Faster algorithms for δ,γ-matching and related problems

P. Clifford, R. Clifford and C. S. Iliopoulos

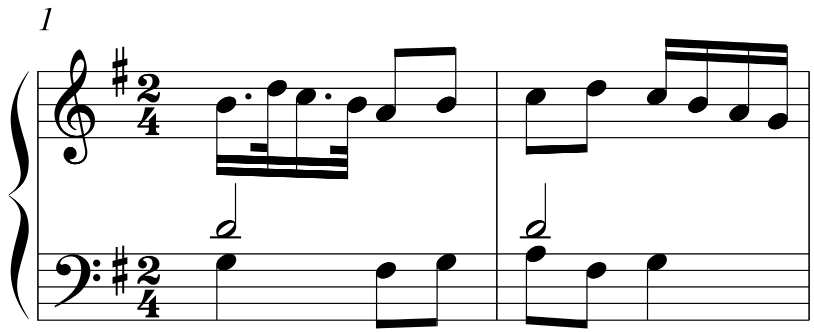
clifford@jesus.ox.ac.uk, {raph,costas}@dcs.kcl.ac.uk

Motivation

- The world is full of numeric data
- Much is inaccurate
- Approximate matching
- String matching algorithms for numerical data
- What do we do with non-integer data?

Musical Matching

A musical score may be represented numerically



W.A. Mozart, Rondo in G, K311, opening

```
Voice1:  71,74,72,71,69,71,72,74,72,71,69,67
Voice2:  62,62
Voice3:  55,54,55,57,54,55
```

Problems

δ **matching** Determine $\mathcal{I}_\delta$ where

$$\mathcal{I}_\delta = \{i : max_{j=1}^{m} |p_j - t_{i+j-1}| \leq \delta\}$$

γ **matching** Determine $\mathcal{J}_\gamma$ where

$$\mathcal{J}_\gamma = \{i : \Sigma_{j=1}^{m} |p_j - t_{i+j-1}| \leq \gamma\}$$

(δ, γ) **matching** Determine $\mathcal{I}_\delta \cap \mathcal{J}_\gamma$

Previous work

- δ matching can be reduced to two instances of the *less-than matching* problem giving an $O(n\sqrt{m \log m})$ time algorithm
- In this conference, Amir et al. have published an $O(n\sqrt{\gamma \log \gamma})$ time solution for γ matching when the data is integer valued

less-than matching Determine $\mathcal{L}$ where

$$\mathcal{L} = \{i : \forall j \bullet p_j \leq t_{i+j-1}\}$$

Our results

- $O(n\sqrt{m \log m})$ algorithm for γ matching and the total difference problem for non-integer data
- $O(\delta n \log m)$ algorithm for δ matching
- $O(\delta n \log m)$ algorithm for (δ, γ) matching

Background on Fourier methods

The most important property of the FFT is that, for numerical strings, all the inner-products,

$$p \cdot t^{(i)} = \sum_{j=1}^{m} p_j t_{i+j-1}, \quad 1 \leq i \leq n - m + 1,$$

can be calculated accurately and efficiently in $O(n \log m)$ time.

Two problems with simple solutions

1. We can express the total squared difference between p and $t^{(i)}$ as

$$\sum_{j=1}^{m} (p_j - t_{i+j-1})^2 = \sum_{j=1}^{m} p_j^2 - 2p \cdot t^{(i)} + \sum_{j=1}^{m} t_{i+j-1}^2$$

which gives an exact match when the sum is 0.

