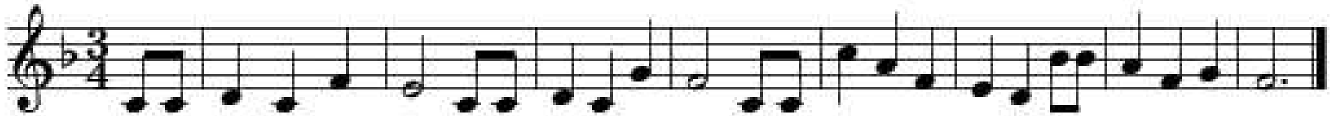
Geometric Algorithms for Transposition Invariant Content-Based Music Retrieval

Esko Ukkonen, Kjell Lemström, and Veli Mäkinen
The C-BRAHMS project
Department of Computer Science, University of Helsinki
{ukkonen,klemstro,vmakinen}@cs.helsinki.fi

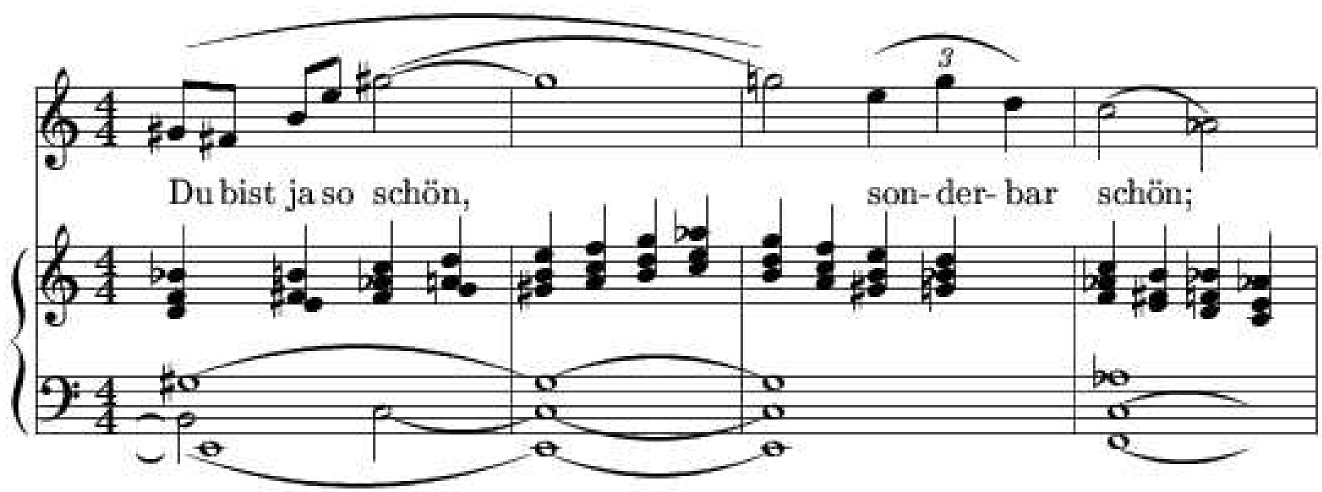
ISMIR'2003, October 26–30, Baltimore, USA.

The Task

Given the *pattern*, i.e. a short music excerpt of m notes



find whether it has transposed occurrences in a *source*: a large database of polyphonic music comprising n notes.



Straightforward Solution: Stringology

1. Encode music by using strings of pitches (or intervals).
2. Apply classical string matching methods (e.g. dynamic programming) separately for each monophonic voice.

Does not work if:

- too much musical decorations (noise) are present,
- voicing information is not available,
- the pattern may be distributed across the voices.

