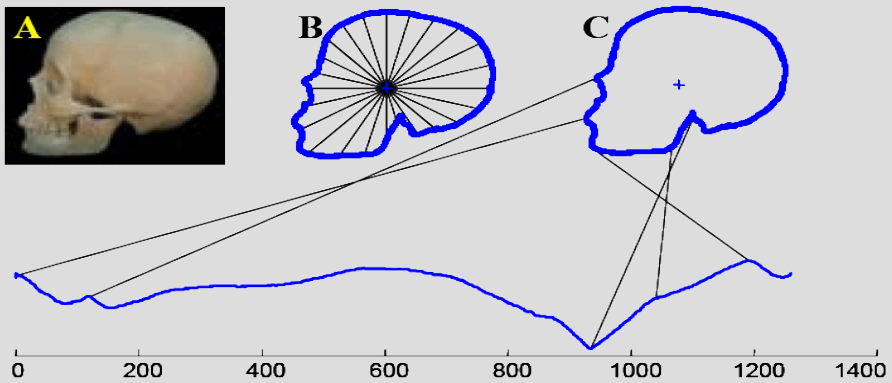
Using Contrast Sets in Time Series Data

Paper by Jessica Lin and Eamonn Keogh

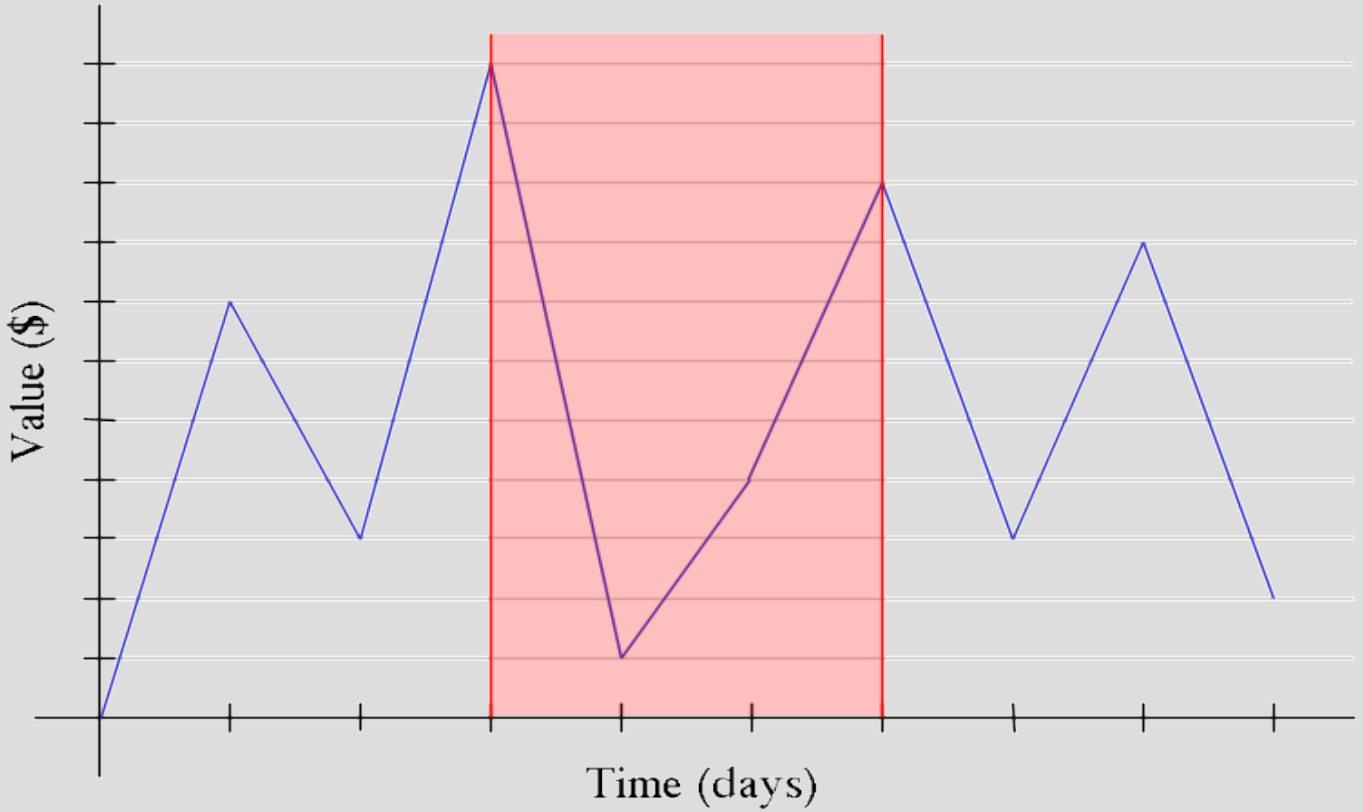


Presentation by Dave Chodos
CMPUT 695

Motivation

- Contrast Sets
  - Understand differences between groups
  - Identify attributes that differ significantly
- Time Series Data
  - Heart monitor, stock market
  - Usual techniques don't work; use key patterns
  - Can convert multimedia data into time series
- Can find key differences in images, video

