String comparison problems, Myers (91)

- So far our goal was to maximize the alignment's similarity score
- Dual perspective: minimize the distance
- Intuitively: look for a minimal "number" of evolutionary operations (substitution,deletion,insertion) that would transform one sequence into the other
- Define $D(x, y) :=$ minimal cost to transform x to y
- Generalized-Levenshtein distance
- In case of the Levenshtein (edit) distance $\delta(a, b) = 1_{a \neq b}$, where $a, b \in \Sigma \cup \{-\}$
- Dual problem: longest common subsequence of x and y: $x_{k_i} = y_{l_i}$
- These problems arise in comparing contents of files and correcting spelling errors

String comparison problems (cont.)

- The 0-1 nature of the cost function allows some improvements
- Masek and Paterson (1980): subquadratic $O(mn/\log^2 n)$ (wlog $n \geq m$)
- Ukkonen (85) and Myers (86): $O(DN)$ where $D = D(x, y)$
  - The more similar are the sequences the faster it runs
  - D can be $O(m + n)$
  - Myers: the algorithm "expected" running time is $O(N + D^2)$

Approximate string matching

- Given a query pattern, a db, a scoring scheme and a threshold look for all words in the db that lie within the threshold distance to the query
- For example, the scoring is the edit distance
- Use an $m \times L$ (virtual) alignment cost table, where m is the size of the query and L is the db
- Based on Fickett (84): compute column by column going as deep as the previous column or the threshold
- If the text is "random" $O(DL)$ where D is the threshold

Searching for alignments against large databases

- Given that a guaranteed alignment costs $O(nm)$ it is impractical for frequent large db searches
- The two most popular heuristic tools are FASTA (88) and BLAST (90)
- Both try to rapidly locate promising starting points
- In the process the optimal alignment might get lost

Basic Local Alignment Search Tool

- First version of BLAST was written by Altschul, Gish, Miller, Myers and Lipman (90)
- A second version by Altschul et al. (97)
- There are many flavors of BLAST:
  - BLAST or blastp (AA query - AA db)
  - blastn (DNA query - DNA db)
  - blastx (DNA query - AA db: translate query in all 6 reading frames)
  - tblastn (AA query - DNA db: translate db in all 6 reading frames)
  - tblastx (DNA query - DNA db: translate query and db in all 6 reading frames)
- blastp and blastn are essentially the only two "real" variations

BLAST (cont.)

- BLAST1 was designed to find either a Maximal Segment Pair, a maximally scored local ungapped alignment
- or a list of High-scoring Segment Pairs
- Finding a combination of HSPs was a surrogate for doing a gapped alignment
- The main idea of BLAST is that a high scoring alignment should contain a high scoring aligned pair of words
- BLAST1 rapidly scans the db for an aligned pair of words (of fixed length w) that scores above a threshold T
- Any such word pair encountered (hit) is extended to an ungapped alignment which is recorded if it scores above S_0
- the expected number of random HSPs scoring above S_0 is about 10

All you wanted to know about BLAST (I)

- BLAST has 3 steps:
  - Given the query compile a list of all high scoring words:
    - let α_i denote the ith word of the query, then

$$L := \bigcup_i \underbrace{\{\beta \in \Sigma^w : S(\beta, \alpha_i) \geq T\}}_{N_T(\alpha_i) := T \text{ neighborhood of } \alpha_i}$$

    - How big/small can $N_T(\alpha_i)$ be?
    - typically 50 ($w = 4, T = 16$, PAM120)
    - Build an automaton that accepts the language L
    - Accepts on transition (Mealy) as opposed to accepts on states (Moore)
  - Using the automaton scan the db
    - Hash tables turned out to be slower

- Extend hits (aligned pair of words scoring above T)
  - Extension is attempted on both ends
  - Using "X dropoff" strategy: extend till the score drops by more than X from the best score observed so far
  - Dropoff parameter is "-y" (AA default 3, DNA is 11)
  - Over 90% of the execution time is spent at this step
  - An X-dropoff version of Smith-Waterman was tested but rejected as too costly for the marginal added sensitivity