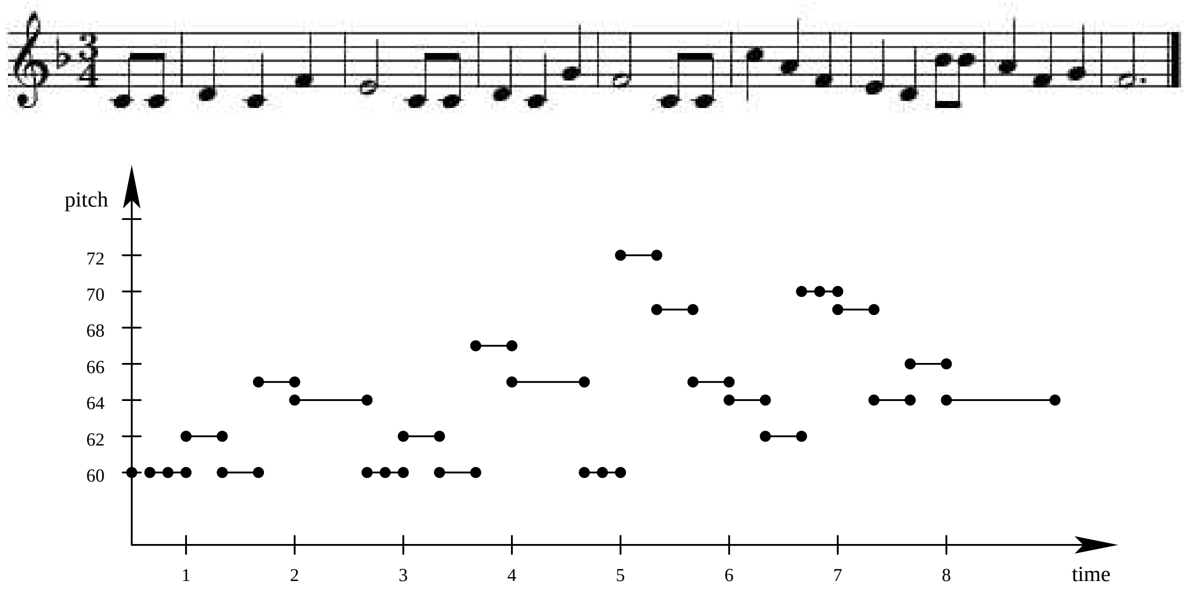
Geometric Representation of Music

We represent music by using *line segments* $[s, s']$, where

- $s = (s_x, s_y) \in \mathbb{R}^2$ is the *starting point*,
- $s' = (s'_x, s'_y) \in \mathbb{R}^2$ is the *ending point*,
- and $s_y = s'_y$ and $s_x \leq s'_x$.

The segment consists of the points between its 2 end points.

Example of geometric representation:



In the special case, when $s = s'$; segments become points in $\mathbb{R}^2$.

Definitions of the Problems

($P1$) Find translations of P such that all starting points of P match with some starting points of T.

($P2$) Find all translations of P that give a partial match of starting points of P among starting points of T.

($P3$) Find translations of P that give longest common shared time with T .

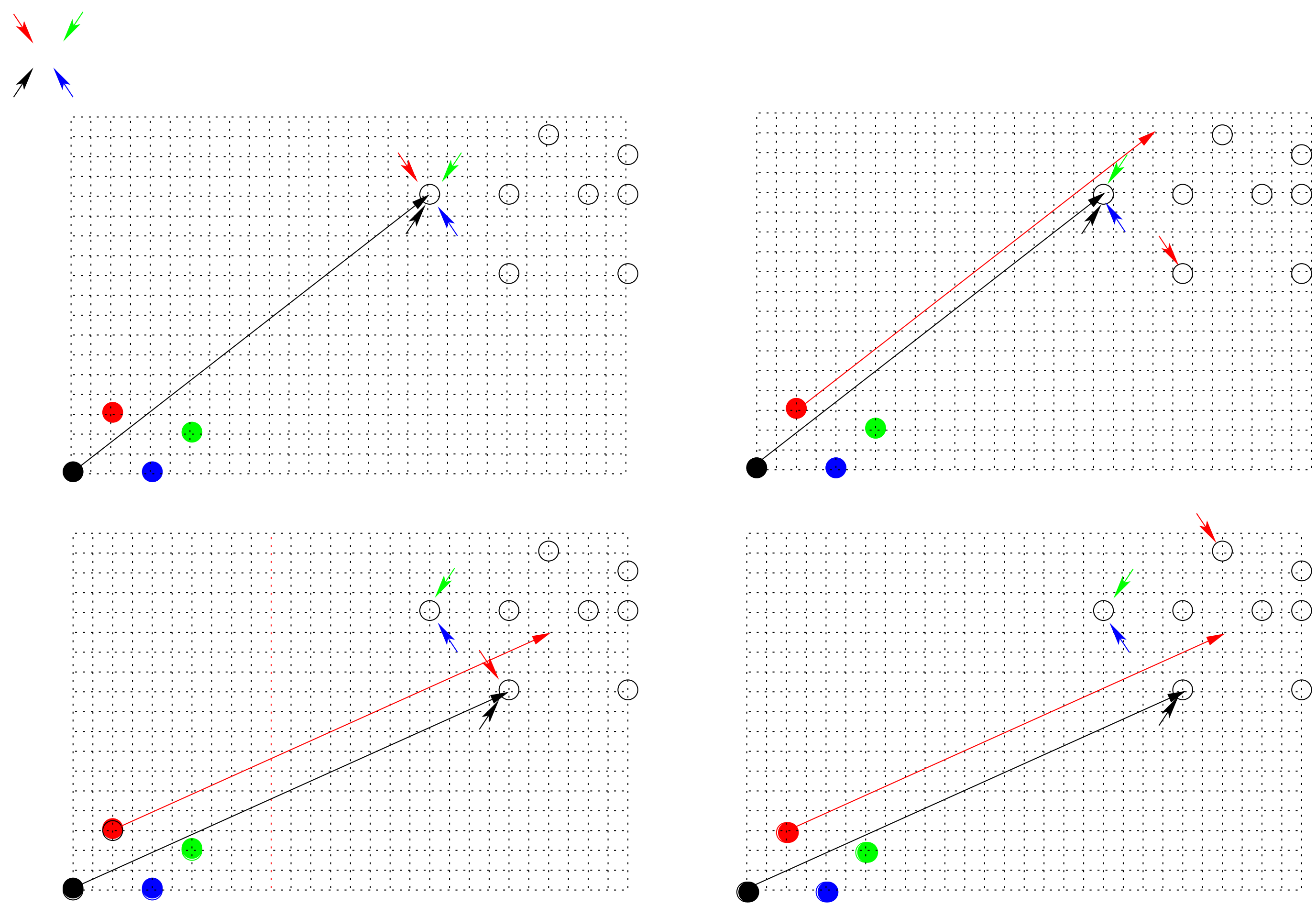
Complexities (worst case): best known vs. our

