BetaSearch: a new method for querying $\beta$-residue motifs

Supplementary Materials

Hui Kian Ho$^*$ Graeme Gange Michael J. Kuiper Kotagiri Ramamohanarao

April 30, 2012

This document contains descriptions of the algorithms and query counts that were omitted from our manuscript entitled “BetaSearch: a new method for querying $\beta$-residue motifs”, for the purposes of brevity.

Should you have any questions regarding the manuscript or items in this document, you may contact me via email: hohkhkh1@csse.unimelb.edu.au.

1 3D substructure search queries

Figure 1 shows the histograms of the number queries and hits for each query size (every two query sizes are binned), generated from the ASTRAL95 dataset.

2 Exploratory querying of Top7

We translated each amino acid of our entire dataset of 209,127 $\beta$-matrices (from the PDB2011 dataset) to a reduced alphabet according to their hydrophobic (“h”) or hydrophilic (“p”) properties. We used a conventional amino acid grouping scheme [1] (SM-Table 2) to determine the amino acid translations. We subsequently formulated an “amphipathic” query as

$$\begin{pmatrix}
h & p & h & p \\ h & p & h & p \\ h & p & h & p \\ h & p & h & p \\
\end{pmatrix}. \quad (1)$$

This query returned 259 matching $\beta$-matrices from 116 protein structures, in which the Top7 protein [PDB:1QYS] was found with a sheet ID of 1QYSA SHEET 000.

The Top7 protein is of particular interest to protein researchers because it was the first (and currently, the only) engineered protein not to be derived from the structure or sequence of any other known protein [2]. The protein was designed using the RosettaDesign suite [3] ab initio then experimentally expressed and crystallised. The x-ray structure exhibited a remarkable 1.2Å similarity to the in silico model.

*corresponding author, hohkhkh1@csse.unimelb.edu.au
Figure 1: Histograms of the number of queries and hits for each query size in the ASTRAL95 dataset.
### L-trimers

|   | $x_0$ | $x_1$ | $x_2$ |
|---|-------|-------|-------|
| $y_0$ | .     | .     | .     |
| $y_1$ | .     | $b$   | $c$   |
| $y_2$ | .     | $a$   | .     |

$t.$ span$1$ (row-span): $(x_1, x_2)$  $(x_1, x_0)$  $(x_1, x_0)$  $(x_1, x_0)$
$t.$ span$2$ (col-span): $(x_1, x_2)$  $(x_1, x_0)$  $(x_1, x_0)$  $(x_1, x_0)$

### V-trimers

\[
\begin{array}{cccc}
  & a & . & c & . & a \\
  b & . & b & . & b & . \\
  c & . & a & . & a & . \\
\end{array}
\]

$t.$ span$1$ (row-span): $(x_1, x_0)$  $(x_1, x_2)$  $(x_1, x_0)$  $(x_1, x_0)$
$t.$ span$2$ (row-span): $(x_1, x_2)$  $(x_1, x_0)$  $(x_1, x_0)$  $(x_1, x_0)$

### H-trimers

\[
\begin{array}{cccc}
  a & b & c & a & b & a \\
  c & b & a & a & b & a \\
  . & . & . & . & . & . \\
\end{array}
\]

$t.$ span$1$ (col-span): $(x_1, x_0)$  $(x_1, x_2)$  $(x_1, x_0)$  $(x_1, x_0)$
$t.$ span$2$ (col-span): $(x_1, x_2)$  $(x_1, x_0)$  $(x_1, x_0)$  $(x_1, x_0)$

Figure 2: Trimer span assignments. The column and row indices are shown for the left-most L-trimer. The spans of symmetric trimers are interchangeable such that $t.$ span$1 = t.$ span$2$. 

---

3
Figure 3: An example of the trimer extraction process for a single $\beta$-graph.
Algorithm 1: **Build-Indices** constructs the $D$, $C$, and $R$ indices from a set of $\beta$-matrices $B$.

**Build-Indices($B$)**

1. For each $\beta$-matrix $b$ in the dataset $B$.
   (a) For each residue $r$ in $b$ and for each $t$ in $T$ where $T$ is the set of the trimers originating from $r$.
      i. If $t$ is not a V-trimer, set $R[\kappa R] \leftarrow R[\kappa R] \cup t$
      ii. If $t$ is not an H-trimer, set $C[\kappa C] \leftarrow C[\kappa C] \cup t$
      iii. Set $D[t.id] \leftarrow D[t.id] \cup t$.matrix-id
2. Return $D$, $R$, and $C$.