2. Exact matching with wildcards can be solved by calculating

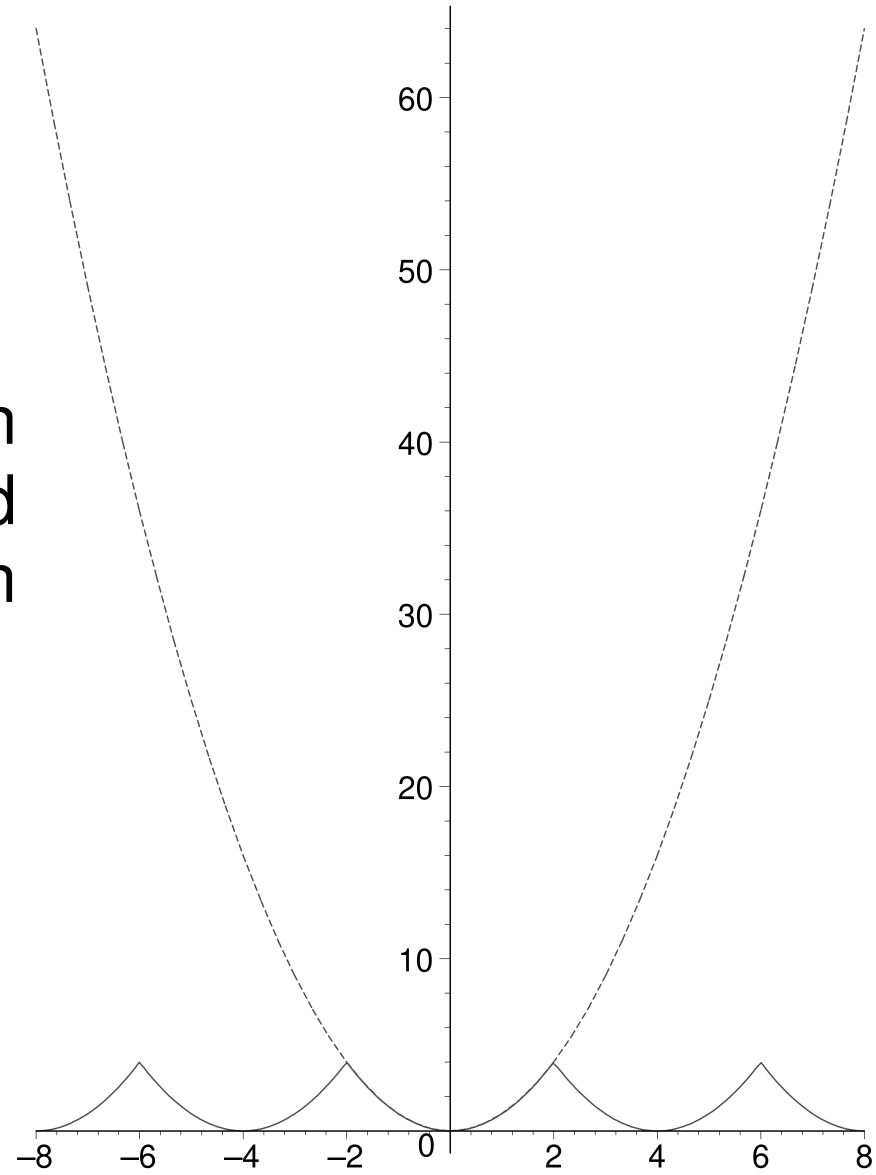
$$\sum_{j=}^{m} p_j t_{i+j-1}(p - t_{i+j-1})^2 = \sum_{j=1}^{m} p_j^3 t_{i+j-1} - 2p^2 \cdot t^{2(i)} + \sum_{j=1}^{m} p_j t_{i+j-1}^3$$

where the wildcard value is set to 0

Our techniques for δ matching

Define $f(x)$ to be an even periodic function defined over the whole of $\mathbb{Z}$ such that for $|x| \leq \delta$

