1 -3 -1 -1 -2 -2 -3 -3 -1 -2 -4  7 -1 -1 -4 -3 -2 -2 -1 -2 -4
S  1 -1  1  0 -1  0  0  0 -1 -2 -2  0 -1 -2 -1  4  1 -3 -2 -2  0  0  0 -4
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -2 -1  1  5 -2 -2  0 -1 -1  0 -4
W -3 -3 -4 -4 -2 -2 -3 -2 -2 -3 -2 -3 -1  1 -4 -3 -2 11  2 -3 -4 -3 -2 -4
Y -2 -2 -2 -3 -2 -1 -2 -3  2 -1 -1 -2 -1  3 -3 -2 -2  2  7 -1 -3 -2 -1 -4
V  0 -3 -3 -3 -1 -2 -2 -3 -3  3  1 -2  1 -1 -2 -2  0 -3 -1  4 -3 -2 -1 -4
B -2 -1  3  4 -3  0  1 -1  0 -3 -4  0 -3 -3 -2  0 -1 -4 -3 -3  4  1 -1 -4
Z -1  0  0  1 -3  3  4 -2  0 -3 -3  1 -1 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
X  0 -1 -1 -1 -2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2  0  0 -2 -1 -1 -1 -1 -1 -4
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1
```



Supplementary Figure 2. *File sizes of the BLASTP output files when employed with the default E-value threshold (10) for the full genome self-comparisons. For relevant definitions refer to Table 2 and section 2.1.3 of the manuscript.*



Supplementary Figure 3. *File sizes (A) and run-times (B) for all sequence search modes (for relevant definitions refer to Table 2 and section 2.1.3) presented herein.*



Supplementary Figure 4. *File sizes (A) and run-times (B) for all 49 sequence search modes for the ASTRAL-based benchmarks (for relevant definitions refer to Table 2 and section 2.1.3) presented herein.*

6.3 Publications / Presentations derived from this work

Kirmitzoglou I and Promponas VJ. *LCRs-eXXXplorer: a web platform to search, visualize and share data for low complexity regions in protein sequences*. (Submitted for publication)

Kirmitzoglou I, Papadopoulos A, […] and Promponas VJ. *A new generation of software and hardware tools for detecting compositionally biased regions in protein sequences*. (In preparation)

Kirmitzoglou I and Promponas VJ. *Pathogenicity inference for Escherichia stains using compositional features from (partial) proteomic data.* (In preparation)

Kirmitzoglou, I., & Promponas, V. J. (2009, June 27). *On the quality of established datasets for benchmarking sequence database search and low-complexity handling tools: the ASTRAL compendium test case*. Presented at the 17th International Conference on Intelligent Systems for Molecular Biology and 8th European Conference on Computational Biology, Stockholm, Sweden.

Kirmitzoglou, I., Mytilineou, E., & Promponas, V. J. (2008, October). *Local compositional extremes in protein sequences: definitions, algorithms and biological relevance*. Presented at the 3rd Conference of the Hellenic Society for Computational Biology, CERTH, Thessaloniki, Greece.

Kirmitzoglou, I., & Promponas, V. J. (2008, October). *Effects of different masking strategies on protein sequence database searches with BLASTP*. Presented at the 3rd Conference of the Hellenic Society for Computational Biology,, CERTH, Thessaloniki, Greece.

Kirmitzoglou, I., & Promponas, V. J. (2007, July 21). *Effects of different masking strategies on protein sequence database searches with BLASTP*. Presented at the 15th International Conference on Intelligent Systems for Molecular Biology and 6th European Conference on Computational Biology, Vienna, Austria.

6.4 Publications / Presentations related with this work

Chrysos, G., Sotiriades, E., Rousopoulos, C., Pramataris, K., Papaefstathiou, I., Dollas, A., … Lagnel, J. (2014). Reconfiguring the Bioinformatics Computational Spectrum: Challenges and Opportunities of FPGA-Based Bioinformatics Acceleration Platforms. *IEEE Design Test*, *31*(1), 62–73. doi:10.1109/MDAT.2013.2284191

Papadopoulos, A., Kirmitzoglou, I., Promponas, V. J., & Theocharides, T. (2013). GPU technology as a platform for accelerating local complexity analysis of protein sequences. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (pp. 2684–2687). doi:10.1109/EMBC.2013.6610093

Papadopoulos, A., Kirmitzoglou, I., Promponas, V. J., & Theocharides, T. (2012). FPGA-based hardware acceleration for local complexity analysis of massive genomic data. *Integration, the VLSI Journal*. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167926012000697

Chrysos, G., Sotiriades, E., Rousopoulos, C., Dollas, A., Papadopoulos, A., Kirmitzoglou, I., … Lagnel, J. (2012). Opportunities from the use of FPGAs as platforms for bioinformatics algorithms. In *2012 IEEE 12th International Conference on Bioinformatics Bioengineering (BIBE)* (pp. 559–565). doi:10.1109/BIBE.2012.6399733

Papadopoulos, A., Kirmitzoglou, I., Promponas, V. J., & Theocharides, T. (2012). FPGA-based hardware acceleration for local complexity analysis of massive genomic data. *Integration, the VLSI Journal*. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167926012000697

6.5 Other Publications

Kalvari, I., Lamprinidis, G., Kirmitzoglou, I., Kannas, C. C., Achilleos, K. G., Antoniou, Z., … Promponas, V. J. (2013, July 21). *Enabling docking-based virtual screening within the GRANATUM-LiSIs platform*. Presented at the 12th European Conference on Computational Biology, Berlin, Germany.

Kannas, C. C., Achilleos, K. G., Antoniou, Z., Kalvari, I., Kirmitzoglou, I., Nicolaou, C. A., … Pattichis, C. S. (2013, July). *LiSIs: An Online Scientific Workflow System for Life Sciences Research*. Presented at the 6th Joint Sheffield Conference on Chemoinformatics, Sheffield, UK.

Kannas, C. C., Achilleos, K. G., Antoniou, Z., Nicolaou, C. A., Pattichis, C. S., Kalvari, I., … Promponas, V. J. (2012). A workflow system for virtual screening in cancer chemoprevention. In *2012 IEEE 12th International Conference on Bioinformatics Bioengineering (BIBE)* (pp. 439–446). doi:10.1109/BIBE.2012.6399766

Kannas, C. C., Achilleos, K. G., Antoniou, Z., Nicolaou, C. A., Pattichis, C. S., Neophytou, C. M., … Promponas, V. J. (2012, October 4). *GRANATUM-LISIs: Making complex in silico predictive models accessible to wet biologists*. Presented at the 7th Conference of the Hellenic Society for Computational Biology & Bioinformatics, Heraclion, Greece.

Kannas, C. C., Kalvari, I., Kirmitzoglou, I., Antoniou, Z., Achilleos, K. G., Neophytou, C. M., … Pattichis, C. S. (2013). *LiSIs: A Galaxy-based platform for Life Sciences Informatics Research*. Presented at the Galaxy Community Conference 2013, Oslo, Norway.

Kannas, C. C., Kalvari, I., Lambrinidis, G., Neophytou, C. M., Savva, C. G., Kirmitzoglou, I., … Pattichis, C. S. (2013). *A Scientific Workflow Management System for Virtual Screening in Cancer Chemoprevention (submitted)*.

Neophytou, C., Savva, C., Promponas, V. J., Kalvari, I., Kannas, C., Kirmitzoglou, I., … Constantinou, A. (2013, March 13). *Application of computational modeling for identifying promising agents to control breast cancer*. Presented at the 13th International Breast Cancer Conference, St Gallen Switzerland.

Neophytou, C., Savva, C., Promponas, V., Kalvari, I., Kannas, C., Kirmitzoglou, I., … Constantinou, A. (2012, October). *An interdisciplinary computational and systems biology approach for the selection of cancer chemoprevention agents*. Presented at the 5th International Congress on Nutrition and Cancer,, Elazig, Turkey.

Nikolaou, N., Sampaziotis, P., Aplikioti, M., Drakos, A., Kirmitzoglou, I., Argyrou, M., … Promponas, V. J. (2010, October). *An Open Source system for semi-automatic species identification from digital images*. Presented at the In the 5th Conference of the Hellenic Society for Computational Biology, Alexandroupolis, Greece.

Ntertilis, M., Kirmitzoglou, I., Promponas, V. J., Kouvelis, V., & Typas, M. (2013). *MitoFun: A Curated Resource of Complete Fungal Mitochondrial Genomes (submitted)*.

Tamana, S., Kirmitzoglou, I., & Promponas, V. J. (2011, July 17). *Accessibility of Compositionally Biased Regions in PDB structures*. Presented at the 19th International Conference on Intelligent Systems for Molecular Biology and 10th European Conference on Computational Biology, Vienna, Austria.

Xenophontos, M., Theodosiou, A., Kirmitzoglou, I., & Promponas, V. J. (2011, July 17). *Evaluation of transmembrane helix (TMH) packing prediction methods on successive in sequence TMHs*. Presented at the 19th International Conference on Intelligent Systems for Molecular Biology and 10th European Conference on Computational Biology, Vienna, Austria.

6.6 Code Snippets

Code Snippet 6-1. *SQL statements required do add view "attributespivot" in a database based on the SeqFeature schema.*

```
CREATE VIEW tags as
(SELECT attribute.id as 'ID',
        feature.seqid as 'SEQID',
       name.name as 'NAME',
       attributelist.tag as 'TAG',
       attribute.attribute_value as 'VALUE',
   feature.end - feature.start + 1 as 'LENGTH'
FROM attribute
JOIN attributelist
       ON attribute.attribute_id = attributelist.id
JOIN feature
       ON attribute.id = feature.id
JOIN name
       ON attribute.id = name.id
WHERE feature.typeid = 1
      AND name.display_name = 1
ORDER BY attribute.id, attributelist.tag);

CREATE VIEW tags_extended as (
SELECT tags.*,
    case when TAG = "cA" then VALUE end as cA,
    case when TAG = "cC" then VALUE end as cC,
    case when TAG = "cD" then VALUE end as cD,
    case when TAG = "cE" then VALUE end as cE,
    case when TAG = "cF" then VALUE end as cF,
    case when TAG = "cG" then VALUE end as cG,
    case when TAG = "cH" then VALUE end as cH,
    case when TAG = "cI" then VALUE end as cI,
    case when TAG = "cK" then VALUE end as cK,
    case when TAG = "cL" then VALUE end as cL,
    case when TAG = "cM" then VALUE end as cM,
    case when TAG = "cN" then VALUE end as cN,
    case when TAG = "cP" then VALUE end as cP,
    case when TAG = "cQ" then VALUE end as cQ,
    case when TAG = "cR" then VALUE end as cR,
    case when TAG = "cS" then VALUE end as cS,
    case when TAG = "cT" then VALUE end as cT,
    case when TAG = "cV" then VALUE end as cV,
    case when TAG = "cW" then VALUE end as cW,
    case when TAG = "cY" then VALUE end as cY,
    case when TAG = "sA" then VALUE end as sA,
    case when TAG = "sC" then VALUE end as sC,
    case when TAG = "sD" then VALUE end as sD,
    case when TAG = "sE" then VALUE end as sE,
    case when TAG = "sF" then VALUE end as sF,
    case when TAG = "sG" then VALUE end as sG,
    case when TAG = "sH" then VALUE end as sH,
    case when TAG = "sI" then VALUE end as sI,
    case when TAG = "sK" then VALUE end as sK,
    case when TAG = "sL" then VALUE end as sL,
    case when TAG = "sM" then VALUE end as sM,
    case when TAG = "sN" then VALUE end as sN,
    case when TAG = "sP" then VALUE end as sP,
    case when TAG = "sQ" then VALUE end as sQ,
    case when TAG = "sR" then VALUE end as sR,
    case when TAG = "sS" then VALUE end as sS,
    case when TAG = "sT" then VALUE end as sT,
    case when TAG = "sV" then VALUE end as sV,
    case when TAG = "sW" then VALUE end as sW,
    case when TAG = "sY" then VALUE end as sY,
    case when TAG = "gene" then VALUE end as gene,
    case when TAG = "Organism" then VALUE end as Organism,
    case when TAG = "Note" then VALUE end as Note,
    case when TAG = "load_id" then VALUE end as load_id,
    case when TAG = "perCAST" then VALUE end as perCAST,
    case when TAG = "perSEG" then VALUE end as perSEG
FROM tags
);

```

```
70      CREATE VIEW tags_pivot AS (
71      SELECT id,
72      IFNULL(SUM(cA), 0) AS cA,
73      IFNULL(SUM(cC), 0) AS cC,
74      IFNULL(SUM(cD), 0) AS cD,
75      IFNULL(SUM(cE), 0) AS cE,
76      IFNULL(SUM(cF), 0) AS cF,
77      IFNULL(SUM(cG), 0) AS cG,
78      IFNULL(SUM(cH), 0) AS cH,
79      IFNULL(SUM(cI), 0) AS cI,
80      IFNULL(SUM(cK), 0) AS cK,
81      IFNULL(SUM(cL), 0) AS cL,
82      IFNULL(SUM(cM), 0) AS cM,
83      IFNULL(SUM(cN), 0) AS cN,
84      IFNULL(SUM(cP), 0) AS cP,
85      IFNULL(SUM(cQ), 0) AS cQ,
86      IFNULL(SUM(cR), 0) AS cR,
87      IFNULL(SUM(cS), 0) AS cS,
88      IFNULL(SUM(cT), 0) AS cT,
89      IFNULL(SUM(cV), 0) AS cV,
90      IFNULL(SUM(cW), 0) AS cW,
91      IFNULL(SUM(cY), 0) AS cY,
92      IFNULL(SUM(sA), 0) AS sA,
93      IFNULL(SUM(sC), 0) AS sC,
94      IFNULL(SUM(sD), 0) AS sD,
95      IFNULL(SUM(sE), 0) AS sE,
96      IFNULL(SUM(sF), 0) AS sF,
97      IFNULL(SUM(sG), 0) AS sG,
98      IFNULL(SUM(sH), 0) AS sH,
99      IFNULL(SUM(sI), 0) AS sI,
100     IFNULL(SUM(sK), 0) AS sK,
101     IFNULL(SUM(sL), 0) AS sL,
102     IFNULL(SUM(sM), 0) AS sM,
103     IFNULL(SUM(sN), 0) AS sN,
104     IFNULL(SUM(sP), 0) AS sP,
105     IFNULL(SUM(sQ), 0) AS sQ,
106     IFNULL(SUM(sR), 0) AS sR,
107     IFNULL(SUM(sS), 0) AS sS,
108     IFNULL(SUM(sT), 0) AS sT,
109     IFNULL(SUM(sV), 0) AS sV,
110     IFNULL(SUM(sW), 0) AS sW,
111     IFNULL(SUM(sY), 0) AS sY,
112     IFNULL(MIN(`LENGTH`), 0) AS `length`,
113     IFNULL(MIN(gene), 0) AS gene,
114     IFNULL(MIN(Organism), 0) AS Organism,
115     IFNULL(MIN(Note), 0) AS Note,
116     IFNULL(MIN(load_id), 0) AS load_id,
117     IFNULL(SUM(perCAST), 0) AS perCAST,
118     IFNULL(SUM(perSEG), 0) AS perSEG
119     FROM tags_extended
120     GROUP BY id
121     );
```

Code Snippet 6-2. *The LCR-eXXXplorer_gff.pl script used to transform masked protein sequence files to a GFF3 compliant format.*

```
#!/usr/bin/Perl


    ########################################################
    #--------------  DESCRIPTION  --------------------#
    ########################################################

    # This script prepares proteome data for usage on LCRxxxPlorer
    # Required inputs are:
    # 1) A valid fasta file containing the protein sequences
    # 2) A valid masked version of the previous file
    # 4) A matching .tab file containing statistics for cast


    ########################################################
    #------------------  USAGE  ----------------------#
    ########################################################

    # LCRxxxplorer_gff.pl usage:
    #       -i <input fasta file> | required
    #       -m <input masked fasta file> | required
    #       -t <input tab file > | required
    #       -h help, Displays this message



    ########################################################
    #------------------  AUTHOR  ---------------------#
    ########################################################

    # Ioannis Kirmitzoglou
    # Bioinformatics Research Laboratory, UCY
    # IoannisKirmitzoglou _a|t_ gmail.com



    ########################################################
    #----------------  INCLUDES  ----------------------#
    ########################################################

    use strict;
    use Getopt::Std;
    use Cwd;
    use Bio::Seq;
    use Bio::SeqIO;
    use Text::Wrap;


    my %options=();
    my @myfiles;
    my $usage = "FastaToGff3.pl usage:
    -i <input fasta file> | required
    -m <input masked fasta file> | required
    -t <input tab file > | required
    ";

    my $count = 0;

    ########################################################
    #-------------------- MAIN  ----------------------#
    ########################################################

    unless (getopts("i:m:s:t:", \%options)) {die "$usage\n";} # not used yet, script only takes
    as input the unmasked fasta file and assumes the rest of the files have the appropriate
    extensions
    my $time = time();
    # my @AAs = ("A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R",
    "S", "T", "V", "W", "Y", "X");

```

```
68      # have to exclude X for now as they are not handled correctly if they exist in the original
        sequence
69      my @AAs = ("A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S",
        "T", "V", "W", "Y");
70
71      my %totalSEG_types = (); #store the total number of masked residues per type
72      my %totalCAST_types = ();        #store the total number of masked residues per type
73
74      #initiate the total counts hashes
75      foreach my $AA (@AAs) {
76
77      $totalCAST_types{$AA} = 0;
78      $totalSEG_types{$AA} = 0;
79
80      }
81
82      my $dir = '';
83
84      if (!$options{i}) {      #no
85      print STDERR "Getting directory file listing\n";
86      $dir = cwd;
87      recurse($dir);
88      } else {#yes
89      if ( -d $options{i}) {   # a dir is given
90              print STDERR "Getting directory file listing\n";
91              $dir = $options{i};
92              recurse($dir);
93
94      } else {# a file is given
95              push (@myfiles, $options{i});
96      }
97      }
98
99      my $tot = scalar @myfiles;
100     my $i = 1;
101
102     print STDERR "Going to process $tot FASTA files\n";
103
104     foreach my $file (@myfiles) {
105
106     # my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
107     # my $timestamp = printf "%4d-%02d-%02d
        %02d:%02d:%02d\n",$year+1900,$mon+1,$mday,$hour,$min,$sec;
108     my $timestamp = localtime();
109
110
111     my @IDs = ();    # Store IDs and their order of sequences in the original FASTA file
112     my %CAST_IDs = ();       # Store IDs of sequences masked by CAST, according to the info in
        the .tab file
113     my %SEG_IDs = ();        # Store IDs of sequences masked by SEG
114     my %Org_FASTA = ();      # Store the original FASTA sequence
115     my %Sequence_desc = ();  # Store the sequence description
116     my %Sequence_refid = (); # Store the sequence refid
117     my %Sequence_genename = (); # Store the sequence gene name
118     my %Sequence_organism = (); # Store the sequence organism name
119     my %CAST_FASTA = ();     # Store the masked sequence as it was masked by CAST
120     my %SEG_FASTA = ();      # Store the masked sequence as it was masked by SEG
121     my %Orig_BitMasks = ();
122     my %CAST_BitMasks = ();
123     my %SEG_BitMasks = ();
124     my %CAST_regions = ();
125
126     #create 2 hashes to store AA types that have been masked per sequence and per algorithm
127
128     my %CAST_types = ();
129     my %SEG_types = ();
130
131     $file =~ /(.+)\.faa/s;
132     my $basefilename = $1;
133     open GFF, ">$basefilename.gff3" or die "Unable to create GFF file $file.gff\n";
134
135     print GFF qq
136     {##gff-version 3
137     ##date $timestamp
138     ##source $file
139     };
```