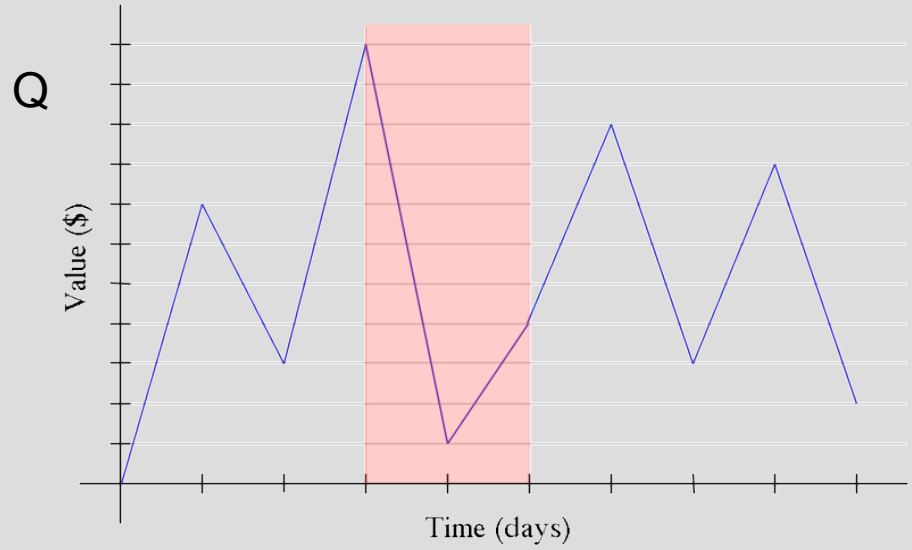
Time Series



T={7,3,11,1,4,9,3,8,2}, m=9

$C_{3,6}$={11,1,4,9}, n=4

Comparing Time Series

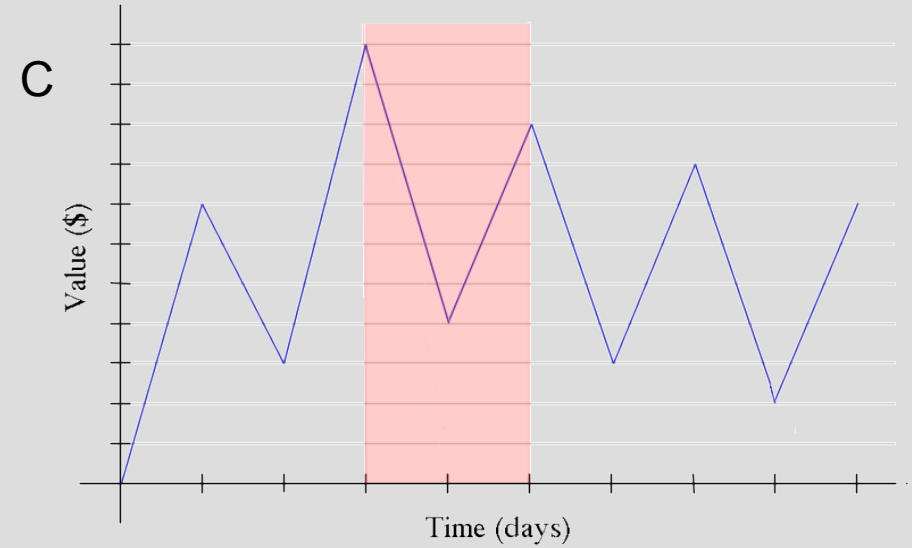


$$Dist(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

Dist(Q,C) = 12.04

TS-Diff(T, S, n) = most different subset from T of size n

TS-Diff(Q,C,3) = {11,1,4}



C=TS-Diff(T,S,n)
D=TS-Diff(S,T,n)
CS={C,D}

CS={{11,1,4},{11,4,9}}, *n*=3

Finding Contrast Sets

- Given sets S and T of size *m*, want to find contrast set of size *n*
- Brute Force Approach
  - For each subset *t* of size *n* in T, compare it with each subset *s* in S to find the closest match
  - This will take $O(m^2)$ time, which is unacceptable for large databases

Brute Force

```
For each t in T
    For each s in S
        If dist(t, s) < nearest_neighbour_dist then
            nearest_neighbour_dist = dist(t,s)
        End if
    End for
    If nearest_neighbour_dist > best_set_dist
        best_set_dist = nearest_neighbour_dist
        best_set = t
    End if
End for
Return best set, best_set_dist
```

Heuristic: TS-Diff Discovery

- Some subsets can be ruled out as candidates for the contrast set
- Can stop checking *t* if its nearest neighbour in S is closer than the current contrast set distance
- If current contrast set distance is 5, then *t* can be ruled out if dist*(t,s)* = 3
- Want to rule out as many subsets as possible, and do so quickly