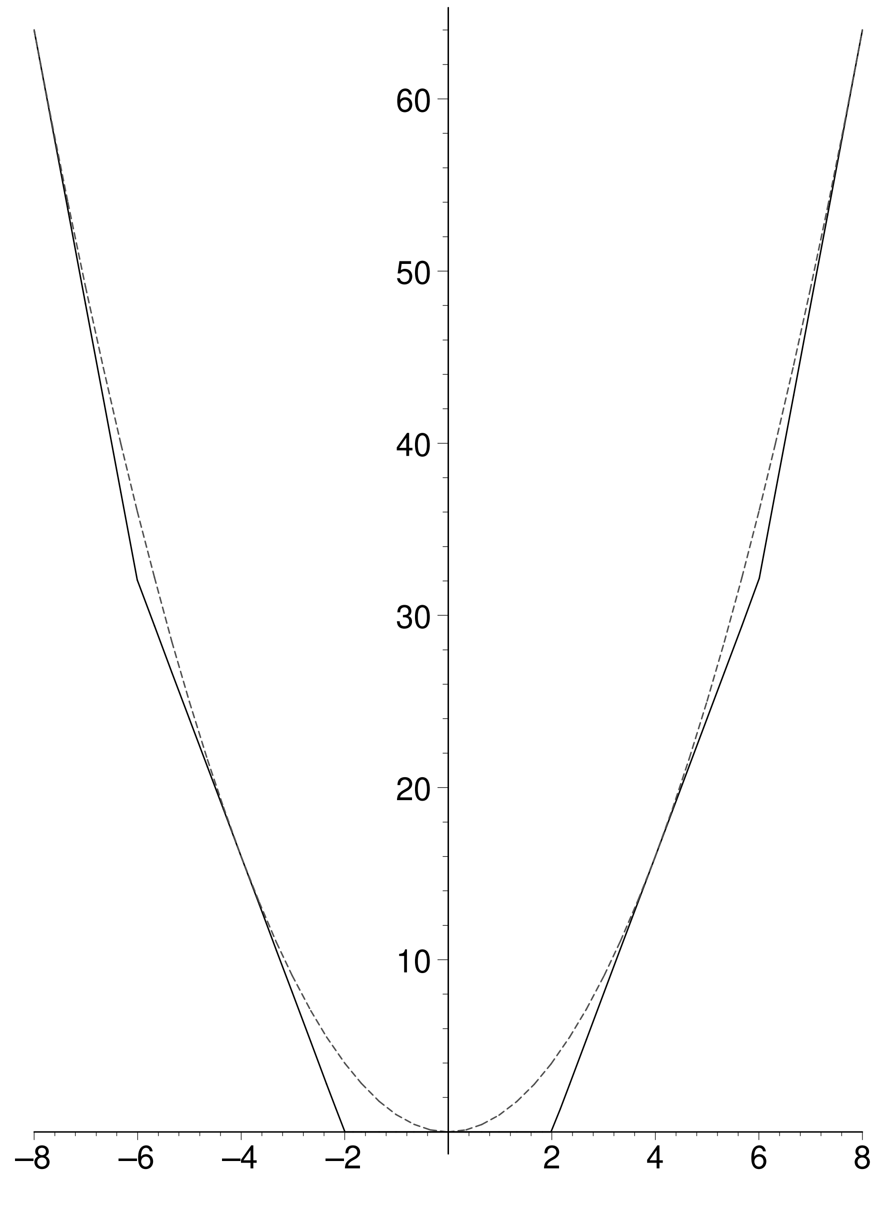
$$f(x) = x^2$$



Scoring function

Consider

$$g(x) = x^2 - f(x)$$



Scoring function

- $g(x) = 0$ if and only if $|x| \leq \delta$ and $g(x) \geq 1$ otherwise
- Therefore $g(x - y) = 0$ if and only if $|x - y| \leq \delta$ as required
- But g is even and periodic and so has a finite discrete cosine expansion

$$
\begin{aligned}
g(x-y) = x^2 + y^2 - 2xy - \alpha_0 r(0) \\
- \sum_{k=1}^{\delta} \alpha_k r(k) cos(xk\pi/\delta) cos(yk\pi/\delta) \\
- \sum_{k=1}^{\delta-1} \alpha_k r(k) sin(xk\pi/\delta) sin(yk\pi/\delta)
\end{aligned}
$$

Algorithm for δ matching

Calculate the α_k coefficients using the equation $\alpha_k = \sum_{x=1-\delta}^{\delta} x^2 h_k(x)$

1. Compute $A = \sum_{j=1}^{m} p_j^2$ and $B_i = \sum_{j=1}^{m} t_{i+j-1}^2$ for $1 \leq i \leq n-m+1$
2. Compute $C_i = p \cdot t^{(i)}$ for $1 \leq i \leq n-m+1$
3. For each $k \in \{1, \ldots, \delta\}$, construct a pair of arrays p' and t' such that $p'_j = \alpha_k r(k) c_k(p_j)$ and $t'_j = c_k(t_j)$ and compute $D_i(k) = p' \cdot t'^{(i)}$ for $1 \leq i \leq n-m+1$
4. For each $k \in \{1, \ldots, \delta-1\}$, construct a pair of arrays p' and t' such that $p'_j = \alpha_k r(k) s_k(p_j)$ and $t'_j = s_k(t_j)$ and compute $E_i(k) = p' \cdot t'^{(i)}$ for $1 \leq i \leq n-m+1$
5. $\sum_{j=1}^{m} g(p_j - t_{i+j-1}) = A + B_i - 2C_i - \alpha_0 r(0) m - \sum_{k=1}^{\delta} D_i(k) - \sum_{k=1}^{\delta-1} E_i(k)$