```
140
141
142     ##########################################################################################################
        ########
143     $count = 0;
144     ##########################################################################################################
        ########
145     my $tabfile = $file . ".tab";
146
147     open TAB, "<$tabfile" or die;
148
149     <TAB>;  # skip first line
150     my $previous_id = 0;
151     my $previous_lcr_type = "";
152     my %tab_count= ();
153
154     while (<TAB>) {
155             chomp;
156             # example of the first column in a CAST tab statistics file
157             # >sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate
        Goorha)
158
159             /sp\|(\w+)\|([\w\_]+).+\t(\w)\t(\d+)\t(\d+)\t(\d+)$/i;
160             my ($id, $refid, $lcr_type, $start, $stop, $score)  = ($1, $2, $3, $4, $5, $6);
161             my @regions = ();
162             # print "$id, $refid, $lcr_type, $start, $stop, $score\n";
163             if ($id eq $previous_id) {
164                     # print "*\t$id\t$previous_id\n";
165                     $tab_count{$lcr_type}++;
166                     # if ($previous_lcr_type eq $lcr_type) {
167                             # $tab_count++ ;
168                     # } else {
169                             # $tab_count = 1;
170                     # }
171                     my $gffline = $id;
172                     $gffline .= "\tCAST\tlow_complexity_region_$lcr_type\t";
173                     $gffline .= "$start\t$stop\t$score" ;
174                     # $gffline .= "\t.\t.\tID=CASTRG_" . $lcr_type . "_" .
        $tab_count{$lcr_type} . "_$id" . ";Parent=PR_$id;Name=$id";
175                     $gffline .= "\t.\t.\tID=CASTRG_" . $lcr_type . "_" . $tab_count{$lcr_type}
        . "_$id";
176                     # $gffline .= ";Name=$id" . " - CASTRG." . $lcr_type . "_" .
        $tab_count{$lcr_type} . ";Alias=$refid";
177                     # $gffline .= " - CASTRG." . $lcr_type . "_" . $tab_count{$lcr_type};
178                     $gffline .= ";Note=CAST detected $lcr_type low complexity region" . " for
        sequence $id;LCR_type=$lcr_type\n";
179                     push (@{$CAST_IDs{$id}}, $gffline );
180                     # $CAST_types{$id}{$lcr_type}++;
181                     $previous_id = $id;
182
183
184             } else {
185                     # print STDERR "TAB RESET\n";
186                     %tab_count = ();
187                     $tab_count{$lcr_type}++ ;
188                     # print "#\t$id\t$previous_id\n";
189                     $CAST_IDs{$id} = \@regions;
190                     my $gffline = $id;
191                     $gffline .= "\tCAST\tlow_complexity_region_$lcr_type\t";
192                     $gffline .= "$start\t$stop\t$score" ;
193                     $gffline .= "\t.\t.\tID=CASTRG_" . $lcr_type . "_" . $tab_count{$lcr_type}
        . "_$id";
194                     # $gffline .= ";Name=$id" . " - CASTRG." . $lcr_type . "_" .
        $tab_count{$lcr_type} . ";Alias=$refid";
195                     # $gffline .= " - CASTRG." . $lcr_type . "_" . $tab_count{$lcr_type};
196                     $gffline .= ";Note=CAST detected $lcr_type low complexity region" . " for
        sequence $refid;LCR_type=$lcr_type\n";
197                     push (@{$CAST_IDs{$id}}, $gffline );
198                     $previous_id = $id;
199
200                     # initiate the CAST_types hash
201
202                     foreach my $t (@AAs) {
203                             $CAST_types{$id}{$t} = 0;
204
205                     }
```

```
206
207                     # $CAST_types{$id}{$lcr_type}++;
208
209             }
210             # $CAST_IDs{$id} = 1;    # store id's of sequences with detected LCRs
211
212
213     }
214
215     close TAB;
216
217
218     ##############################################################################################
        ########
219
220     my $castfile = $file . ".cast";
221     print $castfile;
222
223     my $cseqin = Bio::SeqIO->new(-format =>"fasta", -file=> $castfile, -alphabet =>'protein')
        or die "Invalid input file\n";
224     my $ctotalSeq = 0;
225
226     while((my $seqobj = $cseqin->next_seq())) {
227             my $id = $seqobj->display_id;
228             my $sequence = $seqobj->seq();
229
230
231             # >sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate
        Goorha)
232
233             $id =~ /sp\|(\w+)\|([\w\_]+)/i;
234             $id = $1;
235             if (defined($CAST_IDs{$id})){
236             # print "CAST $id\n";
237                     $CAST_BitMasks{$id} = CreateMask($sequence);
238                     # my $count = 0;
239                     $ctotalSeq++;
240             }
241
242             $count ++;
243             # last if ($count >500);
244             # print "$id\n";
245             print STDERR "Processed $ctotalSeq so far...\r";
246     }
247
248     $count = 0;
249
250     print STDERR "Done reading $ctotalSeq masked sequences from the CAST fasta file\n" ;
251     #close OFILE;
252
253     $cseqin = "";
254
255     ##############################################################################################
        ########
256     #Read the SEG masked FASTA file
257     my $sseqin = Bio::SeqIO->new(-format =>"fasta", -file=> "$file.seg", -alphabet =>'protein')
        or die "Invalid input file\n";
258     my $stotalSeq = 0;
259
260     while((my $seqobj = $sseqin->next_seq())) {
261             my $id = $seqobj->display_id;
262
263
264             # >sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate
        Goorha)
265
266             $id =~ /sp\|(\w+)\|([\w\_]+)/i;
267             $id = $1;
268             my $sequence = $seqobj->seq();
269             if (HasXs($sequence)) {
270                     # initiate the SEG_types hash
271
272                     foreach my $t (@AAs) {
273                             $SEG_types{$id}{$t} = 0;
274
275                     }
```

```
276                     $SEG_IDs{$id} = 1;
277                     $SEG_BitMasks{$id} = CreateMask($sequence);
278                     my $count = 0;
279                     $stotalSeq++;
280
281             }
282
283             $count ++;
284             # last if ($count >500);
285             # print "$id\n";
286             print STDERR "Processed $stotalSeq so far...\r";
287     }
288     $count = 0;
289
290     print STDERR "Done reading $stotalSeq masked sequences from the SEG fasta file\n" ;
291
292     $sseqin = "";
293
294
295     ##########################################################################################
        ########
296     #Read the original FASTA file
297     my $oseqin = Bio::SeqIO->new(-format =>"fasta", -file=> $file, -alphabet =>'protein') or
        die "Invalid input file\n";
298     my $ototalSeq = 0;
299
300     while((my $seqobj = $oseqin->next_seq())) {
301             my $id = $seqobj->display_id;
302             my $desc = $seqobj->description;
303             my $sequence = $seqobj->seq();
304
305             # >sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate
        Goorha)
306
307             $id =~ /sp\|(\w+)\|([\w\_]+)/i;
308             $id = $1;
309             my $refid = $2;
310
311             my @desc = split(/\t/, $desc);
312
313             my $organism = $1 if ($desc =~ /OS=(.+?)\s[G|P][N|E]=/);
314
315             $organism =~ s/\%/\%25/g;
316             # $organism =~ s/\ /\%20/g;
317             $organism =~ s/\;/\%3B/g;
318             $organism =~ s/\=/\%3D/g;
319             $organism =~ s/\,/\%2C/g;
320
321             $Sequence_organism{$id} = $organism;
322
323             my $Sequence_desc = $1 if ($desc =~ /(.+)\sOS=/);
324             # URL escape the sequence description
325             $Sequence_desc =~ s/\%/\%25/g;
326             # $Sequence_desc =~ s/\ /\%20/g;
327             $Sequence_desc =~ s/\;/\%3B/g;
328             $Sequence_desc =~ s/\=/\%3D/g;
329             $Sequence_desc =~ s/\,/\%2C/g;
330
331             # Store the sequence description
332             $Sequence_desc{$id} = $Sequence_desc;
333
334             # Store the gene name
335             if ($desc =~ /GN=(.+)\sPE=/) {
336                     $Sequence_genename{$id} = $1;
337
338             } else {
339
340                     $Sequence_genename{$id} = 'missing';
341
342             }
343
344
345             if (defined($CAST_IDs{$id}) || defined($SEG_IDs{$id}) ){
346             # if (defined($SEG_IDs{$id}) ){
347                     # print "$id C:$CAST_IDs{$id} S:$SEG_IDs{$id}\n";
348                     push (@IDs, $id);
```

```
349                     $Org_FASTA{$id} = $sequence;
350                     # $Sequence_desc{$id} = $desc;
351                     $Sequence_refid{$id} = $refid;
352                     $Orig_BitMasks{$id} = CreateMask($sequence);
353                     my $count = 0;
354                     $ototalSeq++;
355             }
356
357             # $count ++;
358             # last if ($count >00);
359             # print "$id\n";
360             print STDERR "Processed $ototalSeq so far...\n";
361     }
362
363     $count = 0;
364     print STDERR "Done reading $ototalSeq masked sequences from the original fasta file\n" ;
365
366     $oseqin = "";
367
368     print GFF "#proteins-number ". scalar(@IDs) . "\n";
369
370     foreach my $sid (@IDs) {
371             # my %CAST_Mask = %{$CAST_BitMasks{$sid}};
372             # my %Orig_Mask = %{$Orig_BitMasks{$sid}};
373             # my %SEG_Mask = %{$SEG_BitMasks{$sid}};
374
375             # track the total score of Masks comparison so we can calculate % masked for the
    original sequence
376             my $perCAST  = 0;
377             my $perSEG   = 0;
378
379             my $mgffline = "";
380
381             my $ogffline = $sid;
382             $ogffline .= "\tUnmasked\tprotein\t1\t";
383             $ogffline .= length($Org_FASTA{$sid}) ;
384             $ogffline .= "\t.\t.\t.\tID=PR_" . $sid .
    ";Name=$sid;gene=$Sequence_genename{$sid};";
385             $ogffline .=
    "Alias=$Sequence_genename{$sid},$Sequence_refid{$sid};Note=$Sequence_desc{$sid};Organism=$S
    equence_organism{$sid};";
386
387             $count++;
388
389             if (defined(%{$SEG_BitMasks{$sid}}))
390             {
391                     my %Orig_Mask = %{$Orig_BitMasks{$sid}};
392                     my %SEG_Mask  = %{$SEG_BitMasks{$sid}};
393
394                     foreach my $AA (@AAs) {
395
396                             my ($diff, $score) = split (/\|/, CompareMasks ($SEG_Mask{$AA},
    $Orig_Mask{$AA}));
397                             next if ($score == 0);
398                             $perSEG += $score;
399
400                             # Locate starting and ending positions of character runs in the
    final vector
401                             # and store them inside two array refs
402                             my ($pos_start_ref, $pos_stop_ref) = FindPositions($diff);
403                             my @pos_start = @$pos_start_ref;
404                             my @pos_stop = @$pos_stop_ref;
405
406                                     # Process character runs of the final vector
407                                     for (my $x = 0; $x<=$#pos_start; $x++) {
408
409                                             # Create an id of the character run
410                                             my $lcr_id = $x+1;
411
412
413                                             # Print info
414                                             my $gffline = $sid;
415                                             $gffline .= "\tSEG\tmasked_region_$AA\t";
416                                             $gffline .= "$pos_start[$x]\t$pos_stop[$x]" ;
417                                             # $gffline .= "\t.\t.\t.\tID=SEG_" . $AA . "_" .
    $sid . ";Parent=PR_$sid;Name=$sid";
```

```
418                                         $gffline .= "\t.\t.\t.\tID=SEG_" . $AA . "_" .
    $sid;
419                                         # $gffline .= ";Name=$sid" . " - SEG." . $AA .
    ";Alias=$Sequence_refid{$sid}";
420                                         # $gffline .= " - SEG." . $AA;
421                                         if ($AA eq 'X') {       # Unmasked sequence had Xs
422                                                 $gffline .= ";Note=Ambiguous resides in
    the original unmasked sequence;LCR_type=$AA\n";
423
424                                         } else {
425                                                 $gffline .= ";Note=SEG Masked Residues of
    type $AA" . " for sequence $sid;LCR_type=$AA\n";
426                                                 $SEG_types{$sid}{$AA}++;
427                                                 $totalSEG_types{$AA}++;
428                                         }
429
430                                         $mgffline .= $gffline;

431
432                                         # my $gffline = $Sequence_refid{$sid};
433                                         # $gffline .=
    "\tSEG\tlow_complexity_region_$AA\t";
434                                         # $gffline .= "$pos_start[$x]\t$pos_stop[$x]" ;
435                                         # $gffline .= "\t.\t.\t.\tID=SEG_" . $AA .
    "_$lcr_id" . "_$sid" . ";Name=$Sequence_refid{$sid}";
436                                         # $gffline .= " - SEG." . $AA . "_$lcr_id" .
    "_$sid;";
437                                         # $gffline .= "Note=SEG Masked Residues of type
    $AA" . " for sequence $Sequence_refid{$sid};LCR_type=$AA\n";
438                                         # print GFF $gffline;
439
440                                 }
441                         }
442
443                 }
444
445                 if (defined(%{$CAST_BitMasks{$sid}}))
446                 {
447                         my %Orig_Mask = %{$Orig_BitMasks{$sid}};
448                         my %CAST_Mask = %{$CAST_BitMasks{$sid}};
449
450                         foreach my $AA (@AAs) {
451
452                                 my ($diff, $score) = split (/\|/, CompareMasks ($CAST_Mask{$AA},
    $Orig_Mask{$AA}));
453                                 next if ($score == 0);
454
455                                 # track the total score so we can calculate % masked
456                                 $perCAST += $score;
457
458                                 # Locate starting and ending positions of character runs in the
    final vector
459                                 # and store them inside two array refs
460                                 my ($pos_start_ref, $pos_stop_ref) = FindPositions($diff);
461                                 my @pos_start = @$pos_start_ref;
462                                 my @pos_stop = @$pos_stop_ref;
463
464                                         # Process character runs of the final vector
465                                         for (my $x = 0; $x<=$#pos_start; $x++) {
466
467                                                 # Create an id of the character run
468                                                 my $lcr_id = $x+1;
469
470                                                 # Print info
471                                                 my $gffline = $sid;
472                                                 $gffline .= "\tCAST\tmasked_region_$AA\t";
473                                                 $gffline .= "$pos_start[$x]\t$pos_stop[$x]" ;
474                                                 # $gffline .= "\t.\t.\t.\tID=CAST_" . $AA . "_" .
    $sid . ";Parent=PR_$sid;Name=$sid";
475                                                 $gffline .= "\t.\t.\t.\tID=CAST_" . $AA . "_" .
    $sid;
476                                                 # $gffline .= ";Name=$sid" . " - CAST." . $AA .
    ";Alias=$Sequence_refid{$sid}";
477                                                 # $gffline .= " - CAST." . $AA;
478
479                                                 if ($AA eq 'X') {
```

```
480                                                     $gffline .= ";Note=CAST Masked Residues
    for sequence $Sequence_refid{$sid};LCR_type=$AA\n";
481                                             } else {
482                                                     $gffline .= "Note=CAST Masked Residues
    ($AA)" . " for sequence $sid;LCR_type=$AA\n";
483                                                     $CAST_types{$sid}{$AA}++;
484                                                     $totalCAST_types{$AA}++;
485
486                                             }
487
488                                             $mgffline .= $gffline;

489
490
491
492                                     }
493                             }
494
495             }
496
497
498             $perCAST = sprintf("%.3f", ($perCAST/length($Org_FASTA{$sid})) * 100);
499             $perSEG = sprintf("%.3f", ($perSEG/length($Org_FASTA{$sid})) * 100);
500
501             $ogffline .= "perCAST=$perCAST;perSEG=$perSEG;";
502
503             foreach my $AA (@AAs) {
504                     my $c = 0+$CAST_types{$sid}{$AA};
505                     my $s = 0+$SEG_types{$sid}{$AA};
506
507                     $ogffline .= "s$AA=$s;";
508                     $ogffline .= "c$AA=$c;";
509
510
511             }
512
513             print GFF "$ogffline\n";
514             print GFF "$mgffline" ;
515
516             if (defined($CAST_IDs{$sid}))
517             {
518                     my @regions = @{$CAST_IDs{$sid}};
519
520                             foreach my $region (@regions) {
521                                     # print "$region";
522                                     print GFF "$region";
523
524                             }
525
526             }
527             # last if ($count > 200);
528             # foreach my $AA (@AAs) {
529
530                     # my $diff = CompareMasks ($CAST_Mask{$AA}, $Orig_Mask{$AA});
531                     # print ">$sid\t$AA\n";
532                     # print "$Org_FASTA{$sid}\n";
533                     # print "$CAST_FASTA{$sid}\n";
534                     # print "$diff\n";
535
536             # }
537
538     }
539
540     print GFF "\n\n##FASTA\n";
541     $count = 0;
542
543
544
545
546
547
548
549
550     foreach my $sid (@IDs) {
551
552             my $FASTA = fixedwidth_linebreaks( $Org_FASTA{$sid}, 60 );
```

```
553
554             print GFF ">$sid\n$FASTA";
555             $count++;
556             # last if ($count > 200);
557
558
559     }
560
561     # Print total statistics for file
562     print "Statistics for file $file\n";
563     print "---SEG---\n";
564
565     foreach my $AA (@AAs) {
566
567             print "$AA\t$totalSEG_types{$AA}\n";
568
569     }
570
571     print "---CAST---\n";
572
573     foreach my $AA (@AAs) {
574
575             print "$AA\t$totalCAST_types{$AA}\n";
576
577     }
578
579
580     }
581
582     my $dur = time() - $time;
583     print STDERR "Fewww, that was FAST\nIt only took me $dur seconds...\n";
584
585
586
587     ########################################################
588     #---------------  FUNCTIONS   --------------------#
589     ########################################################
590
591     sub recurse($) { # scan working dir and create an array with the *m8 files it contains
592       my($path) = @_;
593
594       ## append a trailing / if it's not there
595       $path .= '/' if($path !~ /\/$/);
596
597       ## print the directory being searched
598       #print $path,"\n";
599
600       ## loop through the files contained in the directory
601       for my $eachFile (glob($path.'*faa')) {
602
603         ## if the file is a directory
604         if( -d $eachFile) {
605           ## pass the directory to the routine ( recursion )
606           next;
607         } else {
608
609           ## print the file ... tabbed for readability
610           #print "$eachFile\n";
611         push (@myfiles, $eachFile);
612           }
613       }
614     }
615
616     sub HasXs { #Returns a binary vector mask (1 if $char matches character in current
       location, otherwise 0)
617     my $seq = shift;
618     # my ($seq, $char) = @_;
619
620                     while ($seq =~ /(.)/g) { # process sequence one character at a time
621                             if ($1=~/x/i) {
622                                     return 1;
623                             }
624                     }
625     }
626
627
```