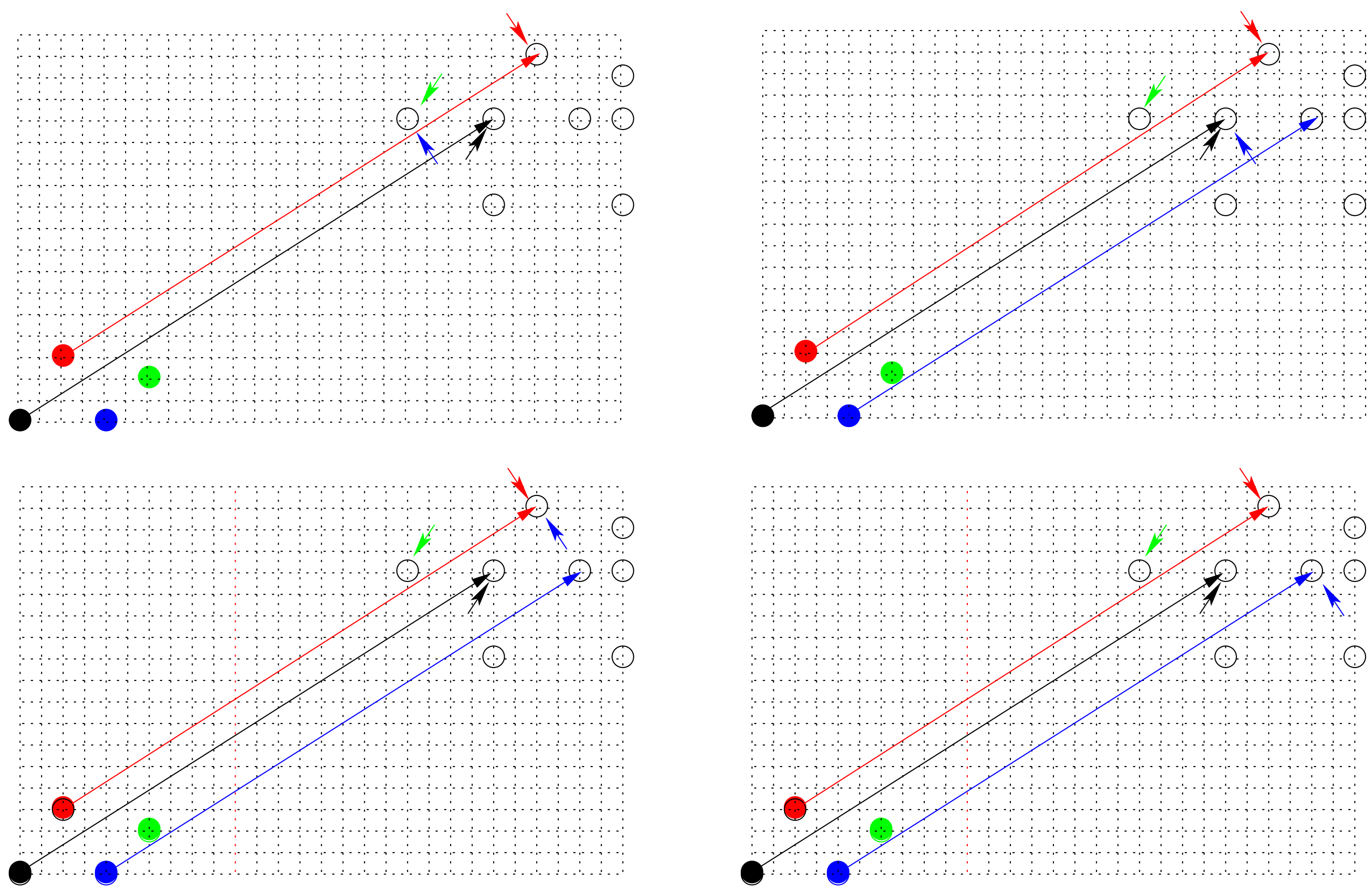
Problem	Time	Space
(P1) best known	$O(mn)^1$	$O(m)$
our	$O(mn)^1$	$O(m)$
(P2) best known	$O(mn\log(mn))$	$O(mn)$
our	$O(mn\log m)$	$O(m)$
(P3) best known	-	-
our	$O(mn\log m)$	$O(m+n)$