Total difference problem

- The difference between two numbers x and y can be calculated as the sum of four products

$$|x - y| = xI_xJ_y - I_xyJ_y + J_xyI_y - xJ_xI_y,$$

  where $I_a = 1$ if $a > \theta$ and $I_a = 0$ otherwise and $J_a = 1 - I_a$, provided x and y are on opposite sides of some arbitrary value θ.
- Otherwise the right hand side is zero
- Contributions to M_i from pairs of values (p_j, t_{i+j-1}) on opposite sides of specific thresholds are accumulated first

The algorithm-partition

- Sort $\{p_1, p_2, \ldots, p_m\}$ and call the array of sorted indices A
- Partition A from left to right into b successive arrays each of length m/b. Denote the kth of these arrays by A_k and let θ_k be its largest element
- Sort $\{t_1, t_2, \ldots, t_{2m}\}$ in increasing order and obtain a corresponding collection of arrays $\{B_k : k = 1, 2, \ldots, b\}$
- B_b contains all the indices j such that $t_j > \theta_{b-1}$ and B_k contains all the indices j such that $\theta_{k-1} < t_j \leq \theta_k$ for $k < b$

The algorithm-FFT

For $k = 1, \ldots, b$ and x a real number define two functions:

$$I_k(x) = \begin{cases} 1 & \text{if } x > \theta_k \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad J_k(x) = \begin{cases} 1 & \text{if } \theta_{k-1} < x \leq \theta_k \\ 0 & \text{otherwise.} \end{cases}$$

- For each k create four new text strings with jth elements $t_j I_k(t_j), I_k(t_j), J_k(t_j)$ and $t_j J_k(t_j)$
- Also four new patterns strings with jth elements $J_k(p_j), -p_j J_k(p_j), p_j I_k(p_j)$ and $-I_k(p_j)$

$$\sum_{j=1}^{m} |p_j - t_{i+j-1}| = \sum_{k=1}^{b} \sum_{\ell=1}^{4} \nu_{k\ell}(p) \cdot \epsilon_{k\ell}(t)^{(i)} + \text{remainder},$$

The algorithm - the remainder

- The remainder contains terms from pairs of values (p_j, t_{i+j-1}) where $p_j \in A_k$ and $t_{i+j-1} \in B_k$ for some k
- Working through each of the arrays B_k, for each element of the text the number of associated elements in the pattern is m/b
- The total number of such pairs considered is then $2m \times m/b$
- Contributions to the total differences can be calculated in a straightforward manner in $O(m^2/b)$ time

Running time

1. The running time for partitioning is $O(m \log m)$
2. The running time for the FFT stage is $4bO(m \log m)$
3. The running time for calculating the remainder is $O(m^2/b)$
4. Therefore, setting b to be $\sqrt{m/\log m}$ the overall running time is $O(m\sqrt{m \log m})$ when $n = 2m$
5. In general the running time is therefore $O(n\sqrt{m \log m})$

Discussion

- Can we determine in $o(nm)$ time

$$M_i = max_{j=1}^{m}|p_j - t_{i+j-1}| \text{ for } 1 \leq i \leq n-m+1$$

or

$$M_i = \Sigma_{j=1}^{m}(p_j - t_{i+j-1})^p \text{ for } 1 \leq i \leq n-m+1$$

- We can solve the *shift-matching* problem using the L_2 norm in $O(n \log m)$ time

$$\mathcal{L}_\delta = \{i : \min_{l_i \in \mathbb{Z}}\{\sum_{j=1}^{m}(p_j - t_{i+j-1} + l_i)^2\} \leq \delta$$

- What about real valued data for δ matching? Can we find an $o(nm)$ solution?