```perl
sub CreateMask { #Returns a binary vector mask (1 if $char matches character in current location, otherwise 0)
my $seq = shift;
my %masks = ();
# my ($seq, $char) = @_;
foreach my $AA (@AAs) {
        my $mask = "";

                while ($seq =~ /(.)/g) { # process sequence one character at a time
                        if ($1=~/$AA/i) {
                                $mask.="1";
                        } else {
                                $mask.="0";
                        }
                }

        $masks{$AA} = $mask;

}
return \%masks;
}

# Compares to masks and retuns a final vector containing all the positions of the character
# in the modified mask minus the ones in the original mask
sub CompareMasks {

my ($orig_mask, $mod_mask) = @_;
my @orig_mask = split(//, $orig_mask);
my @mod_mask = split(//, $mod_mask);
my $final_mask = "";
my $score = 0;

for (my $i=0; $i<=$#orig_mask; $i++) {
        # Calculate the position's value
        my $x = $mod_mask[$i] - $orig_mask[$i];
        # and append it to the final vector
        $final_mask .= abs($x);
        $score += abs($x);
        }

return "$final_mask|$score";

}

# This sub takes a mask an input and returns locates start and stop positions of substrings containing 1
sub FindPositions {
my $mask = $_[0];
my $i = 1;
my $flag = 0;
my @pos_start = ();
my @pos_stop = ();
my $len = length($mask);

while ($mask =~ /(.)/g) { # Process the mask character by character
        if ($1) { # Character is 1
                if ($flag) { # The start of the run is in a previous position

                        # If this is the last character of the mask
                        # the store the stop position of the run
                        push (@pos_stop, $i) if ($i == $len);
                        $i++;

                } else { # The run starts at this position
                        # Store the start position
                        push (@pos_start, $i);
                        # Flag the start of the run
                        $flag = 1;

                        # If this is the last character of the mask
                        # the store the stop position of the run
                        push (@pos_stop, $i) if ($i == $len);
                        $i++;
                }

```

```
            } else { # Character is 0
                    if ($flag) { # The start of the run is in a previous position
                            # Store the stop position
                            push (@pos_stop, $i-1);
                            $flag = 0;
                            $i++;
                    } else {
                            $i++;
                    }
            }
    }

    # Return the refs to the arrays containing start and stop positions
    return (\@pos_start, \@pos_stop);

    }

    sub fixedwidth_linebreaks
    {
        my( $str, $width ) = @_;
        my $newstr;
        while ( $str )
        {
            $newstr .= substr( $str, 0, $width, '' )."\n";
        }
        return $newstr;
    }
```

Code Snippet 6-3. *The "SubmitBLAST" GBrowse plugin is a "dumber" plugin that allows the user to apply any combination of LCR annotations as a hard or soft mask to any part of a sequence stored in LCR-eXXXplorer. The user can choose to invert the mask, thus essentially eliminating all the unbiased regions of the sequence, and fine-tune the BLASTP parameters prior to submitting a BLASTP job to a chosen web-server (currently it only supports NCBI BLAST).*

```
1     package Bio::Graphics::Browser2::Plugin::SubmitBLAST;
2
3      use strict;
4      use Bio::Graphics::Browser2::Plugin;
5      use Bio::Graphics::Browser2::Markup;
6      use CGI qw/standard escape unescape/;
7      use CGI 'html3';
8      use Bio::Graphics::Browser2::Util;
9      use Bio::Graphics::FeatureFile;
10     use CGI qw(:standard *pre);
11     use Data::Dumper;
12     use constant DEBUG => 0;
13     use vars qw($VERSION @ISA);
14
15     =head1 NAME
16
17     Bio::Graphics::Browser2::Plugin::SubmitBLAST -- a plugin that applies hard and/or
18     soft masks to user selectable tracks and submits the final sequence to the Web BLAST
19
20     =head1 SYNOPSIS
21
22
23
24     =head1 DESCRIPTION
25
26
27
28     =cut
29
30
31     $VERSION = '0.01';
32
33     @ISA = qw(Bio::Graphics::Browser2::Plugin);
34
35
36     sub verb {' '}
37     sub name { "Submit to NCBI BLASTP" }
38     sub mime_type {'text/html'}
39
40     sub description {
41       p("This plugin was written by Ioannis Kirmitzoglou.");
42     }
43
44     sub config_defaults {
45       my $self = shift;
46       return {    'FILTER'                        => 0,
47                   'cmd'                           => 'web',
48                   'PAGE'                          => 'proteins',
49                   'FILTER_LOOKUP'                 => 0,
50                   'LCASE_MASK'                    => 1,
51                   'COMPOSITION_BASED_STATISTICS'  => 0,
52                   'INVERT'                        => 0,
53              };
54     }
55
56     sub reconfigure {
57         my $self = shift;
58         my $current = $self->configuration;
59
60         foreach my $param ( $self->config_param() ) {
61             warn "param = $param\n" if DEBUG;
62             my $value = $self->config_param($param) or next;
63             $current->{$param} = $value;
64             warn "current{$param} = $current->{$param}\n" if DEBUG;
65         }
66     }
```

```
67
68      sub configure_form {
69        my $self = shift;
70        my $current_config = $self->configuration;
71          my %cbs_labels = ( 0, 'No adjustment',
72                             1, 'Composition-based statistics',
73                             2, 'Conditional compositional score matrix adjustment',
74                             3, 'Universal compositional score matrix adjustment');
75
76          my @choices = ();
77
78          my $browser = $self->browser_config();
79          my @labels;
80          foreach ( $browser->labels() ) {
81          push @labels, $_ unless ! defined $browser->setting($_,'feature');
82          }
83
84          autoEscape(0);
85          my %selected = map {$_=>1} ($self->selected_tracks);
86
87          foreach my $featuretype ( @labels ) {
88              next if ! $selected{$featuretype};
89          my $realtext = $browser->setting($featuretype,'key') || $featuretype;
90          push @choices, TR({-class => 'searchtitle'},
91                               th({-align=>'RIGHT',-width=>'25%'},
92                                   $realtext,
93                               td(join (' ',
94                                   checkbox(
95                                       -name    => $self->config_name($featuretype),
96                                       -checked => $current_config->{$featuretype} || 0,
97                                       -label   => ''
98                                       ),
99                                   )
100                              )
101                              )
102                         );
103         }
104
105         push @choices, TR({-class => 'searchtitle'},
106                              th({-align=>'RIGHT',-width=>'25%'},
107                                  'Invert Mask',
108                              td(join (' ',
109                                  checkbox(
110                                      -name    => $self->config_name('INVERT'),
111                                      -checked => $current_config->{'INVERT'},
112                                      -label   => ''
113                                      ),
114                                  )
115                              )
116                              )
117                           );
118         my $html= table({-width=>'100%'},@choices);
119
120
121     my $form = $html . h3("Optimal important BLASTP values have been selected for you").
122       table({-border => 0, -width=>'100%'},
123           TR({-class => 'searchtitle'}, [
124             th({-align=>'RIGHT',-width=>'25%'},"Filtering of low complexity regions ",
125                   td(checkbox(
126                       -name    => $self->config_name('FILTER'),
127                       -label   =>'',
128                       -value   =>'L',
129                       -checked =>$current_config->{'FILTER'}
130                       ),
131                       "<a href='http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml#filter'
    target=_blank> <img src='/gbrowse2/images/buttons/query.png'> </a>",
132                   ),
133               ),
134           ]),
135           TR({-class => 'searchtitle'}, [
136             th({-align=>'RIGHT',-width=>'25%'},"Mask lower case letters ",
137                   td(checkbox(
138                       -name    => $self->config_name('LCASE_MASK'),
139                       -label   => '',
140                       -value   => 'yes',
141                       -checked => $current_config->{'LCASE_MASK'}
```

```
                    ),
                    "<a href='http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml#filter'
target=_blank> <img src='/gbrowse2/images/buttons/query.png'> </a>",
                ),
            ),
        ]),
        TR({-class => 'searchtitle'}, [
          th({-align=>'RIGHT',-width=>'25%'},"Mask for lookup table only ",
                td(checkbox(
                    -name    => $self->config_name('FILTER_LOOKUP'),
                    -label   => '',
                    -value   => 'm',
                    -checked => $current_config->{'FILTER_LOOKUP'}
                    ),
                    "<a href='http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml#filter'
target=_blank> <img src='/gbrowse2/images/buttons/query.png'> </a>",
                ),
            ),
        ]),
        TR({-class => 'searchtitle'}, [
          th({-align=>'RIGHT',-width=>'25%'},"Compositional Adjustments ",
                td(popup_menu(
                    -name    => $self->config_name('COMPOSITION_BASED_STATISTICS'),
                    -values  => [0,1,2,3],
                    -labels  => \%cbs_labels,
                    -default => $current_config->{'COMPOSITION_BASED_STATISTICS'}
                    ),
                    "<a
href='http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml#compositional_adjustments'
target=_blank> <img src='/gbrowse2/images/buttons/query.png'> </a>",
                ),
            ),
        ])
    );
    autoEscape(1);
    return $form;
}

sub dump {
    my $self = shift;
    my $segment = shift;

    unless ($segment) {
      print "No sequence specified.\n";
      exit 0;
    }

    my $config  = $self->configuration;
    my $style = !$config->{'INVERT'} ? 'LOWERCASE' : 'UPPERCASE';
    my $protein = !$config->{'INVERT'} ? uc $segment->dna : lc $segment->dna;

    my $browser = $self->browser_config();
    warn("====== beginning dump =====\n") if DEBUG;
    warn "length of protein = ",length($protein) if DEBUG;

    my %types;

    my $markup = Bio::Graphics::Browser2::Markup->new;

    while( my ($type,$val) = each %{$config} ) {

        next unless $val;

        warn "configuring $type => $val\n" if DEBUG;

        (my $feature_type = $type) =~ s/^[^.]+\.//;
        # there may be several feature types defined for each track
        my @types = $browser->label2type($feature_type) or next;
        for my $t (@types) {
        $markup->add_style($t => $style);
        warn "adding style $t => $style\n" if DEBUG
        }

        foreach (@types) { $types{$_}++ };
    }

```

```
214         my @regions_to_markup = $self->make_markup($segment,[keys %types],$markup) if %types;
215
216         # add a newline every 60 positions
217         $markup->add_style('newline',"\n");
218         push @regions_to_markup,map {['newline',60*$_]} (1..length($protein)/60);
219         $markup->markup(\$protein,\@regions_to_markup);
220
221         my $label = "$segment";
222
223         warn("====== end of dump =====\n") if DEBUG;
224
225           my $url       = 'http://blast.ncbi.nlm.nih.gov/Blast.cgi';
226
227           # Whether to print a confirmation page before external submission
228           my $confirm = 1;
229
230           # Format the display sequence as fasta
231           my $fasta = ">$label\n$protein\n";
232
233
234           unless ($url =~ /^http/i) {
235             $url = "http://$url";
236           }
237
238           # pass-thru arguments -- to be sent to the extertnal web-site
239           my %args;
240           for my $arg (keys %$config) {
241             next if $arg =~ /^label$|^confirm$|^url$|^fasta$/;
242             $args{$arg} = unescape($config->{$arg});
243           }
244
245           print start_form(-name=>'f1', -method=>'POST', -action=>$url), "\n";
246           for my $arg (keys %args) {
247             next if $arg =~ /FILTER_LOOKUP|SEG|CAST|MASKED|INVERT/i;
248             print hidden($arg => $args{$arg}), "\n";
249           }
250           print hidden ('FILTER' => $args{'FILTER_LOOKUP'});
251           print hidden('QUERY' => $fasta);
252           print hidden('JOB_TITLE' => $label);
253           print hidden('CMD' => 'web');
254           print hidden('PAGE' => 'proteins');
255
256           if ($confirm) {
257             my @rows = th({-colspan => 2, -style => "background:lightsteelblue"},
258                   b("The following data will be submitted to $url"),
259                   p(submit(-name => 'Confirm'),'  ',
260                    button(-name => 'Cancel', -onclick => 'javascript:window.close()')));
261
262             for my $arg (keys %args) {
263               next if $arg eq $label;
264               push @rows, td({-width => 100, -style => 'background:lightyellow'},
265                     [b("$arg:"), unescape($args{$arg})]);
266             }
267
268
269             push @rows, td({-width => 100, -style => 'background:lightyellow'},
270                   [b($label), pre($fasta)]);
271
272             print div({-id=>'confirm_blast',
273                         -style => " position:relative;
274                                     z-index: 100;
275                                     width:400px;
276                                     height:400px;
277                                     margin-left:auto;
278                                     margin-right:auto;"},
279
280                         table({-border=> 1}, Tr({-valign => 'top'}, \@rows))
281             )
282           }
283
284           print end_form;
285
286           unless ($confirm) {
287             print qq(<script type="text/javascript">document.f1.submit();</script>);
288           }
289     }
```

```
290
291     sub make_markup {
292       my $self = shift;
293       my ($segment,$types,$markup) = @_;
294
295       my @regions_to_markup;
296
297       warn("segment length is ".$segment->length()."\n") if DEBUG;
298       my $iterator = $segment->get_seq_stream(-types=>$types,
299                               -automerge=>1) or return;
300       my $segment_start = $segment->start;
301       my $segment_end   = $segment->end;
302       my $segment_length = $segment->length;
303
304       while (my $markupregion = $iterator->next_seq) {
305
306         warn "got feature $markupregion\n" if DEBUG;
307         #THIS NEEDS FIXING AS IT CREATES SEGMENTS WITH START>END??? INVESTIGATE THIS
    IKIRMITZ
308         # handle both sub seqfeatures and split locations...
309         # somebody rescue me from this insanity!
310         my @parts = eval { $markupregion->sub_SeqFeature } ;
311         @parts = eval { my $id   = $markupregion->location->seq_id;
312                 my @subs = $markupregion->location->sub_Location;
313                 grep {$id eq $_->seq_id} @subs } unless @parts;
314         @parts = ($markupregion) unless @parts;
315
316         for my $p (@parts) {
317           my $start = $p->start - $segment_start;
318           my $end   = $start + $p->length;
319
320           $start++ if $p->strand < 0;
321
322           warn("$p ". $p->location->to_FTstring() . " type is ".$p->primary_tag) if DEBUG;
323           $start = 0                  if $start < 0;  # this can happen
324           $end   = $segment->length   if $end > $segment->length;
325           warn "annotating $p $start..$end" if DEBUG;
326
327           my $style_symbol;
328           foreach ($p->type,$p->method,$markupregion->type,$markupregion->method) {
329         $style_symbol ||= $markup->valid_symbol($_) ? $_ : undef;
330           }
331           warn "style symbol for $p is $style_symbol, and style is ",$markup-
    >style($style_symbol),"\n" if DEBUG;
332           next unless $style_symbol;
333
334           warn "[$style_symbol,$start,$end]\n" if DEBUG;
335           push @regions_to_markup,[$style_symbol,$start,$end];
336         }
337       }
338       @regions_to_markup;
339     }
340
341     1;
342
```

Code Snippet 6-4. *The "AdvancedSearch" plugin source code of LCR-eXXXplorer.*

```
1   package Bio::Graphics::Browser2::Plugin::AdvancedFind;
2
3   use strict;
4   use warnings;
5   use base 'Bio::Graphics::Browser2::Plugin';
6   use Bio::Graphics::GBrowseFeature;
7   use CGI qw(:standard *table :html3);
8   use DBI;
9   use SQL::Abstract::Limit;
10
11
12  use vars '$VERSION','@ISA';
13
14  $VERSION = '0.01';
15
16  sub name { "Advanced Search" }
17
18  sub verb {' '};
19
20  sub description {
21      my $text;
22    $text .= p("Use this form to search LCRs eXXXplorer using a variety of search
    conditions.");
23    $text .= p("The displayed results match all conditions and are limited to 500.");
24    return $text;
25  }
26
27  sub type { 'finder' }
28
29  sub config_defaults {
30    my $self = shift;
31    return {};
32  }
33
34  sub configure_form {
35    my $self = shift;
36    my $current_config = $self->configuration;
37
38    my $form .= hidden(-name=>'plugin_find',-default=>['']);
39
40
41    $form .= h3("Advanced Search:");
42
43    my @cast_types = ("cN", "cA", "cR", "cD", "cC", "cE", "cQ", "cG", "cH", "cI", "cL", "cK",
    "cM", "cF", "cP", "cS", "cT", "cW", "cY", "cV");
44    my @seg_types  = ("sN", "sA", "sR", "sD", "sC", "sE", "sQ", "sG", "sH", "sI", "sL", "sK",
    "sM", "sF", "sP", "sS", "sT", "sW", "sY", "sV");
45    my %cast_labels = (
46          cN => 'Asparagine',
47          cA => 'Alanine',
48          cR => 'Arginine',
49          cD => 'Aspartic acid',
50          cC => 'Cysteine',
51          cE => 'Glutamic acid',
52          cQ => 'Glutamine',
53          cG => 'Glycine',
54          cH => 'Histidine',
55          cI => 'Isoleucine',
56          cL => 'Leucine',
57          cK => 'Lysine',
58          cM => 'Methionine',
59          cF => 'Phenylalanine',
60          cP => 'Proline',
61          cS => 'Serine',
62          cT => 'Threonine',
63          cW => 'Tryptophan',
64          cY => 'Tyrosine',
65          cV => 'Valine',
66    );
67
68    my %seg_labels = (
```