Algorithm 2: **First-Filter** returns the set of $\beta$-matrices that contain all the trimer IDs in the query.

**First-Filter($Q, D$)**

1. Set $C_1 \leftarrow \bigcap_{id \in Q.trimer-ids} D[id]$
   where $D[id] \rightarrow \{c \in D : id \in c.trimer-ids\}$.
2. Return $C_1$

Algorithm 3: **Second-Filter** returns the set of $\beta$-matrices that are congruent to the query. The ‘+’, ‘≪’, ‘⊕’, and ‘&’ operators in **Congruent** are the bitwise operators: OR, left-shift, XOR, and AND.

**Congruent($Q, c$)**

```
begin
\psi_c \leftarrow 15 ; \hspace{1cm} \text{// Initialise with the symmetric V-trimer class number.}
forall id \in Q.trimer-ids do
    \psi_q \leftarrow 0
    forall q \in \{t \in Q.trimers : t.id = id\} do
        \psi_c \leftarrow \psi_q + (q.class \mod 16) \ll (q.orient \oplus t.orient)
        \psi_c \leftarrow \psi_c \& \psi_q
    if \psi_c = 0 then
        return False
    else
        return True
end
**Second-Filter($Q, C_1$)**

```

```
begin
    C_2 \leftarrow \{c \in C_1 : Congruent(Q, c)\}
    return C_2
end
```
Figure 4: The L- (top), V- (middle), and H-trimer (bottom) orientations. Each L-trimer has a unique orientation due to their asymmetry in both axes. V-trimers are symmetric in the y-axis, resulting in an orientation equivalency of 0 with 1; and 2 with 3. Similarly, H-trimers are symmetric in the x-axis, resulting in an orientation equivalency of 0 with 2; and 1 with 3. Trimers with a t.class of 15 or 31 are known as symmetric trimers because they are symmetric in both axes.
Algorithm 4: Make-Query-Graph returns a graph in which each edge denotes an overlap between two trimers.

**Make-Query-Graph**(Q)

1. *col-spans* and *row-spans* map unordered spans to lists of trimers where each list implies a clique in G.

2. For each trimer q in Q.trimers and span s in q.spans, where s is converted to an unordered span.
   (a) If s is a row span, set row-spans[s] ← row-spans[s] ∪ q.
   Otherwise, set col-spans[s] ← col-spans[s] ∪ q.

3. For each span s in col-spans, add an edge \((t_{src}, t_{des})\) to G for each pair of trimers in col-spans[s].

4. For each span s in row-spans, add an edge \((t_{src}, t_{des})\) to G for each pair of trimers in row-spans[s].

5. Return G.
Algorithm 5: BFS-Match.

BFS-Match\((G, S, q_{\text{root}}, t_{\text{src}})\)

1. Set \(matched \leftarrow \text{True}\).

2. For each pair of adjacent trimers \((q_{\text{src}}, q_{\text{des}})\) in a breadth-first traversal of \(G\) beginning at \(q_{\text{root}}\).
   
   (a) The orientation of \(t_{\text{des}}\) is 
   \[ \text{orient} \leftarrow \text{Rel-Orient} \oplus t_{\text{src}}.\text{ORIENT} \]
   
   (b) The overlapping span numbers between \(t_{\text{src}}\) and \(t_{\text{des}}\) are 
   \[ (s_{\text{src}}, s_{\text{des}}) \leftarrow \text{Overlap-Span-Nums}(q_{\text{src}}, q_{\text{des}}) \]
   
   (c) The overlap type between \(t_{\text{src}}\) and \(t_{\text{des}}\) is 
   \[ \text{ol-type} \leftarrow \text{Overlap-Type}(q_{\text{src}}, q_{\text{des}}) \]
   
   (d) The span in \(t_{\text{src}}\) that overlaps with \(t_{\text{des}}\) is 
   \[ \text{span}_{\text{src}} \leftarrow S[t_{\text{src}}][s_{\text{src}}] \]
   