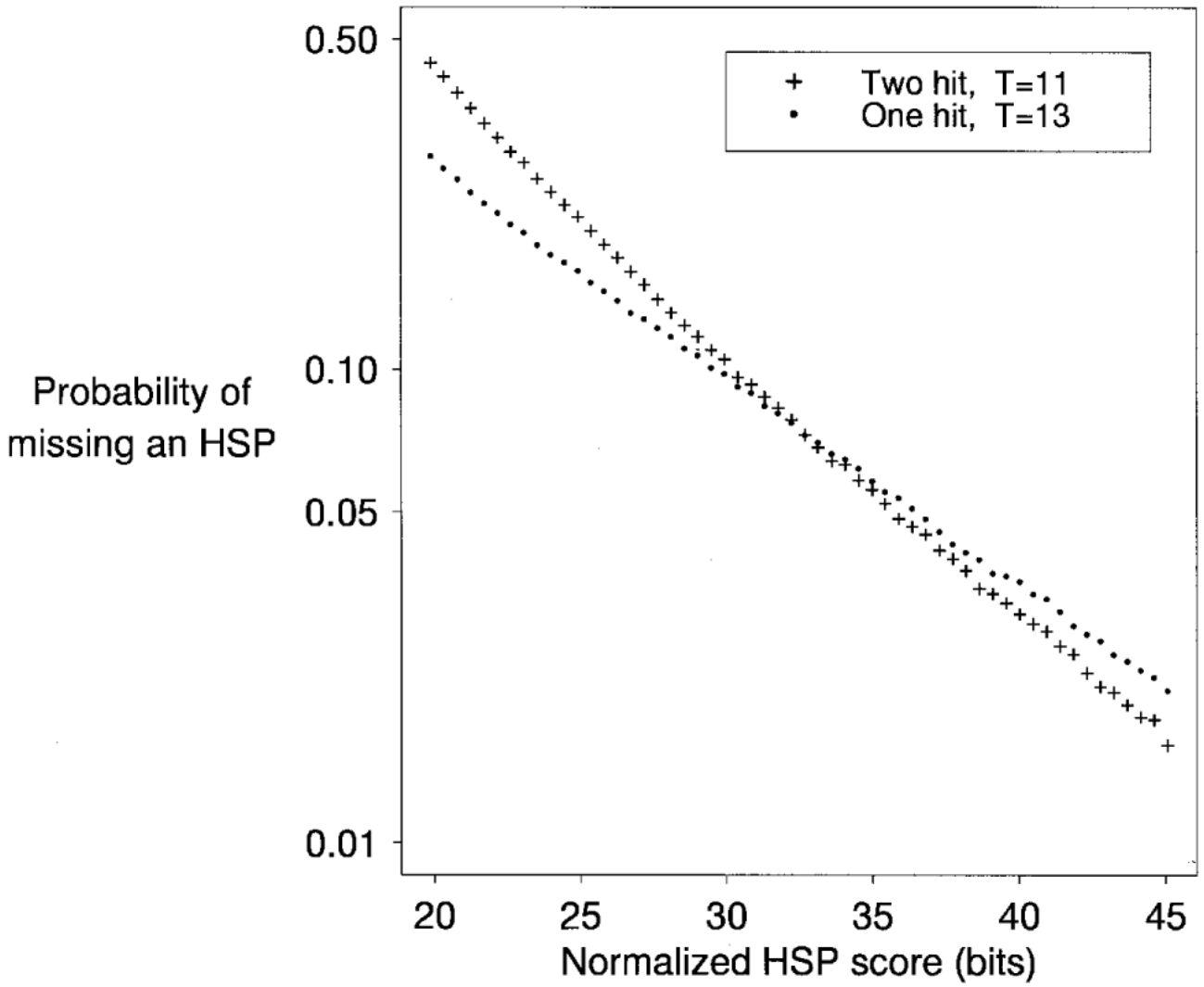
blastn

- In blastn $L = \cup_i \alpha_i$ and $w = 11, 12$
- Preprocessing: db is compressed by packing 4 nucleotides per byte
- If $w \geq 11$ then any hit would necessarily have an 8-mer that lies on a byte boundary
- The db is scanned *byte-wise* for hits of length two bytes
- What do we need to do with L?
- Special problems with DNA: locally biased base composition, repeats
- During the packing of the db, words that are significantly over-represented are stored for future filtering
- Before scanning the db repeat elements are removed from the query

All you wanted to know about BLAST (II)