```
            sN => 'Asparagine',
            sA => 'Alanine',
            sR => 'Arginine',
            sD => 'Aspartic acid',
            sC => 'Cysteine',
            sE => 'Glutamic acid',
            sQ => 'Glutamine',
            sG => 'Glycine',
            sH => 'Histidine',
            sI => 'Isoleucine',
            sL => 'Leucine',
            sK => 'Lysine',
            sM => 'Methionine',
            sF => 'Phenylalanine',
            sP => 'Proline',
            sS => 'Serine',
            sT => 'Threonine',
            sW => 'Tryptophan',
            sY => 'Tyrosine',
            sV => 'Valine',
      );

      my $table = table({-ID => 'SearchTable', -class=>"detailstable", -border => 0, -align=>'center'},
                  TR({-class=>'dtblrows'},
                    td({-nowrap=>"nowrap", -class=>"dtblheader", -width=>'150px'}, "Accession:"),
                    td({-colspan=>"2",-width=>'250px'}, textfield(-align=>'center', -type=>'text', -id=>'fl_accession', -pattern=>'^[A-Z0-9]{6}$', -name=>$self->config_name('accession'),-size=>36), span('')),
                    td({-width=>'250px'}, span("Must be six characters long, containing only capital letters and digits")),
                    ),
                  TR({-class=>'dtblrows'},
                    td({-nowrap=>"nowrap", -class=>"dtblheader"}, "Gene name:"),
                    td({-colspan=>"2"}, textfield(-align=>'center', -type=>'text', -id=>'fl_gene', -pattern=>'^[\w\-]+$', -name=>$self->config_name('gene'),-size=>36), span('')),
                    td({-width=>'250px'}, span("Must contain only letters, digits and '-'")),
                    ),
                  TR({-class=>'dtblrows'},
                    td({-nowrap=>"nowrap", -class=>"dtblheader"}, "Protein name"),
                    td({-colspan=>"2"}, textfield(-align=>'center', -type=>'text', -id=>'fl_name', -pattern=>'^.+$', -name=>$self->config_name('name'),-size=>36), span('')),
                    td({-width=>'250px'}, span("Can contain any character, including SQL wildcards (e.g. %)")),
                    ),
                  TR({-class=>'dtblrows'},
                    td({-nowrap=>"nowrap", -class=>"dtblheader"}, "Organism"),
                    td({-colspan=>"2"}, textfield(-align=>'center', -type=>'text', -id=>'fl_organism', -pattern=>'^.+$', -name=>$self->config_name('organism'),-size=>36), span('')),
                    td({-width=>'250px'}, span("Can contain any character, including SQL wildcards (e.g. %)")),
                    ),
                  TR({-class=>'dtblrows'},
                    td(),
                  ),
                  TR({-class=>'dtblrows'},
                    td({-nowrap=>"nowrap", -class=>"dtblheader"}, "Length"),
                    td("from",textfield(-align=>'left', -type=>'text', -id=>'fl_length_from', -pattern=>'^\d+$', -name=>$self->config_name('length_from'),-size=>8), span('')),
                    td("to",textfield(-align=>'left', -type=>'text', -id=>'fl_length_to', -pattern=>'^\d+$', -name=>$self->config_name('length_to'),-size=>8), span('')),
                    td({-width=>'250px'}, span("Must be an integer")),
                    ),
                  TR({-class=>'dtblrows'},
                    td(),
                  ),
                  TR({-class=>'dtblrows'},
                    td({-nowrap=>"nowrap", -class=>"dtblheader"}, "% of sequence length masked by CAST"),
                    td("from",textfield(-align=>'left', -type=>'text', -id=>'fl_cast_from', -pattern=>'^[\d\.]+$', -name=>$self->config_name('cast_from'),-size=>8), span('')),
                    td("to",textfield(-align=>'left', -type=>'text', -id=>'fl_cast_to', -pattern=>'^[\d\.]+$', -name=>$self->config_name('cast_to'),-size=>8), span('')),
```

```
128                   td({-width=>'250px'}, span("Must be a number")),
129                   ),
130                 TR({-class=>'dtblrows'},
131                   td({-nowrap=>"nowrap", -class=>"dtblheader"}, "% of sequence length masked by SEG"),
132                   td("from",textfield(-align=>'left', -type=>'text', -id=>'fl_seg_from', -pattern=>'^[\d\.]+$', -name=>$self->config_name('seg_from'),-size=>8), span('')),
133                   td("to",textfield(-align=>'left', -type=>'text', -id=>'fl_seg_to', -pattern=>'^[\d\.]+$', -name=>$self->config_name('seg_to'),-size=>8), span('')),
134                   td({-width=>'250px'}, span("Must be a number")),
135                   ),
136                 TR({-class=>'dtblrows'},
137                   td(),
138                 ),
139                 TR({-class=>'dtblrows'},
140                   td({-nowrap=>"nowrap", -class=>"dtblheader"}, "Mask types"),
141                   td({-nowrap=>"nowrap", -class=>"dtblheader"}, "detected by CAST"),
142                   td({-nowrap=>"nowrap", -class=>"dtblheader"}, "detected by SEG"),
143                   ),
144                  TR({-class=>'dtblrows'},
145                   td({-nowrap=>"nowrap", -class=>"dtblheader", -colspan=>"1"}, ""),
146                   td({-nowrap=>"nowrap", -class=>"dtblheader", -colspan=>"1"}, scrolling_list(-name=>$self->config_name('cast_types'), -values => \@cast_types, -multiple => 'true', -labels => \%cast_labels)),
147                   td({-nowrap=>"nowrap", -class=>"dtblheader", -colspan=>"1"}, scrolling_list(-name=>$self->config_name('seg_types'), -values => \@seg_types, -multiple => 'true', -labels => \%seg_labels)),
148                   td({-width=>'250px'}, span("Select multiple values by holding ctrl when clicking")),
149                   ),
150                 TR({-class=>'dtblrows'},
151                   td(),
152                 ),
153                 TR({-class=>'dtblrows'},
154                   td({-nowrap=>"nowrap", -class=>"dtblheader"}, "Search results"),
155                   td({-colspan=>"2", -align=>'left'}, radio_group(-name=>$self->config_name('search_type'), -values=>['browse', 'download'], -default=>'browse',
156                         -labels=>{browse=>"Browse (max 15000)",download=>'Download CSV file (max 50000)'}, -linebreak=>'true')),
157                   ),
158               );
159
160
161
162     $form .= $table;
163
164
165     return $form;
166   }
167
168   sub reconfigure {
169       my $self               = shift;
170       my $config             = $self->configuration;
171       $config->{accession}   = $self->config_param('accession');
172       $config->{gene}        = $self->config_param('gene');
173       $config->{name}        = $self->config_param('name');
174       $config->{organism}    = $self->config_param('organism');
175       $config->{length_from} = $self->config_param('length_from');
176       $config->{length_to}   = $self->config_param('length_to');
177       $config->{cast_from}   = $self->config_param('cast_from');
178       $config->{cast_to}     = $self->config_param('cast_to');
179       $config->{seg_from}    = $self->config_param('seg_from');
180       $config->{seg_to}      = $self->config_param('seg_to');
181       $config->{cast_types}  = $self->config_param('cast_types');
182       $config->{seg_types}   = $self->config_param('seg_types');
183       $config->{search_type} = $self->config_param('search_type');
184   }
185
186   sub auto_find {
187       my $self             = shift;
188       my $searchterm       = shift;
189
190       my $accession        = $self->config_param('accession');
191       my $name             = $self->config_param('name');
192       my $gene             = $self->config_param('gene');
193       my $organism         = $self->config_param('organism');
```

```
194         my $length_from      = $self->config_param('length_from');
195         my $length_to        = $self->config_param('length_to');
196         my $cast_from        = $self->config_param('cast_from');
197         my $cast_to          = $self->config_param('cast_to');
198         my $seg_from         = $self->config_param('seg_from');
199         my $seg_to           = $self->config_param('seg_to');
200         my @cast_types       = $self->config_param('cast_types');
201         my @seg_types        = $self->config_param('seg_types');
202         my $search_type      = $self->config_param('search_type');
203
204         if ($accession || $name || $gene || $organism || $length_from
205             || $length_to || $cast_from || $cast_to || $seg_from || $seg_to
206             || @cast_types || @seg_types )
207         {
208             $self->find();
209         } elsif ($searchterm =~ /^id\:/) {
210             return;
211         } else {
212             # warn "SEARCHTERM2\t$searchterm";
213             my @default_features = $self->db_search->search_features({-search_term =>
    $searchterm});
214             return \@default_features;
215         }
216     }
217
218
219     sub find {
220         my $self             = shift;
221
222         my $accession        = $self->config_param('accession');
223         my $name             = $self->config_param('name');
224         my $gene             = $self->config_param('gene');
225         my $organism         = $self->config_param('organism');
226         my $length_from      = $self->config_param('length_from');
227         my $length_to        = $self->config_param('length_to');
228         my $cast_from        = $self->config_param('cast_from');
229         my $cast_to          = $self->config_param('cast_to');
230         my $seg_from         = $self->config_param('seg_from');
231         my $seg_to           = $self->config_param('seg_to');
232         my @cast_types       = $self->config_param('cast_types');
233         my @seg_types        = $self->config_param('seg_types');
234         my $search_type      = $self->config_param('search_type');
235
236
237         my @fields = ('load_id', 'gene', 'Note', 'Organism', 'length', 'perCAST', 'perSEG',
    'cA', 'cC', 'cD', 'cE', 'cF', 'cG', 'cH', 'cI', 'cK', 'cL', 'cM', 'cN', 'cP', 'cQ', 'cR',
    'cS', 'cT', 'cV', 'cW', 'cY', 'sA', 'sC', 'sD', 'sE', 'sF', 'sG', 'sH', 'sI', 'sK', 'sL',
    'sM', 'sN', 'sP', 'sQ', 'sR', 'sS', 'sT', 'sV', 'sW', 'sY');
238
239         my @ids = ();
240
241         my $db = $self->database or die "I do not have a database";
242         my $dbh = DBI->connect("DBI:mysql:database=lcr_explorer;host=localhost",
243                                "www-data", "*******",
244                                {'RaiseError' => 1});
245         my $sql = SQL::Abstract::Limit->new(limit_dialect => $dbh);
246
247         # Build the where clause
248
249         my %where = ();
250
251         $where{load_id} = {'like', 'PR_%' . $accession . '%'} if ($accession =~ /\w+/);
252         $where{gene} = {'like', "\%$gene\%"} if ($gene =~ /\w+/);
253         $where{Note} = {'like', "\%$name\%"} if ($name =~ /\w+/);
254         $where{Organism} = {'like', "\%$organism\%"} if ($organism =~ /\w+/);
255
256         foreach my $cast_type (@cast_types) {
257         $where{$cast_type} = {'>', 0};
258         }
259
260         foreach my $seg_type (@seg_types) {
261         $where{$seg_type} = {'>', 0};
262         }
263
264         # length
265         if ($length_from =~ /^\d+$/) {
```

```
266             if ($length_to =~ /^\d+$/) {
267                 $where{length} = { -between => [ $length_from, $length_to ] };
268             } else {
269                 $where{length} = {'>=', $length_from};
270             }
271         } else {
272             if ($length_to =~ /^\d+$/) {
273                 $where{length} = {'<=', $length_to};
274             }
275         }
276 
277         # perCAST
278         if ($cast_from =~ /^[\d\.]+$/) {
279             if ($cast_to =~ /^[\d\.]+$/) {
280                 $where{perCAST} = { -between => [ $cast_from, $cast_to ] };
281             } else {
282                 $where{perCAST} = {'>=', $cast_from};
283             }
284         } else {
285             if ($cast_to =~ /^[\d\.]+$/) {
286                 $where{perCAST} = {'<=', $cast_to};
287             }
288         }
289 
290         # perCAST
291         if ($seg_from =~ /^[\d\.]+$/) {
292             if ($seg_to =~ /^[\d\.]+$/) {
293                 $where{perSEG} = { -between => [ $seg_from, $seg_to ] };
294             } else {
295                 $where{perSEG} = {'>=', $seg_from};
296             }
297         } else {
298             if ($seg_to =~ /^[\d\.]+$/) {
299                 $where{perSEG} = {'<=', $seg_to};
300             }
301         }
302 
303         return if (!%where);
304 
305         if ($search_type eq 'browse') {
306             my ($stmt, @bind) = $sql->select('testlcrs_tags_pivot', 'ID', \%where, 'ID', 15000,
    0);
307             my $sth = $dbh->prepare($stmt);
308             $sth->execute(@bind);
309             while (my $ref = $sth->fetchrow_hashref()) {
310                 # warn "Found a row: id = $ref->{'ID'}";
311                 push @ids, $ref->{'ID'};
312             }
313             $sth->finish();
314             $dbh->disconnect;
315 
316             return (\@ids, 'No results found') if (scalar @ids == 0);
317 
318             my @features = $db->fetch_many(@ids);
319             # return (\@features, $organism);
320             return \@features;
321         } else {
322             my ($stmt, @bind) = $sql->select('testlcrs_tags_pivot', \@fields, \%where,
    'load_id', 50000, 0);
323             warn "$stmt";
324             warn "@bind";
325             my $sth = $dbh->prepare($stmt);
326             $sth->execute(@bind);
327 
328             # print header( 'text/csv' ) ;
329             print "Content-Type:application/x-download\n";
330             print "Content-Disposition:attachment;filename=SearchResults.txt\n\n";
331             my $csv_header = join ("\t", @fields);
332             print $csv_header . "\n";
333 
334             while (my @ref = $sth->fetchrow_array()) {
335                 last if (!@ref);
336                 my $row = join ("\t", @ref);
337                 $row =~ s/^PR_//;
338                 print $row . "\n";
339             }
```

```
340             $sth->finish();
341             $dbh->disconnect;
342             exit;
343         }
344 
345         # warn "$stmt";
346         # warn "@bind";
347     }
348 
349     1;
```

Code Snippet 6-5. *The cgi-bin Perl script that builds the html code for the "Record Summary" tables in LCR-eXXXplorer.*

```
#!/usr/bin/Perl
use LWP::UserAgent;
use HTTP::Request;
use CGI qw(:standard *table);
use strict;
use Bio::DB::SeqFeature::Store;
use Chart::Gnuplot;


my $query = new CGI;
my $ref   = $query->param('ref');
my ($entry_name, $organism, $length, $perCAST, $perSEG);
my $seq;
my @protein_names = ();
my @gene_names    = ();
my @pdb_ids    = ();
my %GOs           = ();
my $UniKB_fail    = 0;
my @AAs            = ("A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q",
"R", "S", "T", "V", "W", "Y");
my @db_total_stats = (7.33, 2.02, 5.51, 7.02, 3.77, 5.91, 2.31, 5.21, 5.81, 8.94, 2.48,
4.29, 4.60, 4.40, 5.88, 7.44, 5.65, 6.24, 1.56, 3.64);
my @db_SEG_stats   = (12.82, 3.76, 4.70, 6.54, 4.28, 14.26, 3.71, 7.69, 7.44, 9.49, 3.28,
4.77, 8.89, 9.83, 6.23, 11.65, 5.29, 9.64, 3.03, 3.6);
my @db_CAST_stats  = (1.60, 0.93, 0.75, 2.98, 0.22, 2.51, 1.68, 0.03, 2.24, 0.01, 0.00,
0.56, 2.98, 7.02, 0.53, 4.83, 1.09, 2.45, 0.93, 0.51);
my %CAST_types = ();
my %SEG_types  = ();

# First lets try to collect as much data as possible from UniprotKB

my $URL      =
"http://www.uniprot.org/uniprot/?query=accession:$ref&format=tab&columns=id,entry%20name,pr
otein%20names,genes,organism,length,go-id,go,database(PDB)";

my $agent    = LWP::UserAgent->new(env_proxy => 1,keep_alive => 1, timeout => 30);
my $header   = HTTP::Request->new(GET => $URL);
my $request  = HTTP::Request->new('GET', $URL, $header);
my $response = $agent->request($request);

# Check the outcome of the response
if ($response->is_success){

    my @response   = split /\n/, $response->content;

    my @features   = split /\t/, $response[1];

    @protein_names = split /\(/, $features[2];
    @gene_names    = split /\s/, $features[3];

    my @GOids      = split /\;\ /, $features[6];
    my @GOdesc     = split /\;\ /, $features[7];

    for (my $i = 0; $i < @GOids; $i++) {
          $GOs{$GOids[$i]} = $GOdesc[$i];
          last if ($i==4);  # keep a maximum of 5 GO terms
    }

    @pdb_ids = split /\;/, $features[8];

} else {    # Connectivity issues OR Accession no longer valid

    $UniKB_fail = 1;
}


# Let's get the rest of the data from the local mySQL database
# Open the feature database
my $db = Bio::DB::SeqFeature::Store->new( -adaptor   => 'DBI::mysql',
```

```
64                                                  -dsn        =>
      'dbi:mysql:database=lcr_explorer;host=localhost;user=www-data;password=******');
65
66    my $iterator = $db->get_seq_stream(-seqid=>$ref,
67                                       -type=>'protein',);
68
69    while (my $feature = $iterator->next_seq) {
70        my @fields;
71        @fields = $feature->get_tag_values('Alias');
72        $entry_name = $fields[-1];
73        @fields = $feature->get_tag_values('Organism');
74        $organism = $fields[0];
75        $length = $feature->length;
76        @fields = $feature->get_tag_values('perCAST');
77        $perCAST = $fields[0];
78        @fields = $feature->get_tag_values('perSEG');
79        $perSEG = $fields[0];
80        $seq    = $db->fetch_sequence($ref);
81
82
83        foreach my $AA (@AAs) {
84
85            my $ctag = "c$AA";
86            my $stag = "s$AA";
87            @fields = $feature->get_tag_values($ctag);
88            $CAST_types{$AA} = $fields[0];
89            @fields = $feature->get_tag_values($stag);
90            $SEG_types{$AA} = $fields[0];
91
92        }
93
94        if ($UniKB_fail) {
95
96            @protein_names = $feature->get_tag_values('Note');
97            @gene_names    = $feature->get_tag_values('gene');
98
99        }
100
101   last;   # we shouldn't have more than 1 results anyway
102   }
103
104   my $pr_names;
105
106   for (my $i=0; $i < @protein_names; $i++) {
107       my $name = $protein_names[$i];
108
109       $name =~ s/[\(\)]//g;
110       next if ($name =~ /^EC/);
111       if ($i == 0) {
112
113           $pr_names .= "<i>Recommended name:</i><br><b>$name</b><br>";
114
115       } elsif ($i == 1) {
116
117           $pr_names .= "<i>Alternative name(s):</i><br>$name<br>";
118
119       } else {
120
121           $pr_names .= "$name<br>";
122
123       }
124
125
126   }
127
128   my $gn_names;
129
130   for (my $i=0; $i < @gene_names; $i++) {
131       my $name = $gene_names[$i];
132
133       $name =~ s/[\(\)]//g;
134       if ($i == 0) {
135           if ($i == $#gene_names) {
136
137               $gn_names .= "<b>$name</b>";
138           } else {
```