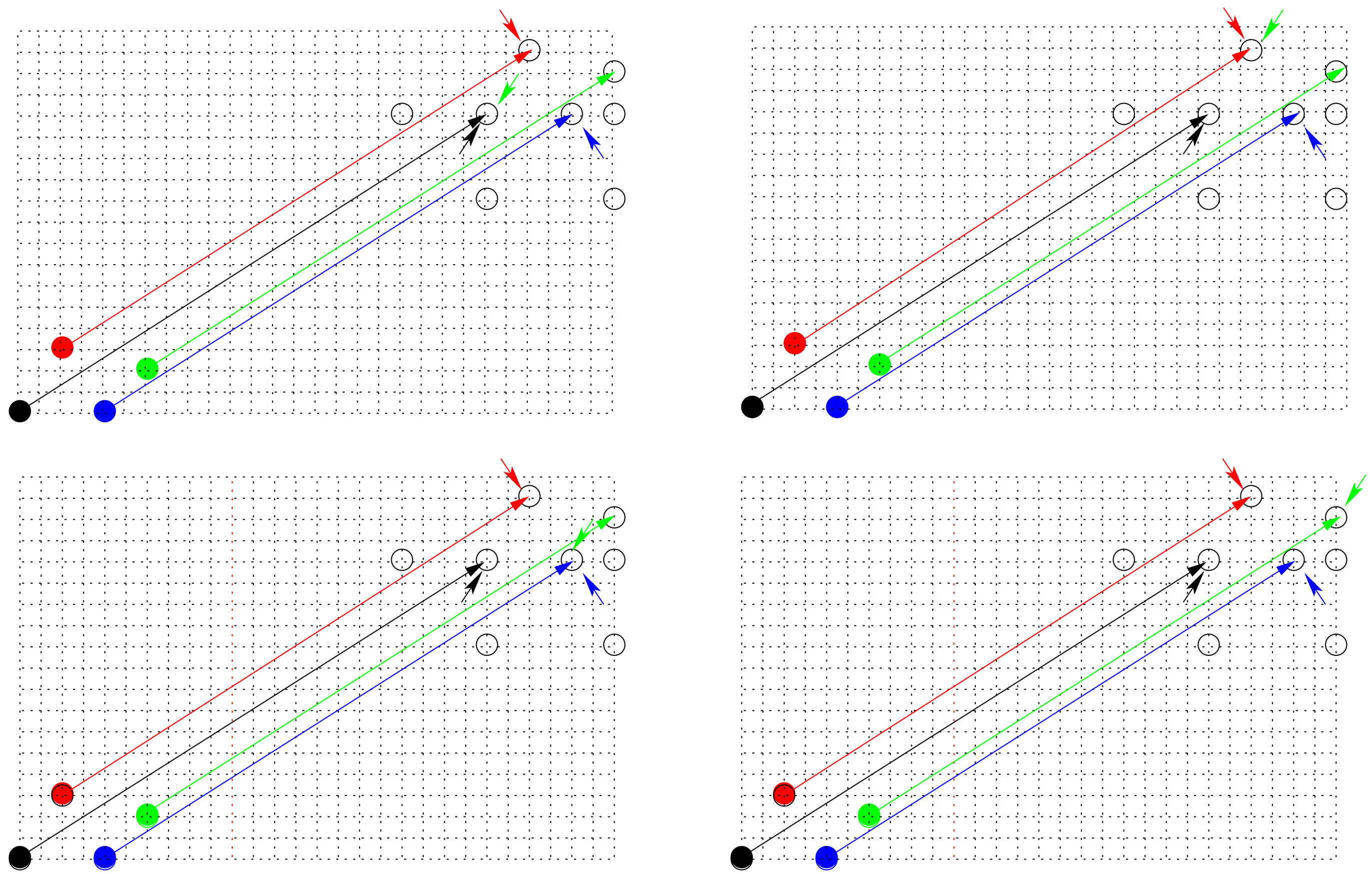
^{1} $O(n)$ on the average

Solving ($P1$)
("Total Matching")

- Notes as points in $\mathbb{R}^2$ (the special case).
- Generalize the naïve string matching algorithm
(see also Lemström and Tarhio 2000; Meredith et al. 2001):
  - Use m pointers, q_i, each pointing to events in T.
  - Function $next(q_i)$ gives the next element in lexicographic order in T.