   (e) Determine the following compound key values for \(t_{\text{des}}\)
   
   \[
   \begin{align*}
   \text{span}_{\text{des}} & : \text{ the span in } t_{\text{des}} \text{ that overlaps with } t_{\text{src}} \\
   \text{coord} & : \text{ the row or column coordinate of } t_{\text{des}} \\
   I & : \text{ the row or column span index in which to find } t_{\text{des}}
   \end{align*}
   \]
   
   where each value is calculated as
   
   \[
   (\text{span}_{\text{des}}, \text{pos}, I) = \begin{cases} 
   (\text{span}_{\text{src}}, t_{\text{src}}.\text{ROW}, R) & \text{if } \text{ol-type} = \text{Bridge} \\
   (\text{span}_{\text{src}}^{-1}, t_{\text{src}}.\text{ROW}, R) & \text{if } \text{ol-type} = \text{Opp-Bridge} \\
   (\text{span}_{\text{src}}, t_{\text{src}}.\text{COL}, C) & \text{if } \text{ol-type} = \text{Peptide} \\
   (\text{span}_{\text{src}}^{-1}, t_{\text{src}}.\text{COL}, C) & \text{otherwise.}
   \end{cases}
   \]
   
   (f) The span index compound key for \(t_{\text{des}}\) is 
   \[ \text{key} \leftarrow (c.\text{SHEET-ID}, q_{\text{des}}.\text{ID}, q_{\text{des}}.\text{EQUIV-ORIENTS}, \text{coord}, \text{span}_{\text{des}}) \]
   
   (g) Set \(t_{\text{des}} \leftarrow I[\text{key}] \).
   
   (h) If \(t_{\text{des}} = \emptyset\), set \(matched \leftarrow \text{False}\) and break out of the loop.
   
   (i) If \(s_{\text{des}} = 1\), set \(S[t_{\text{des}}][2] \leftarrow t_{\text{des}}.\text{SPAN2}\).
   
   Otherwise, set \(S[t_{\text{des}}][1] \leftarrow t_{\text{des}}.\text{SPAN1}\).

3. Return \(matched\).
Algorithm 6: Verify-Candidate.

Verify-Candidate\( (G, S, q_{\text{root}}, t_{\text{src}}) \)

1. Set \( \text{matched} \leftarrow \text{True} \).
2. For each \( t_{\text{src}} \) in \( \text{c.trimers} \) where \( t_{\text{src}}.\text{id} = q_{\text{root}}.\text{id} \).
   \( \text{a) The span table } S \text{ maps a trimer and a span number to a span.} \)
   Set \( (S[t_{\text{src}}][1], S[t_{\text{src}}][2]) \leftarrow (t_{\text{src}}.\text{span1}, t_{\text{src}}.\text{span2}) \).
   \( \text{b) For each combination of span numbers in } S, \text{ swap } S[t_{\text{src}}][1] \text{ with } S[t_{\text{src}}][2] \text{ after the first iteration if } t_{\text{src}} \text{ is a palindromic trimer.} \)
   \( \text{This step is required because the span numbers for symmetric trimers are not well-defined, so the only way to obtain the “correct” span number assignments is to examine both combinations.} \)
   \( \text{i. Set } \text{matched} \leftarrow \text{BFS-Match}(G, S, c, q_{\text{root}}, t_{\text{src}}). \)
   \( \text{ii. If } \text{matched} = \text{False}, \text{ break out of the loop.} \)
   \( \text{iii. If } t_{\text{src}} \text{ is an asymmetric trimer, break out of this loop after the first iteration.} \)
   \( \text{This occurs because an asymmetric trimer has only one possible span number combination.} \)
\( \text{c) If } \text{matched} = \text{True}, \text{ set } C_3 \leftarrow C_3 \cup c \text{ and Return True.} \)
3. Return False.

Algorithm 7: Verify.