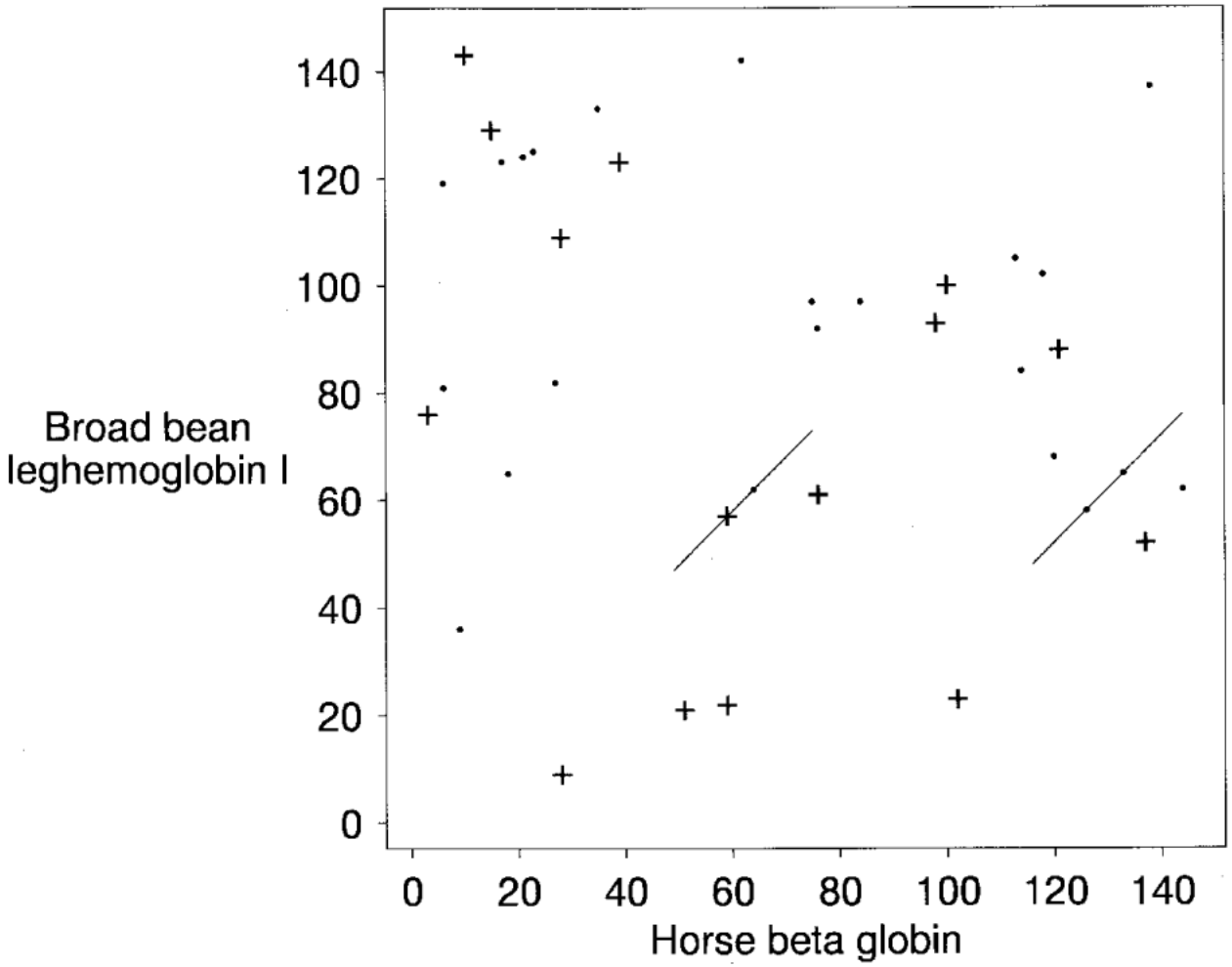
- Bottom line: "gapped BLAST" that is $3\times$ faster than blastp. How?
- Over 90% of the time blastp spends in extending seeds
- HSPs are usually longer than a single word pair
- Multiple word pairs can typically be detected in an HSP
- Therefore require two consistent hits (same diagnoal) to start an extension:
  - Two non-overlapping word pairs, each scoring above T
  - that lie at a distance of $\Delta \leq A$
  - Δ is the same for both sequences
- T by itself will decrease: more single word pair hits but fewer extensions as most of those will not form a consistent pair

sensitivity of two- vs. one-hit



HSPs were simulated using BLOSUM62,AA frequency from Robinson and Robinson (91),10^5 per nominal score 37-92

two- vs. one-hit example



15 hits ≥ 13 ('+'),additional 22 hits ≥ 11 ('.'),$A = 40$

Implementing the two-hit strategy

- The diagonal of a word pair that starts at (x_1, x_2) (db,query) is $x_1 - x_2$
- For each diagonal d, record x_1 of the last word pair that scores above T with $d = x_1 - x_2$
- When while scanning the db a new word pair hit is found at (x_1', x_2') check if the recorded x_1 under $d = x_1' - x_2'$ satisfies $x_1' - x_1 \leq A$
  - Note that $x_1' > x_1$
- Do we really need an array of the size of the db?
  - An array of size $3A$ would do $\pmod d$