```
139
140                 $gn_names .= "<i>Name:</i> <b>$name</b><br>";
141             }
142
143         } elsif ($i == 1) {
144
145             if ($i == $#gene_names) {
146                 $gn_names .= "<i>Synonyms:</i> $name";
147
148             } else {
149                 $gn_names .= "<i>Synonyms:</i> $name,";
150             }
151
152         } elsif ($i == $#gene_names) {
153
154             $gn_names .= "$name";
155
156         } else {
157
158             $gn_names .= "$name, ";
159
160         }
161
162
163     }
164
165     my $GO_html;
166
167     foreach my $key (sort keys %GOs) {
168         # $key =~ s/\s//g;
169         $GO_html .= a({-href=>'http://amigo.geneontology.org/amigo/term/' . $key,  -
    target=>"_blank"},$GOs{$key}) . '<br>';
170     }
171
172     if ((keys %GOs) == 5) {
173         my $UniKB_GO_Link = a({-href=>'http://www.uniprot.org/uniprot/' . $ref .
    '#section_terms', -target=>"_blank"}, "here");
174         $GO_html .= "Showing a maximum of 5 GO terms. Click " . $UniKB_GO_Link . " to see the
    full list in UniProtKB";
175     }
176
177     my $pdb_html;
178     if (@pdb_ids == 0) {
179
180         $pdb_html = "There are no solved 3D structures for this protein";
181
182     } else {
183
184         my $i = 0;
185         foreach my $id (@pdb_ids) {
186             warn "PDB id $i is $id";
187             if ($i == 4) {
188                 $pdb_html .= a({-href=>'http://pdb.org/pdb/explore/explore.do?structureId=' .
    $id,  -target=>"_blank"},$id) . '<br>';
189                 $pdb_html .= "Showing a maximum of 5 PDB is. Click " . a({-
    href=>'http://www.uniprot.org/uniprot/' . $ref . '#section_x-ref_structure', -
    target=>"_blank"}, "here") . " to see the full list in UniProtKB";
190             } elsif ($i < $#pdb_ids) {
191                 $pdb_html .= a({-href=>'http://pdb.org/pdb/explore/explore.do?structureId=' .
    $id,  -target=>"_blank"},$id) . ', ';
192             } else {
193                 $pdb_html .= a({-href=>'http://pdb.org/pdb/explore/explore.do?structureId=' .
    $id,  -target=>"_blank"},$id);
194             }
195             $i++;
196
197         }
198
199     }
200
201
202
203     # Let's make our charts
204     unless (-e "/var/www/gbrowse2/images/lcrcharts/uniprot.png") {
205     # if (-e "/var/www/gbrowse2/images/lcrcharts/uniprot.png") {
206         warn "CREATING CHART";
```

```
207         my $chart = Chart::Gnuplot->new(
208             output    => "/var/www/gbrowse2/images/lcrcharts/uniprot.png",
209             imagesize => "1, 0.35",
210             title     => {
211                     text => 'Compositional properties of UniprotKB',
212                     font => 'arial, 12',
213             },
214             ylabel    => "%",
215             xtics     => {
216                     mirror => 'off',
217             },
218             ytics     => {
219                     mirror => 'off',
220                     fontsize => 8,
221             },
222             border    => {
223                     sides => 'bottom, left',
224             },
225         );
226 
227         # DataSet object of the data set A
228         my $dataA = Chart::Gnuplot::DataSet->new(
229             xdata => \@AAs,
230             ydata => \@db_total_stats,
231             style => "histograms",
232             color => "#31CB31",
233             fill  => {density => 0.5},
234         );
235 
236         # DataSet object of the data set B
237         my $dataB = Chart::Gnuplot::DataSet->new(
238             xdata    => \@AAs,
239             ydata    => \@db_CAST_stats,
240             style    => "histograms",
241             color    => "#FF00FF",
242             fill     => {density => 0.5},
243             linetype => 'solid',
244         );
245 
246         # DataSet object of the data set C
247         my $dataC = Chart::Gnuplot::DataSet->new(
248             xdata    => \@AAs,
249             ydata    => \@db_SEG_stats,
250             style    => "histograms",
251             color    => "#6959CC",
252             fill     => {density => 0.5},
253             linetype => 'solid',
254         );
255 
256         # Plot the graph
257         $chart->plot2d($dataA, $dataB, $dataC);
258     }
259 
260     unless (-e "/var/www/gbrowse2/images/lcrcharts/$ref.png") {
261     # if (-e "/var/www/gbrowse2/images/lcrcharts/$ref.png") {
262 
263         my @seq_total_stats=();
264         my @seq_CAST_stats=();
265         my @seq_SEG_stats=();
266 
267         foreach my $AA (@AAs) {
268 
269             my $count = () = $seq =~ /$AA/g;  # count occurences of each AA type using the
    goatch operator
270             if ($count == 0) {
271                 push @seq_total_stats, 0;
272                 push @seq_CAST_stats, 0;
273                 push @seq_SEG_stats, 0;
274 
275             } else {
276                 push @seq_total_stats, sprintf("%.2f", ($count/$length)*100);
277                 push @seq_CAST_stats, sprintf("%.2f", ($CAST_types{$AA}/$count)*100);
278                 push @seq_SEG_stats, sprintf("%.2f", ($SEG_types{$AA}/$count)*100);
279             }
280         }
281 
```

```
282         warn "CREATING CHART";
283         my $chart = Chart::Gnuplot->new(
284             output    => "/var/www/gbrowse2/images/lcrcharts/$ref.png",
285             imagesize => "1, 0.4",
286             title     => {
287                     text => "Compositional properties of $ref",
288                     font => 'arial, 12',
289             },
290             xlabel    => "Amino acid type",
291             ylabel    => "%",
292             xtics     => {
293                     mirror => 'off',
294             },
295             ytics     => {
296                     mirror => 'off',
297                     fontsize => 8,
298             },
299             border    => {
300                     sides => 'bottom, left',
301             },
302         );
303 
304         # DataSet object of the data set A
305         my $dataA = Chart::Gnuplot::DataSet->new(
306             xdata => \@AAs,
307             ydata => \@seq_total_stats,
308             style => "histograms",
309             color => "#31CB31",
310             fill  => {density => 0.5},
311         );
312 
313         # DataSet object of the data set B
314         my $dataB = Chart::Gnuplot::DataSet->new(
315             xdata    => \@AAs,
316             ydata    => \@seq_CAST_stats,
317             style    => "histograms",
318             color    => "#FF00FF",
319             fill     => {density => 0.5},
320             linetype => 'solid',
321         );
322 
323         # DataSet object of the data set C
324         my $dataC = Chart::Gnuplot::DataSet->new(
325             xdata    => \@AAs,
326             ydata    => \@seq_SEG_stats,
327             style    => "histograms",
328             color    => "#6959CC",
329             fill     => {density => 0.5},
330             linetype => 'solid',
331         );
332 
333         # Plot the graph
334         $chart->plot2d($dataA, $dataB, $dataC);
335     }
336 
337     my $images_html = "<img src=/gbrowse2/images/lcrcharts/uniprot.png><br><img
    src=/gbrowse2/images/lcrcharts/$ref.png><br>";
338 
339     my $html = table({-ID=>"ProteinDetails", -class=>"detailstable", -width=>"100%",  -
    style=>"margin:auto"},
340                 TR({-class=>'dtblSummary', -align => 'center',},
341                     td({-colspan=>"3"},"Record Summary for <b><i>" . $ref . "</i></b>
    (UniProtKB Entry Name: $entry_name)")
342                     ),
343                 TR({-class=>'dtblrows'},
344                     td ({-nowrap=>"nowrap", -class=>"dtblheader", -width=>'150px'},"Protein
    Name(s)"),
345                     td ($pr_names),
346                     td ({-rowspan=>"8", -align => 'center'}, $images_html)
347                     ),
348                 TR({-class=>'dtblrows'},
349                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"Gene Name(s)"),
350                     td ($gn_names),
351                     ),
352                 TR({-class=>'dtblrows'},
353                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"Organism"),
```

```
354                     td ($organism . " " .
355                         a({-
       href=>"http://repeat.biol.ucy.ac.cy/mgb2/gbrowse/testlcrs/?q=$organism;plugin=AdvancedFind;
       plugin_do=Do;AdvancedFind.organism=$organism;AdvancedFind.search_type=browse" },'[Browse
       entries]')
356                         . " " .
357                         a({-
       href=>"http://repeat.biol.ucy.ac.cy/mgb2/gbrowse/testlcrs/?q=$organism;plugin=AdvancedFind;
       plugin_do=Do;AdvancedFind.organism=$organism;AdvancedFind.search_type=download"
       },'[Download]')
358                         )
359                     ),
360                 TR({-class=>'dtblrows'},
361                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"Sequence length"),
362                     td ($length),
363                     ),
364                 TR({-class=>'dtblrows'},
365                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"% masked by CAST"),
366                     td ($perCAST . '%'),
367                     ),
368                 TR({-class=>'dtblrows'},
369                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"% masked by SEG"),
370                     td ($perSEG . '%'),
371                     ),
372                 TR({-class=>'dtblrows'},
373                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"Gene Ontology (GO)"),
374                     td ($GO_html),
375                     ),
376                 TR({-class=>'dtblrows'},
377                     td ({-nowrap=>"nowrap", -class=>"dtblheader"},"3D structures"),
378                     td ($pdb_html),
379                     ),
380                 );
381     $html .= "<br>\n";
382
383     print "Content-type: text/html\n\n";
384     print "<pre>Failed</pre>" if $UniKB_fail;
385     print $html;
386
```

Code Snippet 6-6. *The "cgi-bin" script responsible for serving GBrowse the GFF3 file containing the UniProt/SwissProt annotations.*

```
#!/usr/bin/Perl
use LWP::UserAgent;
use HTTP::Request;
use CGI qw(:standard);
use strict;

my $query = new CGI;

my $ref = $query->param('ref');

my $URL = "http://www.uniprot.org/uniprot/$ref.gff";

my $agent = LWP::UserAgent->new(env_proxy => 1,keep_alive => 1, timeout => 30);
my $header = HTTP::Request->new(GET => $URL);
my $request = HTTP::Request->new('GET', $URL, $header);
my $response = $agent->request($request);

my $catRegions = qq {[RegionsTopological]
feature              = Topological_Domain:UniProtKB Transmembrane:UniProtKB
Intramembrane:UniProtKB
glyph                = box
bgcolor              = thistle
fgcolor              = black
height               = 15
description          = 1
label                = 0
key                  = Regions:Topological
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features

[Regions:Repeats]
feature              = Repeat:UniProtKB Compositional_bias:UniProtKB
glyph                = box
bgcolor              = tomato
fgcolor              = black
height               = 15
description          = 1
label                = 0
key                  = Regions:Repeats & Compositional Bias
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features


[Regions:Others]
feature              = Domain:UniProtKB Region:UniProtKB Coiled_coil:UniProtKB
Motif:UniProtKB Alternative_sequence:UniProtKB
glyph                = box
bgcolor              = violet
fgcolor              = black
height               = 15
description          = 1
label                = 0
key                  = Regions:Others
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features


[SecondaryStructure]
feature              = Helix:UniProtKB Turn:UniProtKB Beta_strand:UniProtKB
glyph                = box
bgcolor              = darkorchid
fgcolor              = black
height               = 15
```

```
description          = 1
label                = 0
key                  = Secondary Structure
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features


[MoleculeProcessing]
feature              = Propeptide:UniProtKB Chain:UniProtKB Signal_peptide:UniProtKB
glyph                = box
bgcolor              = orange
fgcolor              = black
height               = 15
description          = 1
label                = 0
key                  = Molecule Processing
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features


[Sites]
feature              = Active_site:UniProtKB Metal_binding:UniProtKB Binding_site:UniProtKB
Site:UniProtKB
glyph                = box
bgcolor              = violet
fgcolor              = black
height               = 15
description          = 1
label                = 0
key                  = Sites
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features


[AminoAcidMod]
feature              = Lipidation:UniProtKB Glycosylation:UniProtKB
Disulfide_bond:UniProtKB Cross-link:UniProtKB
glyph                = box
bgcolor              = darkorchid
fgcolor              = black
height               = 15
description          = 1
label                = 0
key                  = Amino acid modifications
group_label_position = left
bump                 = fast
hide empty subtracks = 1
link                 = http://www.uniprot.org/uniprot/$ref#section_features


};

my %ignoreTags = (
                  "Initiator methionine" => 1,
                  "Non-standard residue" => 1,
                  "Modified residue" => 1,
                  "Natural variant" => 1,
                  "Mutagenesis" => 1,
                  "Sequence uncertainty" => 1,
                  "Sequence conflict" => 1,
                  "Non-adjacent residues" => 1,
                  "Non-terminal residue" => 1,
                  );

# Check the outcome of the response
if ($response->is_success){

    my @response = split /\n/, $response->content;
    my @gff = ();

```

```
142         foreach my $line (@response) {
143 
144             next if ($line =~ m/^##sequence-region/);
145             my @tags = split /\t/, $line;
146             next if (exists $ignoreTags{$tags[2]});
147             my $type = $tags[2];
148             $type =~ s/Topological/Topo/;
149             $type =~ s/Transmembrane/TM/;
150             $type =~ s/Beta strand/BS/;
151             $type =~ s/Helix/AH/;
152             $type =~ s/Turn/L/;
153             if ($tags[8] =~ /Note=/) {
154                 $tags[8] =~ s/Note=/Note=$type:/;
155             } else {
156                 $tags[8] = "Note=$type;";
157             }
158             $tags[2] =~ s/\s/_/g;
159 
160             $line = join ("\t", @tags);
161             push @gff, $line;
162 
163         }
164 
165         print "Content-type: text/plain\n\n";
166         print $catRegions;
167         map {print "$_\n"} @gff;
168 
169     } elsif ($response->is_error) {
170         print "Content-type: text/plain\n\n";
171         print "Error:$URL\n";
172         print $response->error_as_HTML;
173     } else {
174         print "Content-type: text/plain\n\n";
175         print "Unable to load data from Uniprot\n";
176 
177     }
178 
```

Code Snippet 6-7. *The cgi-bin Perl script responsible for generating the data required to plot the ANCHOR and IUPRED predictions.*

```
#!/usr/bin/Perl

use strict;
use CGI qw(:standard);
use CGI::Carp qw(fatalsToBrowser);
use Bio::DB::SeqFeature::Store;
use Bio::Graphics::Wiggle;


print "Content-type: text/plain\n\n";

my $server = 'http://repeat.biol.ucy.ac.cy/';
my $cgifile = $server . 'cgi-bin/anchor/anchor_cgi';
my $ref    = param('ref');

# Open the feature database
my $db = Bio::DB::SeqFeature::Store->new( -adaptor   => 'DBI::mysql',
                                          -dsn       =>
'dbi:mysql:database=lcr_explorer;host=localhost;user=www-data;password=*******');


my $in_seq = $db->fetch_sequence($ref);

#die $in_seqs;

my $seqlen=length($in_seq);
my $title= ">$ref";
$in_seq = "$title\n$in_seq";


print qq{
[ANCHOR:Binding]
feature              = binding_region:ANCHOR
glyph                = graded_segments
min_score            = 0.5
max_score            = 1
# vary_fg            = 1
bgcolor              = blue
fgcolor              = blue
height               = 15
description          = 0
label                = 0
key                  = ANCHOR binding regions
bump                 = fast

[ANCHOR:Filtered]
feature              = filtered_region:ANCHOR
glyph                = segments
bgcolor              = limegreen
fgcolor              = black
height               = 15
description          = 0
label                = 0
key                  = ANCHOR filtered regions
bump                 = fast


[ANCHOR:Scores]
feature              = anchor_score:ANCHOR
glyph                = wiggle_xyplot
graph_type           = line
linewidth            = 2
description          = 0
scale                = right
bgcolor              = red
fgcolor              = black
height               = 80
key                  = ANCHOR scores
bicolor_pivot        = 0.5
```

```
pos_color            = blue
neg_color            = red
max_score            = 1
min_score            = 0
no_grid              = 1

[IUPRED:Globular]
feature              = globular_region:IUPRED
glyph                = segments
bgcolor              = skyblue
fgcolor              = black
height               = 15
description          = 0
label                = 0
key                  = IUPRED globular regions
bump                 = fast

[IUPRED:Scores]
feature              = iupred_score:IUPRED
glyph                = wiggle_xyplot
graph_type           = line
linewidth            = 2
description          = 0
scale                = right
bgcolor              = red
fgcolor              = black
height               = 80
key                  = IUPRED scores
bicolor_pivot        = 0.5
pos_color            = gold
neg_color            = skyblue
max_score            = 1
min_score            = 0
no_grid              = 1


};

print '##gff-version 3', "\n";
#  CREATE A TEMPORARY FILE FOR PROGRAMS THAT TAKE INPUT FROM A FASTA FILE
# my $tmp=$$.( 0 x (8-length($$)) );
my $tmp= "$ref.fasta" ;

unless (-e "./tmp/$tmp") {
    open TMP, "> ./tmp/$tmp";
    print TMP $in_seq;
    close TMP;
}


my ($binding_regions, $filtered_regions, $position_scores) = callANCHOR($tmp);

if(@$position_scores) { # Some anchor detected

    my @wig_scores;
    my $wig;
    # unless (-e "/var/www/tmp/$ref\_an.wig") {
        warn "Creating new wig file /var/www/tmp/$ref\_an.wig";
        $wig = Bio::Graphics::Wiggle->new("/var/www/tmp/$ref\_an.wig",1,
                                          {seqid => $ref,
                                           start => 1,
                                           step  => 1,
                                           min   => 0,
                                           max   => 1,
                                          }
        );
    # }

    foreach my $region (@$binding_regions){
        my @fields = split /\t/, $region;
        print
"$ref\tANCHOR\tbinding_region\t$fields[1]\t$fields[2]\t$fields[3]\t.\t.\tName=ANCHOR
binding $fields[0];Note=ANCHOR binding region $fields[0]\n";

    }

```

```
143         foreach my $region (@$filtered_regions){
144             my @fields = split /\t/, $region;
145             print "$ref\tANCHOR\tfiltered_region\t$fields[1]\t$fields[2]\t.\t.\t.\tName=ANCHOR
    filtered$fields[0];Note=ANCHOR filtered region $fields[0]\n";
146         }
147         foreach my $region (@$position_scores){
148             my @fields = split /\t/, $region;
149             # print "AN\t$fields[0]\t$fields[1]\n";
150 
151             # print
    "$ref\tANCHOR\tscore\t$fields[0]\t$fields[0]\t$fields[1]\t.\t.\t$ref:score\n";
152             push @wig_scores, $fields[1];
153         }
154 
155         print
    "$ref\tANCHOR\tanchor_score\t1\t$seqlen\t.\t.\t.\t;wigfile=/var/www/tmp/$ref\_an.wig\n";
156 
157         $wig->set_values(2 => \@wig_scores) # unless (-e "/var/www/tmp/$ref\_an.wig");
158 
159     } else { # No anchor detected
160         # print "No anchor detected\n";
161     }
162 
163     my ($globular_regions, $iuposition_scores) = callIUPRED($tmp);
164 
165     if(@$iuposition_scores) { # Some anchor detected
166 
167         my @wig_scores;
168         my $wig;
169         # unless (-e "/var/www/tmp/$ref\_iu.wig") {
170             warn "Creating new wig file /var/www/tmp/$ref\_iu.wig";
171             $wig = Bio::Graphics::Wiggle->new("/var/www/tmp/$ref\_iu.wig",1,
172                                               {seqid => $ref,
173                                                start => 1,
174                                                step  => 1,
175                                                min   => 0,
176                                                max   => 1,
177                                               }
178             );
179         # }
180 
181         foreach my $region (@$globular_regions){
182             my @fields = split /\t/, $region;
183             print "$ref\tIUPRED\tglobular_region\t$fields[1]\t$fields[2]\t.\t.\t.\tName=IUPRED
    globular $fields[0];Note=IUPRED globular region $fields[0]\n";
184 
185         }
186 
187         foreach my $region (@$iuposition_scores){
188             my @fields = split /\t/, $region;
189             push @wig_scores, $fields[1];
190         }
191 
192         print
    "$ref\tIUPRED\tiupred_score\t1\t$seqlen\t.\t.\t.\t;wigfile=/var/www/tmp/$ref\_iu.wig\n";
193 
194         $wig->set_values(2 => \@wig_scores) # unless (-e "/var/www/tmp/$ref\_iu.wig");
195 
196     } else { # No anchor detected
197         # print "No anchor detected\n";
198     }
199 
200 
201     sub callANCHOR
202     {
203             my $tmp = shift @_;
204             my $out = './tmp/' . $tmp . '.aout';
205 
206             `./anchor ./tmp/$tmp > $out` unless (-e $out);
207             open(ANCHOR, $out) or die("ANCHOR OUTPUT failure");
208             my @ret=<ANCHOR>;
209             close(ANCHOR);
210 
211             my @binding_regions;
212             my @filtered_regions;
213             my @position_scores;
```