Algorithm P1

(1) **for** $i \leftarrow 1, \ldots, m$ **do** $q_i \leftarrow -\infty$

(2) $q_{m+1} \leftarrow \infty$

(3) **for** $j \leftarrow 1, \ldots, n-m$ **do**

(4) $\quad f \leftarrow t_j - p_1$

(5) $\quad i \leftarrow 1$

(6) $\quad$ **do**

(7) $\quad\quad i \leftarrow i + 1$

(8) $\quad\quad q_i \leftarrow \max(q_i, t_j)$

(9) $\quad\quad$ **while** $q_i < p_i + f$ **do** $q_i \leftarrow next(q_i)$

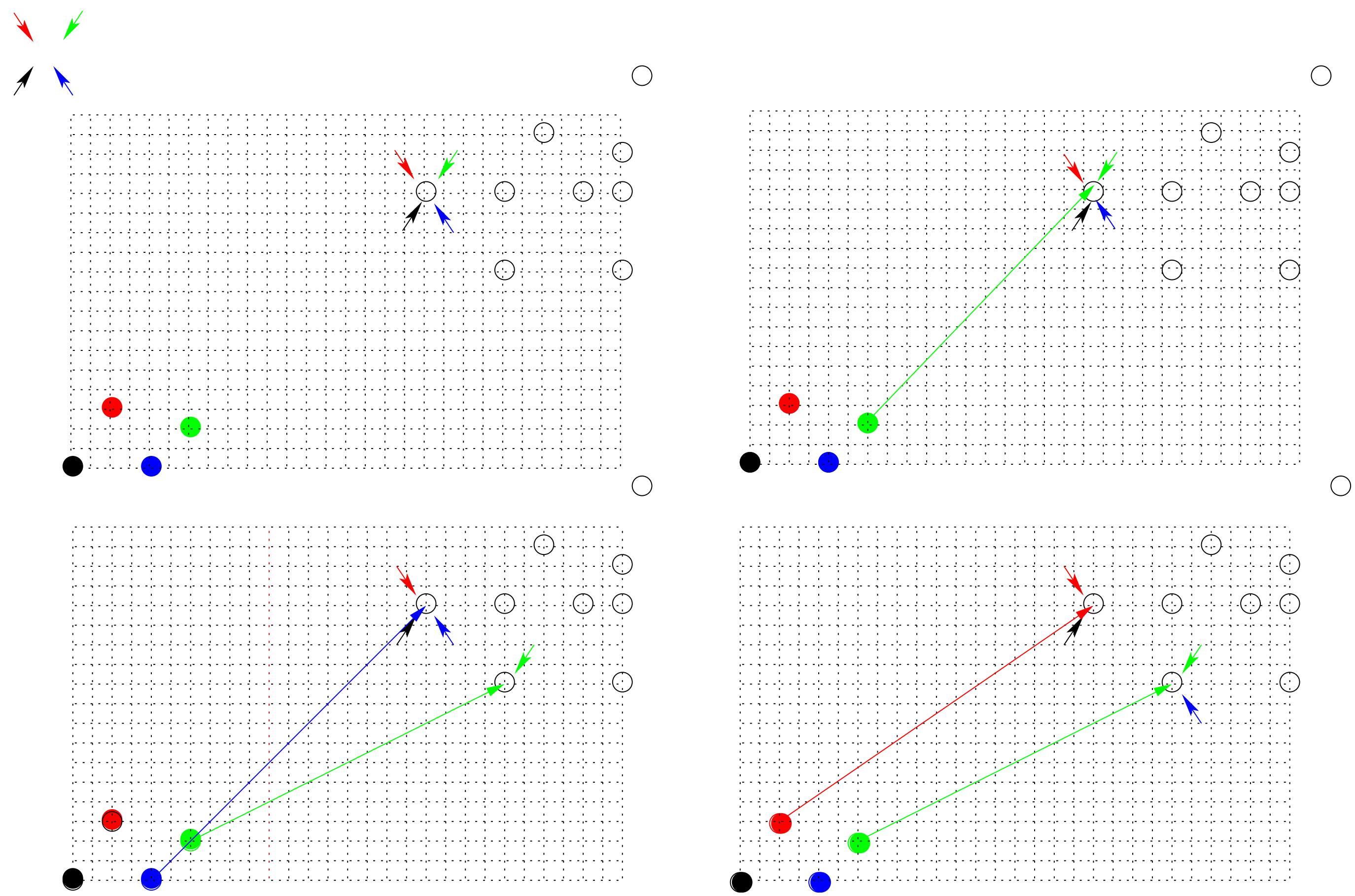
(10) $\quad$ **until** $q_i > p_i + f$

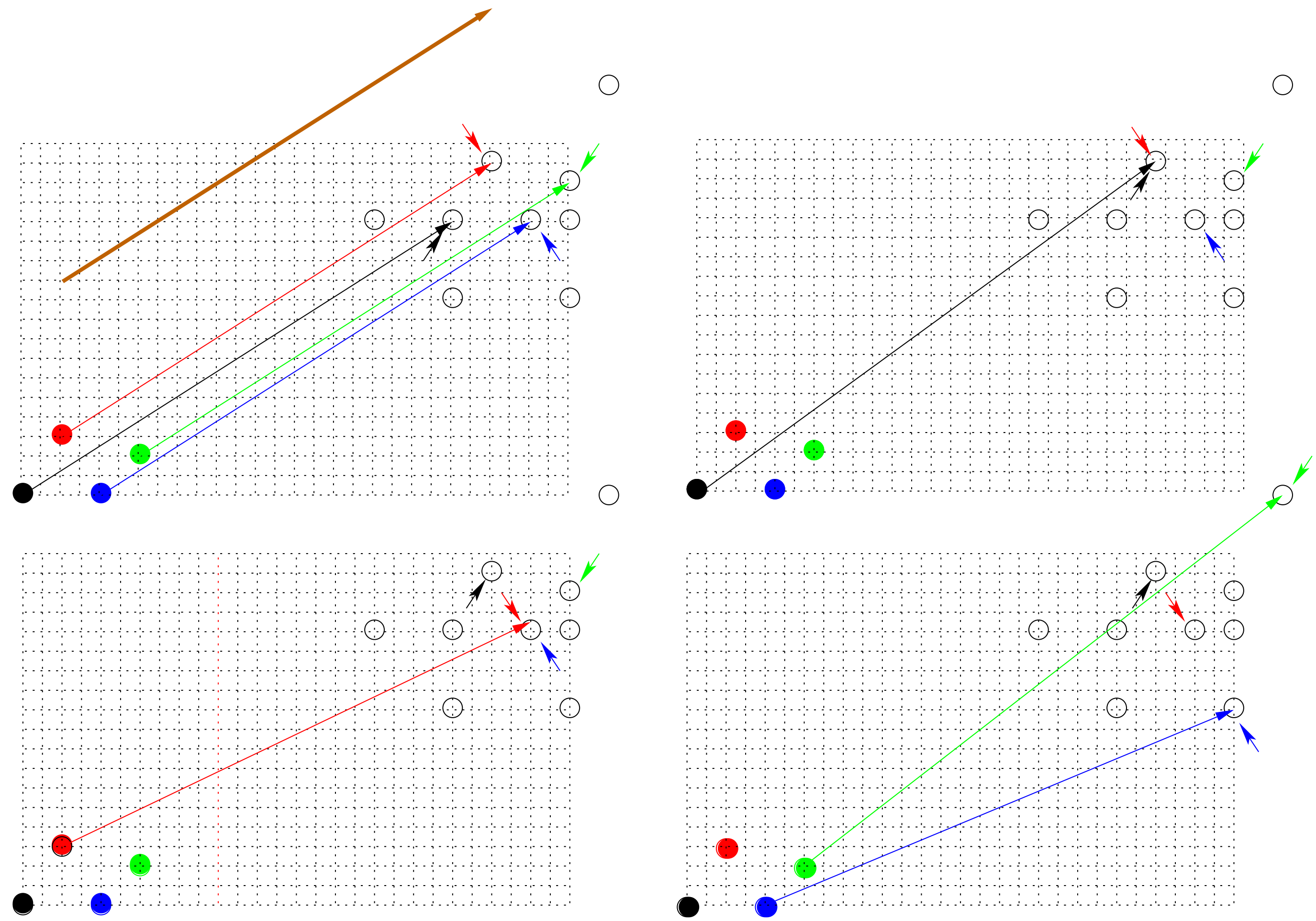
(11) $\quad$ **if** $i = m + 1$ **then** $output(f)$