Gain of the two-hit strategy

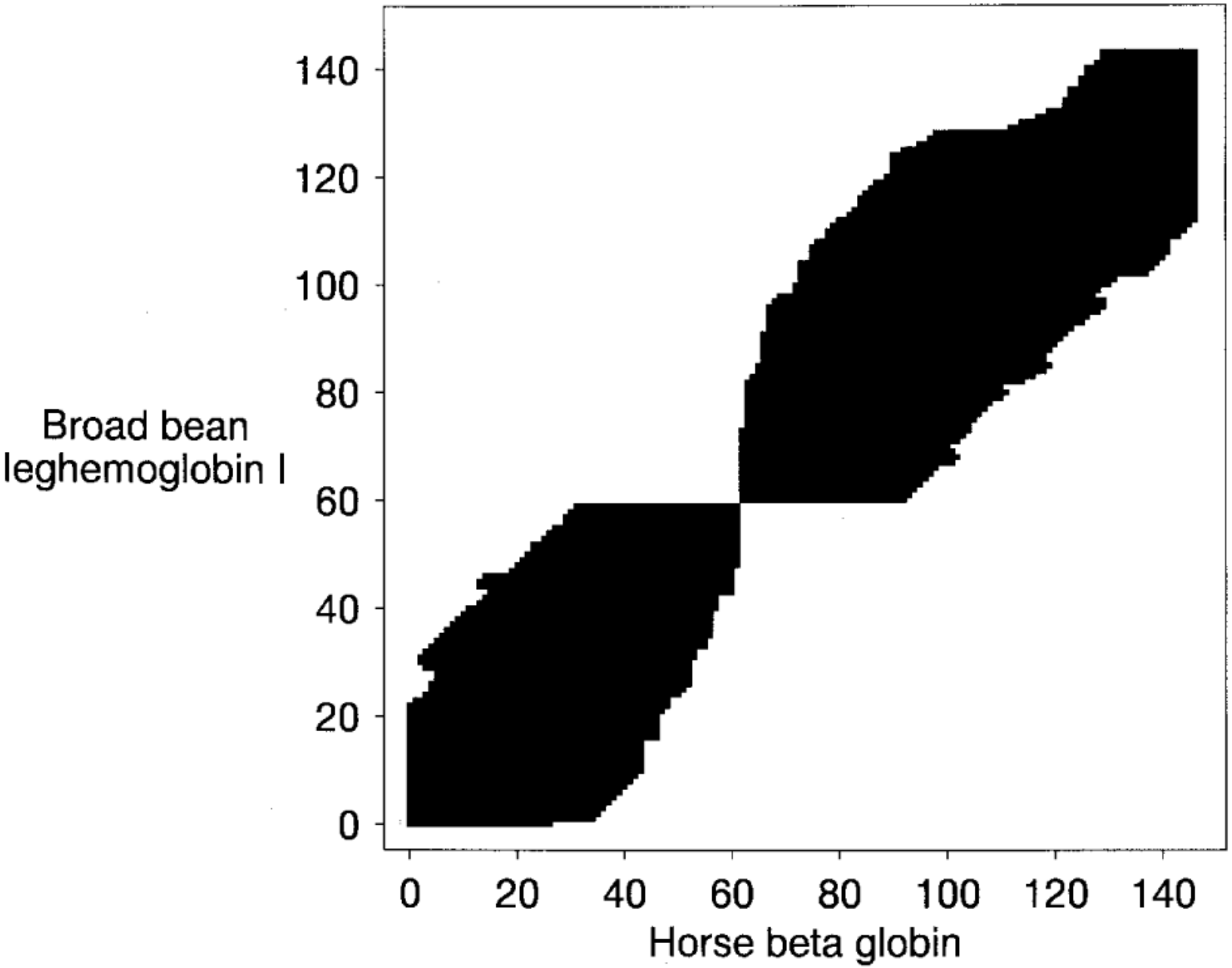
- Claim(?): using BLOSUM62, the R&R marginals, $w = 3$, $T_1 = 13$, $T_2 = 11$ and $A = 40$, on average there are about
  - 3.2 more single word hits using T_2
  - 7.14 more one-hit extensions than two-hit ones
- It is 9 times faster to test for a gapless extension than to do it
- Corollary: two-hits seed extension is about twice as fast on average
- How can we justify the claim?
  - MC simulation, or analytic
    - ▷ we have the distribution of $S(a, b)$ under H_0
    - ▷ find the distribution of $S(\alpha, \beta)$ (words of length $w = 3$)
    - ▷ find the probability of having two hits within $A = 40$ positions