```
214             my $region = 0;
215             my $binding_reg = 0;
216             my $prev_line = 0;
217             my $score = 0;
218
219             foreach my $line (@ret) {
220                 last if ($line =~ /^\#\s+None/);
221                 next if ($line =~ /^\#\ \w+/);
222                 next if ($line =~ /^\#\n/);
223                 chomp($line);
224
225                 $region++ if ($line =~ /^\#\tNo\./);
226                 # Capture binding regions
227                 if ($region == 1 & $line =~ /^\#\t\d/) {
228                     $line =~ /^\#\t(\d+)\s+(\d+)\s+(\d+)/;
229                     push @binding_regions, "$1\t$2\t$3";
230                 }
231                 if ($region == 2 & $line =~ /^\#\t\d/) {
232                     $line =~ /^\#\t(\d+)\s+(\d+)\s+(\d+)/;
233                     push @filtered_regions, "$1\t$2\t$3";
234                 }
235                 if ($line =~ /^\d+\t\w/) {
236                     $line =~ /^(\d+)\t\w+\s+([\d\.]+)\s+(\d)/;
237                     push @position_scores, "$1\t$2";
238                     if ($3 == 1) {
239                         $score += $2;
240                         $prev_line++;
241                     } elsif ($3 == 0 & $prev_line > 0) {
242                         $score = sprintf("%.2f", $score/$prev_line);
243                         $binding_regions[$binding_reg] .= "\t$score";
244                         $binding_reg++;
245                         $prev_line = 0;
246                         $score = 0;
247                     }
248                 }
249             }
250
251             return(\@binding_regions, \@filtered_regions, \@position_scores);
252     }
253
254
255     sub callIUPRED
256     {
257             my $tmp = shift @_;
258             my $lnout = './tmp/' . $tmp . '.iulnout';
259             my $glout = './tmp/' . $tmp . '.iuglout';
260
261             `./iupred ./tmp/$tmp long > $lnout` unless (-e $lnout);
262             `./iupred ./tmp/$tmp glob > $glout` unless (-e $glout);
263
264             my @globular_regions;
265             my @position_scores;
266
267
268             open(SCIUPR, $lnout) or die("SCIUPR OUTPUT failure");
269             my @ret=<SCIUPR>;
270             close(SCIUPR);
271
272
273             foreach my $line (@ret) {
274                 chomp($line);
275                 next if ($line eq '');
276                 next if ($line =~ /^\#/);
277                 $line =~ /\s+(\d+).+?([\d\.]+)/;
278                 push @position_scores, "$1\t$2";
279                 # warn "SC\t$1\t$2";
280             }
281
282
283             open(GLIUPR, $glout) or die("GLIUPR OUTPUT failure");
284             @ret=<GLIUPR>;
285             close(GLIUPR);
286
287
288             foreach my $line (@ret) {
289                 chomp($line);
```

```
290                 next if ($line =~ /^\#/);
291                 last if ($line =~ /^>/);
292                 next if ($line =~ /^Number/);
293 
294                 $line =~ /^\s+.+?(\d+)\.\s+(\d+)\D+(\d+)/;
295                 push @globular_regions, "$1\t$2\t$3";
296             }
297 
298             return(\@globular_regions, \@position_scores);
299     }
300 
```

Code Snippet 6-8. *check_all_rawblast.pl is a Perl script to run best hit analysis on BLASTP tabular output files. This script is only applicable to the results of an All-Against-All self-comparison.*

```
#!/usr/bin/Perl -w

=head1 NAME

 check_all_rawblast.pl  - run best hit analysis on BLAST m8 tabular output

=head1 VERSION

 version 1.01

=head1 SYNOPSIS

 check_all_rawblast.pl [--stats] [-e evalue] [-o outfile] inputfile1 inputfile2 ...

=head1 DESCRIPTION

 Command line options:
  --stats             -- boolean flag to print column header
  -e/--evalue evalue  -- filter by evalue (default is 100)
  -o/--out            -- optional outputfile to write data,
                         otherwise will write to STDOUT
  -h/--help           -- show this documentation

This script is only applicable to the results of an All-Against-All analysis
It creates a file with the status (see abbreviations) of each sequence in the BLAST
output and, optionally, a stats file with aggregated statistics.

This script has been optimized for use over huge BLAST m8 files (i.e. > 3GBs)

=head1 ABBREVIATIONS

Abbreviations used in the output file(s)

 EX:  Sequence not found in the m8 file (only applicable if the evalue filtering is used).
      Essentially its meaning is that sequences marked this way would have been excluded
      from the m8 file since they wouldn't produce any hits with evalues smaller than the
      evalue provided for filtering.
 SW:  Only one hit was found in the m8 file for this sequence and that was with itself
 EW:  Multiple hits were for this sequence but the one with itself had the smaller evalue.
 BW:  Multiple hits for this sequence had the same evalue but the one with itself had the
      highest bit score.
 IW:  Multiple hits for this sequence had the same evalue and bit score, but the one with
      itself had the highest Identity Percentage.
 AW:  Multiple hits for this sequence had the same evalue, bit score and identity percenta-
      ge, but the one with itself had the longest Alignment length.
 BHW: Sequence had multiple hits with all features (evalue, bit score, identity
      percentage, alignment length) equal but all sequences reported as hits (except for
      itself) gave themselves as best hit.
 NHW: Sequence had multiple hits with all features (evalue, bit score, identity
      percentage, alignment length) equal but all sequences reported as hits (except for
      itself) didn't produced any hits with this sequence.
 BNHW: Sequence had multiple hits with all features (evalue, bit score, identity
      percentage, alignment length) equal but all sequences reported as hits (except for
      itself) either gave themselves as best hit or didn't produced any hits with this
      sequence.
 SL:  Only one hit was found in the m8 file for this sequence and that wasn't with
      itself.
 EL:  Multiple hits were for this sequence but the one with itself didn't had the smaller
      evalue.
 BL:  Multiple hits for this sequence had the same evalue but the one with itself didn't
      had the highest bit score.
 IL:  Multiple hits for this sequence had the same evalue and bit score, but the one with
      itself didn't had the highest Identity Percentage.
 AL:  Multiple hits for this sequence had the same evalue, bit score and identity
      percentage, but the one with itself didn't had the longest Alignment length.
 T:   All features checked (evalue, bit score, identity percentage, alignment length plus
      the heuristics described for BHW, NHW, BNHW) couldnt separate the self-hit from the
      rest of the hits. Therefore there is a Tie.

```

```
70
71  =head1 AUTHORS - Ioannis Kirmitzoglou & Vasilis J. Promponas
72
73   Ioannis Kirmitzoglou    IoannisKirmitzoglou_at_gmail-dot-org
74   Vasilis J. Promponas    vprobon_at_ucy_dot_ac_dot_cy
75
76  =head1 DISCLAIMER
77
78  Copyright (c) <2009> <Ioannis Kirmitzolgou, Vasilis J. Promponas>
79
80  Permission to use, copy, modify, merge, publish and distribute
81  this software and its documentation, with or without modification,
82  for any purpose, and without fee or royalty to the copyright holder(s)
83  is hereby granted with no restictions and/or prerequisites.
84
85  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
86  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
87  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
88  IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
89  CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
90  TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
91  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
92
93  =cut
94
95
96  my $starttime = time();
97
98  use strict;
99
100 use Getopt::Long;
101
102 #==================================
103 #       Initilization
104 #==================================
105
106 my %wins = ('S',0,'E',0,'B',0,'I',0,'A',0,'BH',0,'NH',0,'BNH',0);
107 my %loses = ('S',0,'E',0,'B',0,'I',0,'A',0);
108 my $nties = 0;
109 my $excluded=0;
110 my %h=();
111 my %losers=();
112 my %lstats=();
113 my @proteins=();
114 my @mask=();
115 my %lmasks=();
116 my $linecount = 0;
117 my @gis = ();
118 my ($currGI, $prevGI)= ();
119
120 my ($evalue,$stats,$outfile) = ( 100);
121
122 GetOptions(
123         'stats'                 => \$stats,
124         'e|evalue:f'            => \$evalue,
125         'o|out|outfile:s'       => \$outfile,
126         'h|help'                => sub { exec('Perldoc',$0); exit; }
127         );
128
129 my $FOUT;
130 if( $outfile ) {
131     open($FOUT, ">$outfile") || die("$outfile: $!");
132 } else {
133     $FOUT = \*STDOUT;
134 }
135
136 $|=1;
137
138 while (<>) {    #Start reading input file(s)
139
140     $linecount++; print STDERR $linecount,"\t" if !($linecount % 100000);
141     chomp;
142
143         next if !$_; # Skip empty lines
144
145     my @t=split /\s+/;
```

```
146     my @temp = ($t[0],$t[1],$t[10],$t[11],$t[3],$t[2]);
147     my $a =shift @temp;             #Store Queryname in a separate variable
148     if (!defined $h{$a}) {  #Query not already in hash
149         $h{$a} = [\@temp];
150         if( $linecount > 1 ) {  #Not the first line of file
151             $prevGI = $currGI;
152             @gis = ($prevGI);
153             techologicalAdvance(\%h, \%losers, \%lstats, \@gis);
154         }
155         $currGI = $a;
156     } else {            #Query already in hash, just add data for the extra pairs
157         push @{$h{$a}},\@temp;
158     }
159
160 }
161
162 @gis = ($currGI);
163
164 techologicalAdvance(\%h, \%losers, \%lstats, \@gis);
165
166 print STDERR "Proceeding to CrossChecks\nPlease wait ...\n";
167 foreach my $natives( keys %losers ) {
168     my @subjects = ();
169     my @mask = @{$lmasks{$natives}};
170     foreach (@{$losers{$natives}})
171     {
172         push @subjects, ${$_}[0];
173     }
174
175     CrossCheck($natives, \@subjects, \@mask, \%losers);
176 }
177
178
179 #print STATS "$file";
180 if ($stats) {     #print aggregated statistics is stats file
181 my $stfile;
182 if( $outfile ) {
183     open($stfile, ">$outfile.stats") || die("$outfile.stats: $!");
184 } else {
185     $stfile = \*STDOUT;
186 }
187
188 print $stfile "EX\tSW\tEW\tBW\tIW\tAW\tBHW\tNHW\tBNHW\tSL\tEL\tBL\tIL\tAL\tT\n";
189 print $stfile "$excluded";
190 foreach ('S', 'E','B','I','A','BH','NH','BNH') {print $stfile "\t$wins{$_}";}
191 foreach ('S', 'E','B','I','A') {print $stfile "\t$loses{$_}";}
192 print $stfile "\t$nties";
193 print $stfile "\n";
194 %wins = ('S', 0, 'E',0,'B',0,'I',0,'A',0,'BH',0,'NH',0,'BNH',0);
195 %loses = ('S', 0, 'E',0,'B',0,'I',0,'A',0);
196 $nties = 0;
197 $excluded = 0;
198 }
199
200 sub techologicalAdvance {
201 my ($href, $lref, $lsref, $refgi) = @_;
202
203     foreach (@{$refgi}) {
204
205         my $native=$_;                  #QueryName and statistics
206                 my $native_score=0;
207                 my $native_eval=999999999;
208         my $native_align=0;
209                 my $native_identities=0;
210         my @subjects = ();
211                 my @evals = ();
212                 my @identities=();
213         my @bitsc = ();
214                 my @align_length=();
215         my ($bmin,$bmax) = (9999999999, 0);     #Dummy min and max bit score values
216         my ($emin,$emax) = (9999999999, -1);    #Dummy min and max e-values
217         my ($amin,$amax) = (9999999999, 0); #Dummy min and max aligned length
218         my ($imin,$imax) = (9999999999, 0); #Dummy min and max percent identities
219                 my $exs = 0;   #Excluded sequences count
220
221         foreach ( @{${$href}{$_}} ) {
```

```
222
223         if (${$_}[1]<=$evalue) {        #Filter results by evalue
224                     push @subjects, ${$_}[0];
225             push @evals, ${$_}[1];
226             push @bitsc, ${$_}[2];
227             push @align_length, ${$_}[3];
228             push @identities, ${$_}[4];
229
230
231             if ($native eq ${$_}[0]) { #keep best statistics of self comparison
232                                 $native_score= ${$_}[2]         if (${$_}[2]>$native_score) ;
233                                 $native_eval= ${$_}[1]          if (${$_}[1]<$native_eval) ;
234                                 $native_identities= ${$_}[4]    if
    (${$_}[4]>$native_identities) ;
235                                 $native_align= ${$_}[3]         if (${$_}[3]>$native_align) ;
236             }
237
238             if (${$_}[2]<$bmin) {$bmin=${$_}[2];}   #Find min and max values of bit score
239             if (${$_}[2]>$bmax) {$bmax=${$_}[2];}
240             if (${$_}[1]<$emin) {$emin=${$_}[1];}   #Find min and max values of e-value
241             if (${$_}[1]>$emax) {$emax=${$_}[1];}
242             if (${$_}[3]<$amin) {$amin=${$_}[3];}   #Find min and max values of aligned length
243             if (${$_}[3]>$amax) {$amax=${$_}[3];}
244             if (${$_}[4]<$imin) {$imin=${$_}[4];}   #Find min and max values of identities
    percentage
245             if (${$_}[4]>$imax) {$imax=${$_}[4];}
246         } else {
247                         $exs++;
248
249                 }
250                 }
251                 push  @{${$lsref}{$native}}, $#subjects+$exs+1; #Insert total hits in the
    statistics array
252
253                 if (@subjects==0) {
254                         my $c=$#{${$lsref}{$native}};
255                         while ($c<7) {push @{${$lsref}{$native}}, 0; $c++;}
256                         print $FOUT "$native\tEXCLUDED ($evalue)\t@{${$lsref}{$native}}\n";
257                         $excluded++;
258                         delete ${$href}{$native};
259                         next;
260                 } elsif( @subjects==1 ) {
261                         push @{${$lsref}{$native}}, 1;
262                         my $c=$#{${$lsref}{$native}};
263                         while ($c<7) {push @{${$lsref}{$native}}, 0; $c++;}
264             if( $subjects[0] eq $native ) {
265                 print $FOUT "$native SINGLE-WIN @{${$lsref}{$native}}\n";
266                 $wins{'S'} ++;
267             } else {
268                 print $FOUT "$native SINGLE-LOSE @{${$lsref}{$native}}\n";
269                 $loses{'S'} ++;
270             }
271             delete ${$href}{$native};
272             next;
273         }
274
275         @mask=0..$#subjects;
276         my $status=0;
277         #my @prt = ();
278         if (!CompareList (\@evals,\@mask, $native_eval, $emin, $emax, 'E', $native,
    \@{${$lsref}{$native}})) {
279             if (!CompareList (\@bitsc,\@mask, $native_score, $bmax, $bmin, 'B', $native,
    \@{${$lsref}{$native}})) {
280                 if (!CompareList (\@identities,\@mask, $native_identities, $imax, $imin, 'I',
    $native, \@{${$lsref}{$native}})) {
281                     if (!CompareList (\@align_length,\@mask, $native_align, $amax, $amin, 'A',
    $native, \@{${$lsref}{$native}})) {
282                         my $tmp = delete ${$href}{$native};  #delete from hash to free memory
283                         my @panaplo = @mask;
284                         $lmasks{$native} = \@panaplo;
285                                                 ${$lref}{$native} = $tmp;  #add deleted element
    to losers array (to pass it to CrossChecks)
286                     }
287                 }
288             }
289         }
```

```
290
291
292     }
293 }
294
295
296 print "Elapsed time: ", time()-$starttime, " seconds\n";
297
298 sub CrossCheck {
299 my $GI = shift;      #Proteins to crosscheck
300 my $ties = shift;    #Reference to array @subjects, which contains GI and proteins in tie
301 my $mask = shift;    #Reference to array containing index of $ties proteins
302 my $h = shift;       #Reference to hash containing all proteins and data
303 #my @pr=shift;
304
305 my @pr=@{$lstats{$GI}};
306 my $n=$#{$mask}; my $b=$n; my $c=$n; my $eye = $pr[$#pr];
307
308 foreach (@{$mask}) {    #For each protein that ties with GI
309 my @temp=(); my $GItmp=${$ties}[$_];
310     if ($GI ne $GItmp && defined(${$h}{$GItmp})) {      #Check if GItemp has already a Best hit
    (it will be absent from %h)
311         my $i=0;
312         foreach (@{${$h}{$GItmp}}) {
313             push @temp, ${$_}[0]
314         }
315         foreach (@temp) {if ($GI eq $_) {$i++;}}    #Check if GItemp reports GI as a hit
316             if ($i==0) {$n--;}
317
318     } elsif ($GI ne $GItmp && !defined(${$h}{$GItmp})) {$b--;}
319
320     }
321 #if ($b+$n>$c && $b+$n<2*$c) {print "$n, $b, $c\n";}
322 if ($b==0) {        #All proteins in tie with GI have Best hits
323     push @pr, $b+1;
324     my $c=$#pr;
325     while ($c<7) {push @pr, 0; $c++;}
326     $wins{'BH'}++;
327     print $FOUT "$GI BH-WIN @pr\n";
328 } elsif ($n==0) {   #All proteins in tie with GI don't report it as a hit
329     push @pr, $b+1;
330     push @pr, $n+1;
331     push @pr, 0;
332     $wins{'NH'}++;
333     print $FOUT "$GI NH-WIN @pr\n";
334 } elsif ($n+$b==$c) {   #All proteins in tie with GI either have best hits or don't report it
    as a hit
335     push @pr, $b+1;
336     push @pr, $n+1;
337     push @pr, 1;
338     $wins{'BNH'}++;
339     print $FOUT "$GI BNH-WIN @pr\n";
340 }
341 else {
342 $nties++;
343     push @pr, $b+1;
344     push @pr, $n+1;
345     push @pr, $n+1;
346 #   print "Hello\n";
347 print $FOUT "$GI TIE @pr\n";
348 }
349
350 }
351
352 sub CompareList {
353 my $lref=shift; #Array reference
354 my $mask=shift; #Array reference to check
355 my $val=shift;
356 my $extr1=shift;
357 my $extr2=shift;
358 my $mode=shift;
359 my $native=shift; #Protein name
360 my $pr = shift;
361 my @ret =();
362
363
```

```
364 if ($val==$extr1) {
365
366     if ($val==$extr2) {
367                 push @{$pr}, $#{$mask}+1; return 0;
368         } else {
369         my $n=0;
370
371         foreach (@{$mask}) {
372                         if (${$lref}[$_]==$extr1) {push @ret, $_; $n++;}
373         }
374
375                 if ($n>1) {
376             @{$mask}=@ret;
377             push @{$pr}, $#{$mask}+1;
378             return 0;
379         } else {
380             push @{$pr}, $#{$mask}+1;
381             my $c=$#{$pr};
382             while ($c<7) {push @{$pr}, 0; $c++;}
383             print $FOUT "$native $mode-WIN @{$pr}\n";
384             $wins{$mode}++;
385             delete $h{$native};
386             return 1;
387         }
388     }
389
390
391 } else {
392 push @{$pr}, $#{$mask}+1;
393 my $c=$#{$pr};
394 while ($c<7) {push @{$pr}, 0; $c++;}
395 print $FOUT "$native $mode-LOSE @{$pr}\n";
396 delete $h{$native};
397 $loses{$mode}++;
398 return 1;
399 }
400
401 }
402
```

Code Snippet 6-9. *Combine_stat_files.pl is a script to combine the output files created by check_all_rawblast.pl into a single file with just one header. The script can also convert the raw numbers in the output files to percentages and calculate the number of sequences that didn't produce any significant hits.*

```
#!/usr/bin/Perl -w

# v1.0 2009 by Ioannis Kirmitzoglou
# The purpose of this script is to combine *.check.stats files created using the
# check_all_rawblast_final.pl script in to a single file with just one
# header. An added benefit is that the script can (optionally) convert the
# raw numbers in the *.check.stats files to percentages and calculate the number
# of sequences not having a hit in the *.m8 files and add this to the EX
# category calculated by the check_all_rawblast_final.pl script. To do
# that the user has to provide the number of sequences in the query file.