(12) **end for**.

Solving ($P2$)
("Partial Matching")

- Notes as points in $\mathbb{R}^2$ (the special case).
- ($P2$): find translations f such that $(P + f) \cap T$ is nonempty.
- We call such $P + f$ a *partial occurrence* of P in T.
- We need:
  - m pointers q_i (as above);
  - a priority queue F (*min* queries and updating: $O(\log m)$);
  - and a counter c.





CoreOfP2

(1) $f \leftarrow -\infty$; $c \leftarrow 0$;

do

(2) $f' \leftarrow min(F)$; $update(F)$

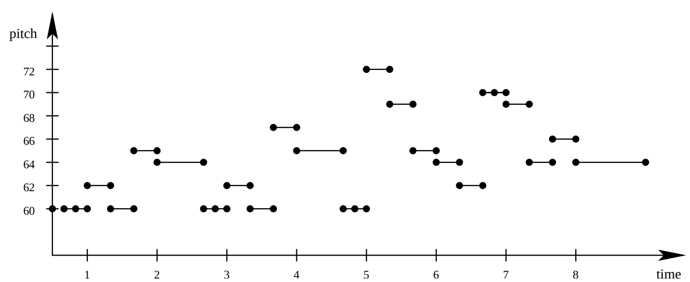
(3) **if** $f' = f$ **then** $c \leftarrow c + 1$