Gapped alignments

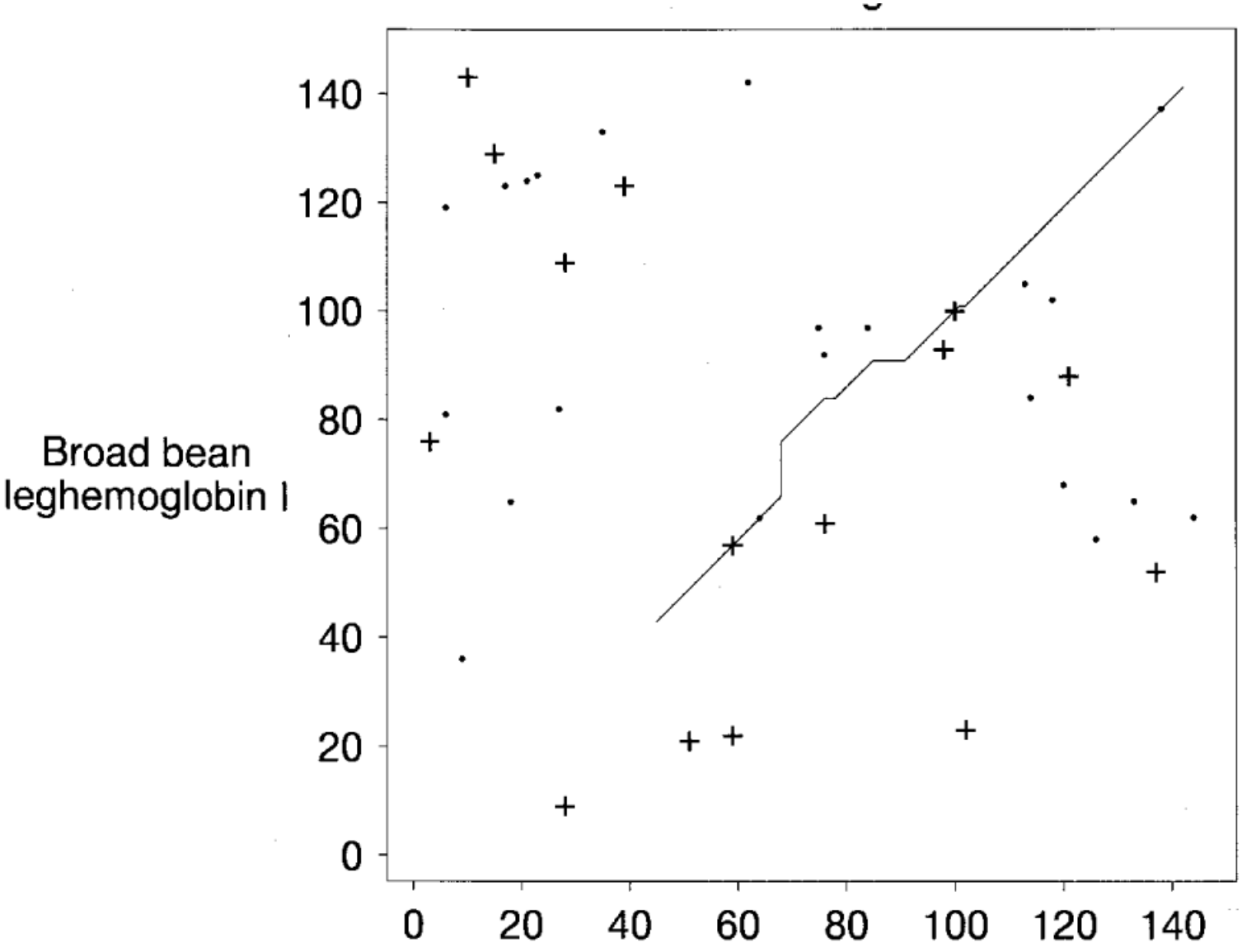
- With BLAST1 people would often set T much lower than needed for the probability of missing an HSP to be at most, say, 0.04
- The main reason was that BLAST1 would detect significant gapped alignments by fidning its HSPs
- Solution: perform X-dropoff ("-X, AA default 15") gapped extension on selected few HSPs
  - An HSP with score $\geq S_g$ is subjected to gapped extension
  - S_g is set so that such extension will occur once in about 50 db sequences

  It is important to choose a reasonably good initially aligned seed
  - Choose the central residue pair in an optimally scored segment of length 11

dropoff gapped extension



dropoff gapped extension - the alignment



deadend gapped extension