# The script accepts 4 parameters, whether it should calculate
# percentages, the number of sequences in the query file used to run
# blastp, a regex to match the files it should combine and the output filename

#########################################################
#------------------  USAGE  ----------------------#
#########################################################
# Combine_stats_files.pl usage:
#    -o <base output filename> | optional, extension .merged will be added. If ommited
       # STDOUT will be used instead
#    -i <input *.stats files> | optional - wildcards are supported, if ommited all the
#       *.check.stats files in the current directory will be parsed.
#       e.g. -i E_coli will match all E_coli*.check.stats files
#           # -i \"E*col\" will match all E*col*.check.stat files
#       When using wildcards this options MUST BE QUOTED
#    -n <number of sequences in the query file> | required
#    -p <true|false> | required, if true the script will convert raw numbers to percentages.
The
#       output file will have .merged.perc as extension. Default is false
#    -s <separator, number> | optional, set every how many files will a separator (double new
line)
#       be added to the output file. Default is 0 (no separators)

#########################################################
#----------------  INCLUDES  --------------------#
#########################################################

use strict;
use Getopt::Std;
use Cwd;
#########################################################
#--------------  DECLARATIONS  -------------------#
#########################################################

my %options=();
my @myfiles=();
my $usage = "
Combine_stats_files.pl usage:
    -o <base output filename> | optional, extension .merged will be added. If ommited
       STDOUT will be used instead
    -i <input *.stats files> | optional - wildcards are supported, if ommited all the
       *.check.stats files in the current directory will be parsed.
       e.g. -i E_coli will match all E_coli*.check.stats files
            -i \"E*col\" will match all E*col*.check.stat files
       When using wildcards this options MUST BE QUOTED
    -n <number of sequences in the query file> | required
    -p <true|false> | required, if true the script will convert raw numbers to percentages. The
       output file will have .merged.perc as extension. Default is false
    -s <separator, number> | optional, set every how many files will a separator (double new
line)
       be added to the output file. Default is 0 (no separators)
       ";

#########################################################
#--------------  GET OPTIONS  --------------------#
```

```perl
#######################################################

unless (getopts("o:i:n:s:p", \%options)) {die "$usage\n";} # wrong arguments provided

# Check for required options
unless($options{n}) {die "\n********* ERROR *********\n* option -n is REQUIRED *\n" . '*' x 25
. "\n$usage\n";}

my $SeqNum = $options{n};

# Determine which output option to use
if (!$options{o}) {
    open OUTPUT, ">-";
} else {

    if (!$options{p}) { # Should I convert raw number to percentages?
        open OUTPUT, ">$options{o}.merged" or die "Cannot create file $options{o}.merged: $!";
    } else {
        open OUTPUT, ">$options{o}.merged.perc" or die "Cannot create file
$options{o}.merged.perc: $!";
    }

}

$options{s} = 0 if (!$options{s});
print STDERR "----- $options{s} ----\n";

#######################################################
#                 AUXILIARY FUNCTIONS                 #
#######################################################

sub recurse($) {    # scan working dir and create an array with the matching files it contains
  my($path) = @_;

  # print the directory being searched
  # print "** Path is $path\n";

  # loop through the files contained in the directory
  for my $eachFile (glob($path.'*check.stats')) {

    # if the file is a directory
    if( -d $eachFile) {
      # pass the directory to the routine ( recursion )
      next;
    } else {

      # print the file ... tabbed for readability
      # print "$eachFile\n";
      push (@myfiles, $eachFile);
    }
  }
}
# sub round accepts 2 arguments:
# The first argument is the number to round, and the second argument is the
# number of places after the decimal point to round to.
sub round {
    my ($n, $places) = @_;
    my $factor = 10 ** ($places || 0);
    return int(($n * $factor) + ($n < 0 ? -1 : 1) * 0.5) / $factor;
}

#######################################################
#                       MAIN                          #
#######################################################

# Collect the files to be merged
my $dir = cwd;
if ($options{i}) {
    # if an input mask is given append it to the current working dir, also append "/" if
necessary
    $dir !~ /\/$/ ? ($dir .= "/".$options{i}) : ($dir .= $options{i});
} else {
    # append "/" if necessary
    $dir .= '/' if($dir !~ /\/$/);
}

```

```
recurse($dir);
# Inform user how many files we found or exit if there is no match

my $arrSize = scalar @myfiles;

print STDERR "Found $arrSize files matching $dir*.check.stats\n";

exit if ($arrSize == 0);


# Merge the files
my $printLine = 0;
my $separator = 0;
my $header = "";
my $fileIndex = 1;
foreach (@myfiles) {

    my $file = $_;
    my $mode = $file;
    $mode =~ s#.*/##;
    $mode =~ s/\.check\.stats//i;
    $mode =~ s/.*_//i;
    print "$mode\n";
    my $dieStr = "
***** ERROR *****
The number of sequences you provided is smaller than the total of hits inside:\n$file
** NOW EXITING **
";

    print STDERR "Now processing $file\n";
    open FILE, "<$file" or die "Couldn't load $file: $!";
    my $lineNumber = 1;
    while (<FILE>) {

        chomp;
        next if !$_; # Skip empty lines

        if (($printLine == 0) && ($lineNumber == 1)) {

            $header = "Mode\t$_\n";
            print OUTPUT $header;

        }


        if (($lineNumber>1) && (!$options{p}) && ($lineNumber > 1) ) {

            my @t = split /\s+/;
            my $total = 0;
            foreach (@t) {$total += $_};

            # options{n} is smaller than the actual number of sequences
            die $dieStr if ($options{n}<$total);
            $t[0] = $options{n} - $total;

            # print numbers
            print OUTPUT "$mode\t";
            foreach (@t) {print OUTPUT "$_\t"};
            print OUTPUT "\n";

        } elsif (($lineNumber>1) && ($options{p}) && ($lineNumber > 1) ) {

            my @t = split /\s+/;
            my $total = 0;

            foreach (@t) {$total += $_};

            # options{n} is smaller than the actual number of sequences
            die $dieStr if ($options{n}<$total);

            $t[0] = $options{n} - $total;

            # print percentages
            print OUTPUT "$mode\t";

            foreach (@t) {
```

```
213
214                 my $perc = ($_ / $options{n}) * 100;
215                 $perc = round($perc,3);
216                 print OUTPUT "$perc\t"
217
218             }
219
220             print OUTPUT "\n";
221         }
222
223         $lineNumber ++;
224     }
225     $separator ++;
226     $printLine ++;
227     $fileIndex ++;
228
229     if (($separator == $options{s}) && ($fileIndex < $arrSize)) {print OUTPUT "\n\n$header";
    $separator = 0;}
230 }
231
232 close OUTPUT;
```

Code Snippet 6-10. *A Perl script used to parse tabular outputs of BLASTP, determine TP and FP hits, rank them by ascending $E-value$ and generate various plots. Several similar scripts we created to produce different plot types.*

```
1   #!/usr/bin/Perl
2
3   ######################################################
4   #                  VERSION INFO                      #
5   ######################################################
6   # v2.1 12:09 μμ 4/5/2009 Ioannis Kirmitzoglou
7   #    changes from v2.0:
8   #    added the -e option to filter m8 files based on the evalue
9   #    added missing explanation for the -g switch
10  #    corrected explanation for the -s switch
11
12
13  ######################################################
14  #                     PURPOSE                        #
15  ######################################################
16  # This script was created to:
17  # 1) Create ROCn curves for various n which can be inputed as an array of numbers
18  # 2) Calculate the mean ROCn value and standard deviation as described in NAR 29:2994-3005,
    2001
19  # It is designed to work on one as well as multiple datasets. Those datasets are actually
    BLASTP m8
20  # output files containing the results of ASTRAL database self comparison.
21  #First positives next negatives and vice-versa (clarify this comment)
22
23  ######################################################
24  #                      USAGE                         #
25  ######################################################
26  #
27  # You can call the script using a combination of the switches described below
28  # usage:
29  #    -o <base output filename> | required. Extensions (e.g. ROC50, png) will be added
    accordingly
30  #    -i <input m8 file> | optional, if omitted all files in the working dir with extension *.m8
    will be used
31  #    -a <original ASTRAL fasta file> | required. This is the original fasta file used as a
    BLASTP query
32  #    -n <ROCn> | optional, if omitted the standard ROC100 will be estimated. It can be an array
    of numbers seperated with spaces
33  #    -f <0/1> | optional (default is 0). If set to true then only the members of a SCOP family
    will be considered homologues. Otherwise
34  #        as homologues (TP) are considered the members of a SCOP superfamily
35  #   -u <0/1> | optional (default is 0). If set to true all undetermined hits are treated as FP
36  #   -s <e/a/b> | optional (defaults is e). If set to e, e-values will be used for the sorting
    of hits. If set to b, bit score,  if set to a all
37  #       features will be used instead
38  #   -e <e-value filter> | optional. If set, all records having greater e-values will be
    dismised from further analysis
39  #   -g <0/1> | optional, default is false. If true, the script will not create any intermediate
    files and will just create graphs.
40  #      Useful if you already have the intermediate files
41  #
42
43  #
44  # The original ASTRAL file is used to a) calculate the number of all the possible true
    positives b) identify true and false positives according to the -f switch.
45  # If -f is set to true then as TP are considered only the hits between the members of a SCOP
    family. Hits between members of a superfamily are treated as undetermined and excluded from the
    analysis while all other hits are treated as FP.
46  # if -f is set to false (default) then as TP are considerer all the hits between members of a
    SCOP superfamily (as described in IEEE proc 90: 1834-1847, 2002). Hits between the members of a
    SCOP fold are treated as undeterminded and excluded from the analysis while all other hits are
    treated as FP.
47  #
48  # Output files: ROCn diagramms (from R?), A summary file for each m8 output file with NFP, NTP,
    ROCn, SD etc
49
50  ######################################################
51  #                     INCLUDES                       #
52  ######################################################
53
54  use strict;
```

```perl
use Getopt::Std;
use Statistics::R;  #for sending commands directly to R
use Cwd;
use Sort::Fields;
use POSIX qw(ceil floor);

use Switch;

$|=1;
#####################################################
#                   DECLARATIONS                    #
#####################################################


my $usage = "usage:
    -i <input m8 file> | optional, if omitted all files in the working dir with
       extension *.m8 will be used
    -a <original ASTRAL fasta file> | required. This is the original fasta file
       used as a BLASTP query
    -n <ROCn> | optional, if omitted the standard ROC100 will be estimated. It
       can be a series of numbers enclosed in quotes and seperated with spaces
    -f <0/1> | optional (default is 0). If set to true then only the members of
       a SCOP family will be considered homologues. Otherwise as homologues (TP)
       are considered the members of a SCOP superfamily
    -u <0/1> | optional (default is 0). If set to true all undetermined hits
       are treated as FP
    -s <e/a/b> | optional (defaults is e). If set to e, e-values will be used
       for the sorting of hits. If set to b, bit score,  if set to a all features
       will be used instead
    -e <e-value filter> | optional. If set, all records having greater e-values
       will be dismised from further analysis
    -g <0/1> | optional, default is false. If true, the script will not create
       any intermediate files and will just create graphs. Useful if you already
       have the intermediate files
    ";

my %options=();
my %SeqFam=();
my %SeqSfam =();
my %SeqFld = ();
my %HomCounts = ();
my %SeqLen = ();
my $TotTP = 0;
my %scores = ();
my @myfiles = ();   # array with *m8 files that will be proccessed
my ($dir, $dir1)="";    #current working directory
my $HUnd=""; # hash reference for the detection of undetermined
my $HRef = ""; # hash reference for the detection of TP and FP
my @n = (); # array containing the values of the -n switch
#my (@eval, @bitsc, @perident, @normlen) =(); # arrays containing e-values, bit scores, % identity and normalized alignment length (alingment length/query length) respectively)
my @scores = ();
my @cpfiles = ();
my $s = 0;
my %ls =();
my $evalFilter = 1000000; # evalue filter default value must be high enough to prevent dismising records under normal blastp use


#####################################################
#                   GET OPTIONS                     #
#####################################################
my $args = "";
foreach (@ARGV) { $args.= " $_"}
unless (getopts("i:a:n:fus:e:g", \%options)) {die "$usage\n";} # wrong arguments provided
unless ($options{a}) {die "$usage\n";} #check if required options are provided

unless (!$options{s} || $options{s} eq 'e' || $options{s} eq 'a' || $options{s} eq 'b') {die "WRONG PARAMETER: -s switch must either e, b or a\n";}

$options{s} = "e" if (!$options{s});    # set default value for -s switch if ommited
if (!$options{n}) {
    push (@n, 100);
} else {
    @n = split (/\s/, $options{n});
}
```

```
128
129 $evalFilter = $options{e} if (defined $options{e});
130
131
132 ######################################################
133 #               AUXILIARY FUNCTIONS                  #
134 ######################################################
135
136  sub hashValueDescendingNum {   #sort descending, good for bit scores
137     $ls{$b} <=> $ls{$a};
138  }
139
140   sub hashValueAscendingNum {   #sort ascending, good for e-values
141     $ls{$a} <=> $ls{$b};
142  }
143
144  sub recurse($) {   # scan working dir and create an array with the *m8 files it contains
145   my($path) = @_;
146
147   ## append a trailing / if it's not there
148   $path .= '/' if($path !~ /\/$/);
149
150   ## print the directory being searched
151   #print $path,"\n";
152
153   ## loop through the files contained in the directory
154   for my $eachFile (glob($path.'*m8')) {
155
156     ## if the file is a directory
157     if( -d $eachFile) {
158       ## pass the directory to the routine ( recursion )
159       next;
160     } else {
161
162       ## print the file ... tabbed for readability
163       print "$eachFile\n";
164       push (@myfiles, $eachFile);
165     }
166   }
167 }
168
169 ######################################################
170 #               SET GRAPH OPTIONS                    #
171 ######################################################
172
173     my %colors = ();
174     $colors{"40N"} = qq{set style line 401 lt 1 lc rgb "black" lw 2 pt 0};
175     $colors{"40QH"} = qq{set style line 402 lt 5 lc rgb "black" lw 2 pt 0};
176     $colors{"40QHCBS"} = qq{set style line 403 lt 4 lc rgb "black" lw 2 pt 0};
177
178     $colors{"20N"} = qq{set style line 201 lt 4 lc rgb "black" lw 2 pt 0};
179     $colors{"20QH"} = qq{set style line 202 lt 2 lc rgb "black" lw 2 pt 0};
180     $colors{"20QHCBS"} = qq{set style line 203 lt 6 lc rgb "black" lw 2 pt 0};
181     $colors{100} = qq{set style line 100 lt 0 lc rgb "gray" lw 2 pt 0};
182
183
184
185     $ls{"40N"} = 401;
186     $ls{"40QH"} = 402;
187     $ls{"40QHCBS"} = 403;
188
189     $ls{"20N"} = 201;
190     $ls{"20QH"} = 202;
191     $ls{"20QHCBS"} = 203;
192
193
194
195 ######################################################
196 #                       MAIN                         #
197 ######################################################
198 # is a directory or a file given via the -i switch ?
199 if (!$options{i}) { #no
200     print "Getting directory file listing:\n";
201     $dir = cwd;
202     recurse($dir);
203 } else {    #yes
```

```
204     if ( -d $options{i}) {  # a dir is given
205         print "Getting directory file listing:\n";
206         $dir = $options{i};
207         recurse($dir);
208 
209     } else {    # a file is given
210         push (@myfiles, $options{i});
211     }
212 }
213 
214 push @cpfiles, @myfiles;
215 
216 
217     $/="\n>";
218     open ASTRAL, "<$options{a}" or die "Unable to open $options{a}: $!";
219     my $sn=0;
220     my $dbSize =0;
221     print "Reading ASTRAL file and determining possible True Positives...\n";
222 
223     while (<ASTRAL>) {  # read the ASTRAL fasta file and extract SCOP code, Fold, Superfamily
    and family for each sequence
224             chomp;
225             s/^\>//;
226             $sn++;
227             #$1 Protein name,  $2Protein family, $3 Protein superfamily, $4 Protein Fold, $5
    Protein Sequence
228             /^(\S+)\s(((\S+\.\d+\.)\d+\.)\d+).+\s([\w\s]+)/i;
229             my $id = $1;
230             $SeqFam{$id} = $2;
231             $SeqSfam{$id} = $3;
232             $SeqFld{$id} = $4;
233             my $seq = $5;
234             $seq =~ s/\n//g; # Remove line brakes
235             my $len = length($seq);
236             $SeqLen{$id}= $len;
237             $dbSize++;
238 
239         }
240     close (ASTRAL);
241 
242 
243     $/="\n";
244 
245     #calculate number of possible TP
246     if (!$options{f}) { # TP are considered members of the same superfamily
247 
248         foreach my $key (keys %SeqSfam) {
249             if (defined $HomCounts{$SeqSfam{$key}})
250             {
251             $HomCounts{$SeqSfam{$key}}++;
252             } else {
253             $HomCounts{$SeqSfam{$key}} = 1;
254             }
255         }
256 
257     } else {
258 
259         foreach my $key (keys %SeqFam) {    # TP are considered members of the same family
260             if (defined $HomCounts{$SeqFam{$key}})
261             {
262             $HomCounts{$SeqFam{$key}}++;
263             } else {
264             $HomCounts{$SeqFam{$key}} = 1;
265             }
266         }
267     }
268     my $loners = 0;
269     foreach my $key (keys %HomCounts) {
270         if ($HomCounts{$key} ==1)
271         {
272             delete ($HomCounts{$key});
273             $loners++;
274 
275         } else {
276             $TotTP+= $HomCounts{$key} * ($HomCounts{$key}-1);   #Total numbers of possible TP
277         }
```