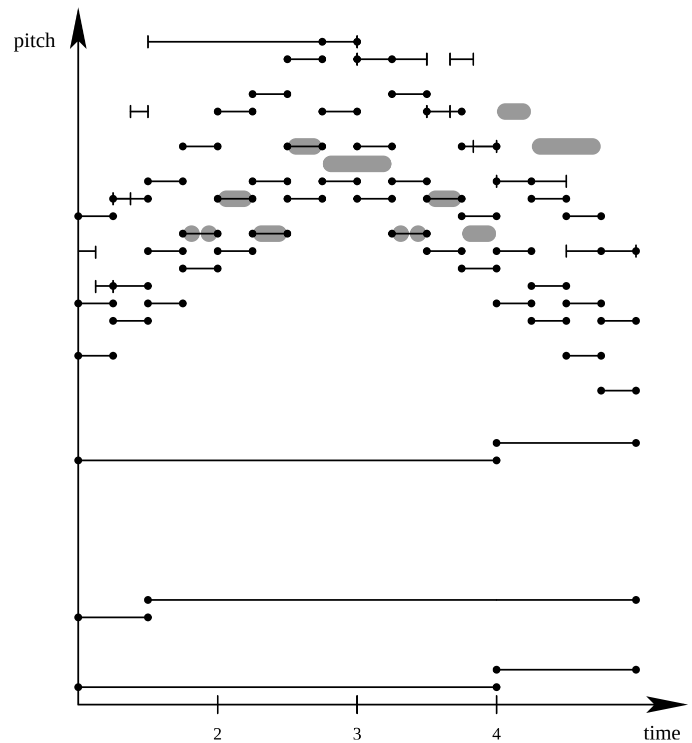
(4) **else** $\{output(f, c);\ f \leftarrow f';\ c \leftarrow 1\}$

(5) **until** $f = \infty$

Solving ($P3$)
("Longest Common Time Matching")

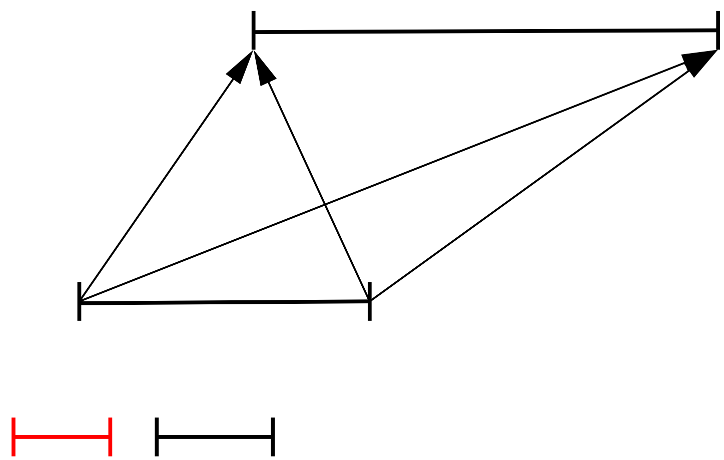
- Task: find translation f such that line segments of $P + f$ intersects T as much as possible.





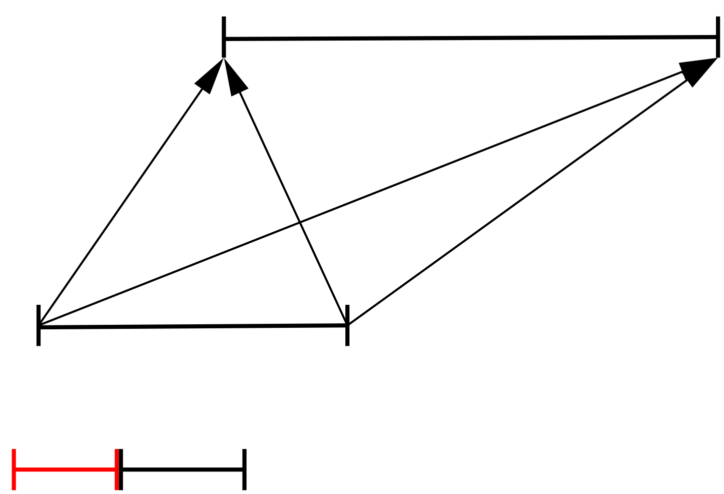
We need:

- lexicographic order of end points;
- Translation vector f separated into a pair (f_x, f_y);
- priority queue F;
- array F_y.
- $4m$ pointers q_i to define *turning points* in a feature space:



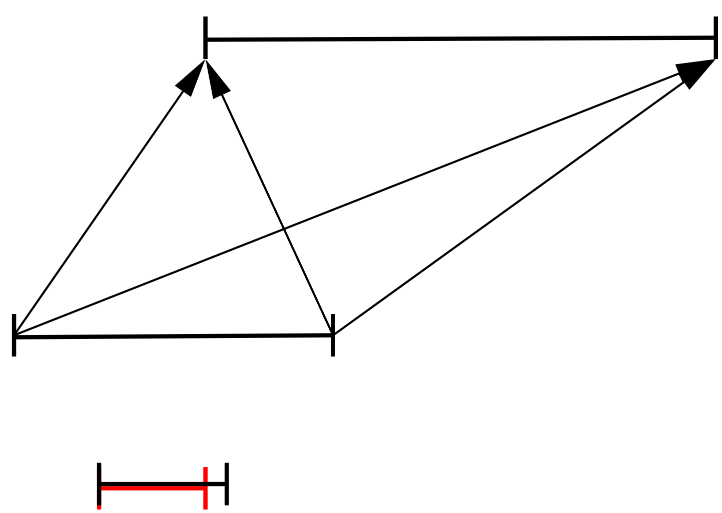
We need:

- lexicographic order of end points;
- Translation vector f separated into a pair (f_x, f_y);
- priority queue F;
- array F_y.
- $4m$ pointers q_i to define *turning points* in a feature space:



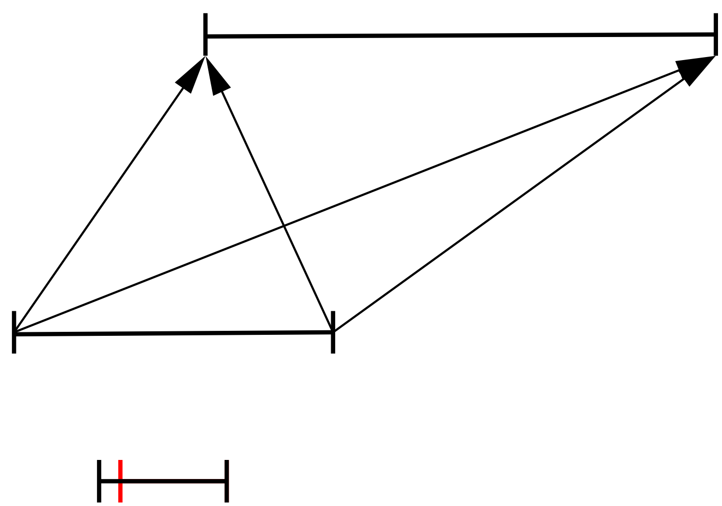
We need:

- lexicographic order of end points;
- Translation vector f separated into a pair (f_x, f_y);
- priority queue F;
- array F_y.
- $4m$ pointers q_i to define *turning points* in a feature space:



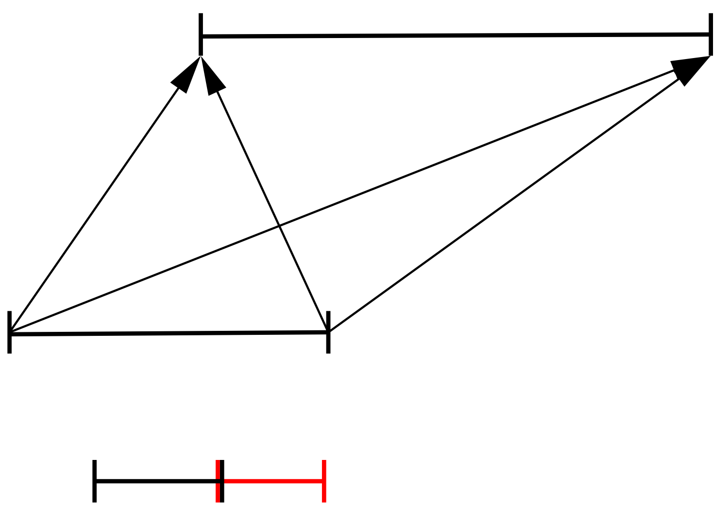
We need:

- lexicographic order of end points;
- Translation vector f separated into a pair (f_x, f_y);
- priority queue F;
- array F_y.
- $4m$ pointers q_i to define *turning points* in a feature space:



We need:

- lexicographic order of end points;
- Translation vector f separated into a pair (f_x, f_y);
- priority queue F;
- array F_y.
- $4m$ pointers q_i to define *turning points* in a feature space:



- The turning points are scanned through with the aid of the priority queue,
- Array F_y is needed for tracking values associated to distinct vertical translations.

Conclusions

The three presented algorithms solved the considered problems

($P1$) total matching;

($P2$) partial matching; and

($P3$) longest common time matching

in time and space:

Problem	Time	Space
(P1)	$O(mn)$	$O(m)$
(P2)	$O(mn \log m)$	$O(m)$
(P3)	$O(mn \log m)$	$O(m+n)$