Verify\( (Q, C_2) \)

1. Build the overlap graph
   \( G \leftarrow \text{MAKE-QUERY-GRAH}(Q). \)
2. Set \( q_{\text{root}} \) to be a trimer in \( Q.\text{trimers} \) where \( q_{\text{root}}.\text{id} \) is the least frequently occurring trimer ID in \( Q.\text{trimer-ids} \) such that
   \[
   q_{\text{root}}.\text{class} = \begin{cases} 
   1 & \text{if } Q \text{ has L-trimers} \\
   3 \text{ or } 5 & \text{if } Q \text{ has asymmetric trimers} \\
   15 \text{ or } 31 & \text{otherwise.}
   \end{cases}
   \]
3. Set \( C_3 \leftarrow \{ c \in C_2 : \text{Verify-Candidate}(G, c, q_{\text{root}}) \} \)
4. Return \( C_3 \)
Table 1: Dataset query counts, the number of queries generated for each query size.

| Dataset size | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   |
|--------------|------|------|------|------|------|------|------|------|------|
| 1000         | 1000 | 999  | 827  | 965  | 824  | 722  | 798  | 717  | 675  | 678  |
| 2000         | 2000 | 1999 | 1678 | 1919 | 1675 | 1464 | 1611 | 1450 | 1348 | 1372 |
| 4000         | 4000 | 3999 | 3337 | 3853 | 3329 | 2957 | 3196 | 2930 | 2707 | 2793 |
| 8000         | 8000 | 7998 | 6725 | 7727 | 6718 | 6005 | 6461 | 5969 | 5513 | 5679 |
| 16000        | 16000| 15994| 13490| 15376| 13461| 11961| 12936| 11886| 10990| 11321|

Table 2: We used an existing amino acid grouping scheme \(\[1\]\) and considered the non-polar and aromatic residues to be hydrophobic (“h”) and others hydrophilic (“p”).

| Group         | Amino acids |
|---------------|-------------|
| non-polar     | G, A, V, L, I, P, M |
| polar (neutral)| C, N, Q, S, T |
| charged       | E, D, K, R, H |
| aromatic      | F, Y, W |

Algorithm 8: Helper procedures required for candidate verification.

**REL-ORIENT**(\(t_{\text{src}}, t_{\text{des}}\)) \(\rightarrow t_{\text{src}.\text{ORIENT}} \oplus t_{\text{des}.\text{ORIENT}}\)

**OVERLAPS**(\(t_{\text{src}}, t_{\text{des}}\)) \(\rightarrow\) True if \(t_{\text{src}}\) and \(t_{\text{des}}\) have a span that overlaps, False otherwise.

**OVERLAP-TYPE**(\(t_{\text{src}}, t_{\text{des}}\))

1. Set \((\text{span}_{\text{src}}, \text{span}_{\text{des}}) \leftarrow \text{OVERLAP-SPANS}(t_{\text{src}}, t_{\text{des}})\).
2. If \(\text{span}_{\text{src}}\) is a row span and \(\text{span}_{\text{src}} = \text{span}_{\text{des}}\), set \(\text{type} \leftarrow\) Bridge.
3. Otherwise, set \(\text{type} \leftarrow\) Opp-Bridge.
2. If \(\text{span}_{\text{src}}\) is a column span and \(\text{span}_{\text{src}} = \text{span}_{\text{des}}\), set \(\text{type} \leftarrow\) Peptide.
3. Otherwise, set \(\text{type} \leftarrow\) Opp-Peptide.
4. If \(\text{OVERLAPS}(t_{\text{src}}, t_{\text{des}}) = \text{False}\), set \(\text{type} \leftarrow\) No-Overlap.
5. Return \(\text{type}\).

**OVERLAP-SPAN-NUMS**(\(t_{\text{src}}, t_{\text{des}}\)) \(\rightarrow\) a tuple \((s_{\text{src}}, s_{\text{des}})\), in which \(s_{\text{src}}\) is the span number of \(t_{\text{src}}\) that overlaps with \(t_{\text{des}}\) and \(s_{\text{des}}\) is the span number of \(t_{\text{des}}\) that overlaps with \(t_{\text{src}}\).

**References**

[1] A. Kessel and N. Ben-Tal, *Introduction to proteins: structure, function, and motion*. CRC Press, 2011.

[2] J. J. Havranek, “Specificity in computational protein design,” *J Biol Chem*, vol. 285, pp. 31095–31099, 2010.

[3] Y. Liu and B. Kuhlman, “RosettaDesign server for protein design,” *Nucleic Acids Res*, vol. 34, pp. W235–W238, 2006.