```
278     }
279     my $related = $sn-$loners;
280     print "ASTRAL file contains $sn sequences out of which $related are related to at least one
    other sequence.\n";
281     print "Maximum number of True Positives is $TotTP\n";
282 
283         @n = sort {$a <=> $b} @n;
284         $s = $n[-1]; # store the highest provided n
285         # print "$s\n";
286 
287 
288 
289 if (!$options{g}) { # Intedmediate files have already been created, just draw graphics
290 
291     if ($options{f}) {  # TP are considered members of the same family
292         $HRef =\%SeqFam;
293         $HUnd =\%SeqSfam;
294         %SeqFld = (); # no longer needed
295     } else {    # TP are considered members of the same superfamily
296         $HRef =\%SeqSfam;
297         $HUnd =\%SeqFld;
298         %SeqFam = (); # no longer needed
299     }
300 
301 
302     # Iterate through the list of *m8 files to be proccesed
303     foreach (@myfiles) {
304 
305         my $file = $_; # store file name
306         print "============================\nNow processing file $file\n";
307         open STATS, ">$file.rstats" or die "Unable to create $file.rstats file: $!";
308         open M8, "<$_" or die "Unable to open $_: $!";
309         my ($tp, $fp, $un) =0; # count TP, FP and undetermined in the file
310         my $c =0; # just a counter
311         my ($prevQuer, $prevHit) = (0,0);
312         print "Reading and parsing file...\n";
313         if (!$options{u}) { # ignore undetermined
314             while (<M8>)
315                 {
316                 chomp;
317                 next if !$_; # Skip empty lines
318 
319                 my @t=split /\s+/;
320 
321                 next if ($t[1] eq $prevHit && $t[0] eq $prevQuer); # skip multiple HSPs
322                 next if ($t[0] eq $t[1]); #exlude self matches
323                 next if ($t[10] > $evalFilter); # exlude records based on evalue filter (option
    -e)
324                 # print "$t[0]\t$SeqLen{$t[0]}\n";
325                 my $NormLen = $t[3]/$SeqLen{$t[0]};
326                 if (${$HRef}{$t[0]} eq ${$HRef}{$t[1]}) { # track TP
327                     $c++;
328                     $tp++;
329                     #ID, e-value, bit score, %ident, alinglen,  1 (TP), QueryID, HitID
330                     push @scores, "$c $t[10] $t[11] $t[2] $NormLen 1 $t[0] $t[1]";
331                 } elsif (${$HUnd}{$t[0]} eq ${$HUnd}{$t[1]}) {  # exclude untermemined hits
    from further analysis
332                     $un++;
333                     next;
334                 } else {
335                     $c++;
336                     $fp++;
337                     #ID, e-value, bit score, %ident, alinglen,  0 (FP), QueryID, HitID
338                     push @scores, "$c $t[10] $t[11] $t[2] $NormLen 0 $t[0] $t[1] ";
339                 }
340                  $prevHit = $t[1];
341                  $prevQuer = $t[0];
342             }
343         }
344         if ($options{u}) {  # treat undetermined as FP
345             while (<M8>)
346                 {
347                 chomp;
348                 next if !$_; # Skip empty lines
349                 my @t=split /\s+/;
350                 my $NormLen = $t[3]/$SeqLen{$t[0]};
```

```
351                 next if ($t[1] eq $prevHit && $t[0] eq $prevQuer);
352                 next if ($t[0] eq $t[1]); #exlude self matches
353                 next if ($t[10] > $evalFilter); # exlude records based on evalue filter (option
    -e)
354
355
356                 if (${$HRef}{$t[0]} eq ${$HRef}{$t[1]}) {   # track TP
357                     $c++;
358                     $tp++;
359                     #ID, e-value, bit score, %ident, alinglen,  1 (TP), QueryID, HitID
360                     push @scores, "$c $t[10] $t[11] $t[2] $NormLen 1 $t[0] $t[1]";
361                 } else {
362                     $c++;
363                     $fp++;
364                     #ID, e-value, bit score, %ident, alinglen,  0 (FP), QueryID, HitID
365                     push @scores, "$c $t[10] $t[11] $t[2] $NormLen 0 $t[0] $t[1] ";
366                 }
367                  $prevHit = $t[1];
368                  $prevQuer = $t[0];
369             }
370         }
371
372
373         my @sorted = ();
374
375         print "Sorting hits...\n";
376
377         if ($options{s} eq "e") {
378             @sorted =  stable_fieldsort ['2n'], @scores; # sort the scores only by eval
379         } elsif ($options{s} eq "b") {
380             @sorted =  stable_fieldsort ['-3n'], @scores; # sort the scores
381         } else {
382             @sorted =  stable_fieldsort ['2n', '-3n', '-4n', '-5n'], @scores; # sort the scores
383         }
384
385         @scores =  (); # save some memory
386         # open SORTED, ">sorted.txt" or die;
387         # foreach my $line (@sorted) {print SORTED "$line\n"}
388         # close SORTED;
389
390
391
392
393         my $x =1;
394         my $stScore ="";
395         my $hitKey = "";
396         my $hitClass = "";
397         my @ranked = ();
398         my $switch = 0;
399         my ($prScore, $prHitKey, $prHitClass, $prX, $prQueryID, $prHitID, $QueryID, $HitID) = -
    1 ;
400         my $t = 0; # tie rank
401         my $midrank = 0 ;
402         my @ties = (); # hitKeys in tie
403         my $fpc = 0; # FP count
404         my $tpc = 0; # TP count
405         my $tc = 0; #ties count
406         my $thc = 0; # tied hits count
407         my $eval = 0; # tied hits EVALUE (switched back to this since bit scores doesn't work
    well with unified e-values)
408         my $prEval =0;
409
410
411         $_ = shift(@sorted);
412         #print "$_\n";
413         @_= split(/\s/);
414
415         if ($options{s} eq "e") {
416             $prScore = "$_[1]"
417             } elsif ($options{s} eq "b") {
418             $prScore = "$_[2]"
419             } else {
420             $prScore = "$_[1]$_[2]$_[3]$_[4]"
421         }
422
423         $prEval = $_[1];
```

```
424         $prHitKey=$_[0]; # grab hit identifier from string
425         $prHitClass = $_[5]; #is it TP or FP?
426         $prQueryID = $_[6];
427         $prHitID = $_[7];
428         $prX = 1;
429         $x = 2;
430
431
432         # store ties and tied hits count for each FP
433         open RTIES , ">$file.rties" or die "Unable to create $file.rties file: $!";
434         print RTIES "FPn\tTies\tHit in Ties\n";
435
436         print "Calculating midranks for all the False Positives...\n";
437
438         while (@sorted>0) {
439
440                 $_ = shift(@sorted);
441
442                 #print "$_\n";
443                 @_ = split(/\s/);
444                 $eval = $_[1];
445                 $hitKey=$_[0]; # grab hit identifier from string
446                 $hitClass = $_[5]; #is it TP or FP?
447                 $QueryID = $_[6];
448                 $HitID = $_[7];
449
450                 if ($options{s} eq "e") {
451                     $stScore = "$_[1]"
452                 } elsif ($options{s} eq "b") {
453                     $stScore = "$_[2]"
454                 } else {
455                     $stScore = "$_[1]$_[2]$_[3]$_[4]"
456                 }
457
458
459
460                 if ($prScore eq $stScore) {
461                     if (!$switch) {
462                         $t=$prX; # mark tie start rank
463                         $switch =1;
464                     }
465                     if (!$prHitClass) {
466                         push @ties, "$prX $prHitClass $prEval $prQueryID $prHitID";
467                         $fpc++; # count FP
468                         print RTIES "$fpc\t$tc\t$thc\n";
469                     } else {
470                         push @ties, "$prX $prHitClass $prEval $prQueryID $prHitID";
471                         $tpc++; # count TP
472                         # print RTIES "$fpc\t$tc\t$thc\n";
473                     }
474                     $thc++;
475                 } else {
476                     if ($switch) {
477                         $midrank = ($prX*($prX + 1) - $t*($t - 1)) / (2*$prX -2*$t + 2); #
    calculate midrank for tied hits
478                         while (@ties>0) {
479                             $_ = shift(@ties);
480                             @_ = split(/\s/);
481
482                             push @ranked, ("$_[0] $midrank $_[2] $_[3] $_[4] $_[1]");
483                         }
484                         if (!$prHitClass) {
485                             push @ranked, "$prX $midrank $prEval $prQueryID $prHitID
    $prHitClass";
486                             $fpc++; # count FP
487                             print RTIES "$fpc\t$tc\t$thc\n";
488                         } else {
489                             push @ranked, "$prX $midrank $prEval $prQueryID $prHitID
    $prHitClass";
490                             $tpc++; # count TP
491                             # print RTIES "$fpc\t$tc\t$thc\n";
492                         }
493                         $switch = 0;
494                         $tc++;
495                         $thc++;
496
```

```
497                     } else {
498                         if (!$prHitClass) {
499                             push @ranked, "$prX $prX $prEval $prQueryID $prHitID $prHitClass";
500                             $fpc++; # count FP
501                             print RTIES "$fpc\t$tc\t$thc\n";
502                         } else {
503                             push @ranked, "$prX $prX $prEval $prQueryID $prHitID $prHitClass";;
504                             $tpc++; # count TP
505                             # print RTIES "$fpc\t$tc\t$thc\n";
506                         }
507                     }
508                 }
509                 $prScore = $stScore;
510                 $prHitKey = $hitKey;
511                 $prHitClass = $hitClass;
512                 $prX = $x;
513                 $prEval=$eval;
514                 $prQueryID = $QueryID;
515                 $prHitID = $HitID;
516                 $x++;
517 
518                 #print "$fpc ";
519                 last if ($fpc>=$s+1 && !$switch);
520             }
521 
522 
523         close RTIES;
524 
525         # statistics for all hits in the file
526         print STATS "filename:$file\nArguments provided:\n$args\nTotal TP
    expected:$TotTP\nTP:$tp\tFP:$fp\tUND:$un\n";
527         print STATS "Number of ties: $tc\nNumber of hits in ties: $thc\n";
528 
529         close STATS;
530 
531      ######### OPEN OUTPUT FILES ##########
532 
533         open ROCS, ">$file.rocs" or die "Unable to create $file.rocs file: $!";
534         print ROCS "#AA\tn\tROCn\tHitRank\tTPcount\tU-score(t_i)\tE-
    value\tQueryID\tHitID\tHitClass\n";
535 
536 
537     ###################################
538         my $FPc = 0;
539         my $Usc = 0; # Wilcoxon U score
540         my $Rsc = 0; # Wilcoxon R score
541         my $ROCn = 0;
542         my $class = 0;
543         my $aa = 0;
544         print "Writing output data files...\n";
545         while (@ranked>0) {
546 
547             $_=shift(@ranked);
548             @_ = split(/\s/);
549             $aa++;
550             my $rank = $_[0];
551             my $midrank = $_[1];
552             my $eval = $_[2];
553             my $class = $_[5];
554             $FPc++ if (!$class); # FP count
555             my $TPc = $rank - $FPc; # calculate TP count
556             $Rsc += $midrank if (!$class);
557             $Usc = (!$class) ? $Rsc - ($FPc*($FPc +1))/2 : $Usc;
558             $ROCn = (!$class) ? (1/($FPc * $TotTP)) * $Usc : $ROCn ;
559             my $HitID = $_[4];
560             my $QueryID = $_[3];
561 
562             print ROCS
    "$aa\t$FPc\t$ROCn\t$rank\t$TPc\t$Usc\t$eval\t$QueryID\t$HitID\t$class\n";
563         }
564 
565         close ROCS;
566 
567     }
568     %SeqFam=();
569     %SeqSfam =();
```

```
    %SeqFld = ();
    %HomCounts = ();

}



    my $nfil = @cpfiles;
    my %plotsd = ();

    my $maxTP = '';

open ROCSD, ">ROCn.sd" or die "Unable to create ROCn.sd file: $!";

    foreach my $n (@n) {
        my %plotcm = ();
        my %plotcm1 = ();
        my %plotcm2 = ();


        my $xmax = $n;
        foreach my $file (@cpfiles) {


        my (%TPc, %ROCn, %ROC, %rank, %Usc, %Eval, %QueryID, %HitID, %FPn, %Class) = ();

        my $l =0;
        my $sd = 0;

        my $filNmb =  $#cpfiles +1;
        $file =~ /.+?\.(\w+)[\.noLCR]*\.m8/; ## Change file extension to match given files
        my $mode = $1;


        open RDATA, "<$file.rocs" or die "Unable to open $file.rocs file: $!";
        readline <RDATA>;
        my $maxN = 0;
        while (<RDATA>) {
            chomp;
            @_ = split(/\t/);
            $FPn{$_[1]} = $_[0];
            $ROCn{$_[0]} = $_[2];
            $TPc{$_[0]} = $_[4];
            $maxTP = $_[4] if $maxTP <$_[4];
            $Usc{$_[0]} = $_[5];
            $QueryID{$_[0]} = $_[7];
            $HitID{$_[0]} = $_[8];
            $Class{$_[0]} = $_[9];
            $maxN = $_[1];
        }

        close RDATA;

        #foreach my $n (@n) {

        $maxN = $n if ($FPn{$n});

        #calculate standard deviation for each n
        for (my $i=1; $i<= $maxN; $i++) {
                $sd +=  ($Usc{ $FPn{$maxN}+1 }-$Usc{ $FPn{$i} } )**2;
            }


        $sd = sqrt($sd)/($maxN*$TotTP);
        $sd = $sd/sqrt($maxN*$TotTP);

        $plotsd{$mode} .= qq($maxN\t$ROCn{ $FPn{$maxN} }\t$sd\n);



        my $dd = $ls{$mode};

        open SCRIPT, ">batch$n.gs" or die;

        print SCRIPT qq{set terminal postscript eps colour enhanced "Times-Roman" 12\n};
        print SCRIPT "set output 'ROC$n" . ".eps'\n";
```

```
646         foreach my $key (keys %colors) {print SCRIPT qq{$colors{$key}\n}};
647
648         $plotcm{$mode} = qq{'$file.rocs' using 2:3 every 100 title "$mode" with line ls $dd};
649         $plotcm1{$mode} = qq{'$file.rocs' using 2:5 every 100 title "$mode" with line ls $dd};
650         $plotcm2{$mode} = qq{'$file.rocs' using 2:7 every 100 title "$mode" with line ls $dd};
651         }
652         print SCRIPT qq{
653 set key at screen 1.1 , 0.3 Left title 'Legend'  box ls -1
654 set lmargin 0
655 set rmargin 0
656 set tmargin 0
657 set bmargin 0
658 set multiplot
659 set size 0.9, 0.8
660 set border 3
661 set title "ROCn values for first $n FP hits" font "Helvetica, 14"
662 set xtics border out nomirror
663 set xlabel "Count of FP hits"
664 set xrange [1:$xmax]
665 set logscale x
666 set ytics border out nomirror
667 set ylabel "ROCn"\n};
668
669         my $stplotcm = "plot ";
670         foreach my $key (sort hashValueAscendingNum (keys(%ls))) {
671         $stplotcm .= $plotcm{$key} . ", " if (defined $plotcm{$key});
672         print "$key\n" if (defined $plotcm{$key});
673
674         }
675         my $com = rindex($stplotcm, ",");
676         print SCRIPT substr($stplotcm,0, $com);
677
678         print SCRIPT qq{
679         unset key
680         set size 0.9, 0.8
681         set origin 0, 0.95
682         set title "TP counts for first $n FP hits" font "Helvetica, 14"
683         set yrange [0:$maxTP]
684         set ylabel "Count of TP hits"
685         set logscale x\n};
686
687
688         $stplotcm = qq {plot (x*99) notitle with line ls 100, (x*9) notitle with line ls 100,
    (x*4) notitle with line ls 100, };
689
690         foreach my $key (sort hashValueAscendingNum (keys(%ls))) {
691         $stplotcm .= $plotcm1{$key} . ", " if (defined $plotcm1{$key});
692         }
693
694         $com = rindex($stplotcm, ",");
695         print SCRIPT substr($stplotcm,0, $com);
696
697         print SCRIPT qq{
698         set size 0.9, 0.8
699         set origin 0, 1.9
700         set title "E-values for first $n FP hits" font "Helvetica, 14"
701         set ylabel "E-value"
702         set logscale y
703         set logscale x
704         set yrange [*:*]
705         };
706
707         # $stplotcm = qq {plot (x/$dbSize) notitle with line 0, };
708         $stplotcm = qq {plot };
709
710         foreach my $key (sort hashValueAscendingNum (keys(%ls))) {
711         $stplotcm .= $plotcm2{$key} . ", " if (defined $plotcm2{$key});
712         }
713
714         $com = rindex($stplotcm, ",");
715         print SCRIPT substr($stplotcm,0, $com);
716         print SCRIPT qq{\nunset multiplot};
717         close SCRIPT;
718 my $cmd = "gnuplot batch$n.gs";
719         #print "##$cmd\n";
720 my $status = system( $cmd );
```

```
 if ( $status ) {
     warn( "$cmd\nfinished with status $status\n" );
     }

}

foreach my $key (sort hashValueAscendingNum (keys(%ls))) {
    if (defined $plotsd{$key}) {
        print ROCSD "#$key\n";
        print ROCSD $plotsd{$key} ;
        print ROCSD "\n\n";
    }
}
close ROCSD;

###################3

open SCRIPT, ">SDs.gs" or die;

print SCRIPT qq{set terminal postscript eps colour enhanced "Times-Roman" 12\n};
print SCRIPT "set output 'SDs.eps'\n";
foreach my $key (keys %colors) {print SCRIPT qq{$colors{$key}\n}};
print SCRIPT qq{
set key outside bottom center horizontal title 'Legend'  box ls -1
set border 3
set title "ROCn values" font "Helvetica, 14"

set ytics border out nomirror
set xtics border out nomirror
set ylabel "ROCn"\n};

my $inx = 0;
my $plotcmd = "plot";
foreach my $key (sort hashValueAscendingNum (keys(%ls))) {

    if (defined $plotsd{$key}) {

    $plotcmd .= qq[ 'ROCn.sd' index $inx title "$key" with errorbars ls $ls{$key}, ];
    $inx++;

    }
}
my $com = rindex($plotcmd, ",");

print SCRIPT substr($plotcmd,0, $com);

my $cmd = "gnuplot SDs.gs";
        print "##$cmd\n";
my $status = system( $cmd );
 if ( $status ) {
     warn( "$cmd\nfinished with status $status\n" );
 }
```