Ordering T, items

- Ideally, the furthest item in T is checked first
  - All other subsets *t* may be ruled out
- Ideally, for each subset *t*, the closest item in S is checked first, so that *t* is ruled out quickly
  - Only one item in S is checked
- Use ordering heuristics *Outer* and *Inner* to try and achieve this ideal ordering

TS-Diff Discovery

```
For each t in T ordered by Outer
    For each s in S ordered by Inner
        If dist(t, s) < best_set_dist then
            Break out of loop
        Else if dist(t, s) < nearest_neighbour_dist then
            nearest_neighbour_dist = dist(t,s)
        End if
    End for
    If nearest_neighbour_dist > best_set_dist
        best_set_dist = nearest_neighbour_dist
        best_set = t
    End if
End for
Return best set, best_set_dist
```

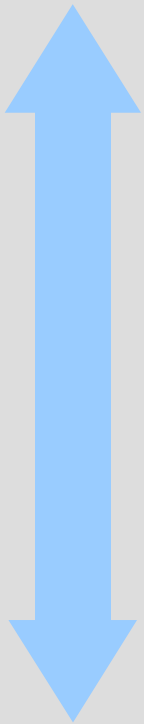
Magic heuristic

- Ideally, Outer and Inner heuristics will order T and S so that:
  - Item in T which is furthest from any item in S is placed first in T
  - For each item *t* in T, the closest element to *t* in S is placed first in S
- This results in O(*m*) runtime, as each item in T is only checked against one item in S

Perverse Heuristic

- In worst case, Outer and Inner heuristics will order items in T and S so that:
  - Items in T are ordered in ascending order w.r.t. their nearest neighbour in S
  - For each item *t* in T, the item in S which is closest to *t* is placed last in S
- This ordering results in every item in T being checked against every item in S
- Thus, we have $O(m^2)$ runtime = Brute Force

Heuristic Summary



$O(m)$ Magic – Best ordering

Approximation of best ordering

Randomized ordering

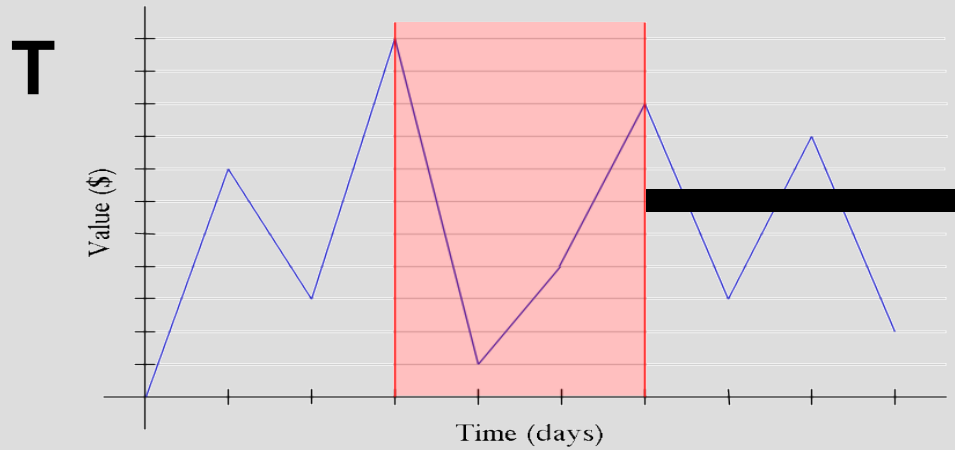
$O(m^2)$ Perverse – Worst ordering

Group SAX

- Coming close to ideal ordering will achieve dramatic speedup
- Approximation of ideal ordering requires discretization of time series
  - Use **S**ymbolic **A**ggregate Appro**X**imation (SAX)
- SAX approximates a time series of length *m* with *w* coefficients
- Coefficients are converted to one of α symbols
- Thus, have a string of characters of length *w*

Approximation of Outer

- Want to find subsets of T which are not in S
- Turn all subsets of length *n* from S, T into words
- Put words into hashtables *Hash_S*, *Hash_T*
- Scan *Hash_S* for empty buckets *b*
  - If *b* is empty in *Hash_S* but not *Hash_T*, then we have found a subset of T that is not in S
- These subsets are checked first by outer loop
  - Likely to have large distance value
  - Will result in many subsets being ruled out
- All other subsets are checked in random order



Word: jacf

aaaa	---
aaab	1
⋮	
jacf	**3**
⋮	
nnnm	---
nnnn	---

Hash_T

aaaa	1
aaab	---
⋮	
jacf	---
⋮	
nnnm	2
nnnn	---

Hash_S

aaaa	---
aaab	1
⋮	
jacf	3
⋮	
nnnm	1
nnnn	---

Hash_T

aaab jacf
⋮
nnnm
⋮

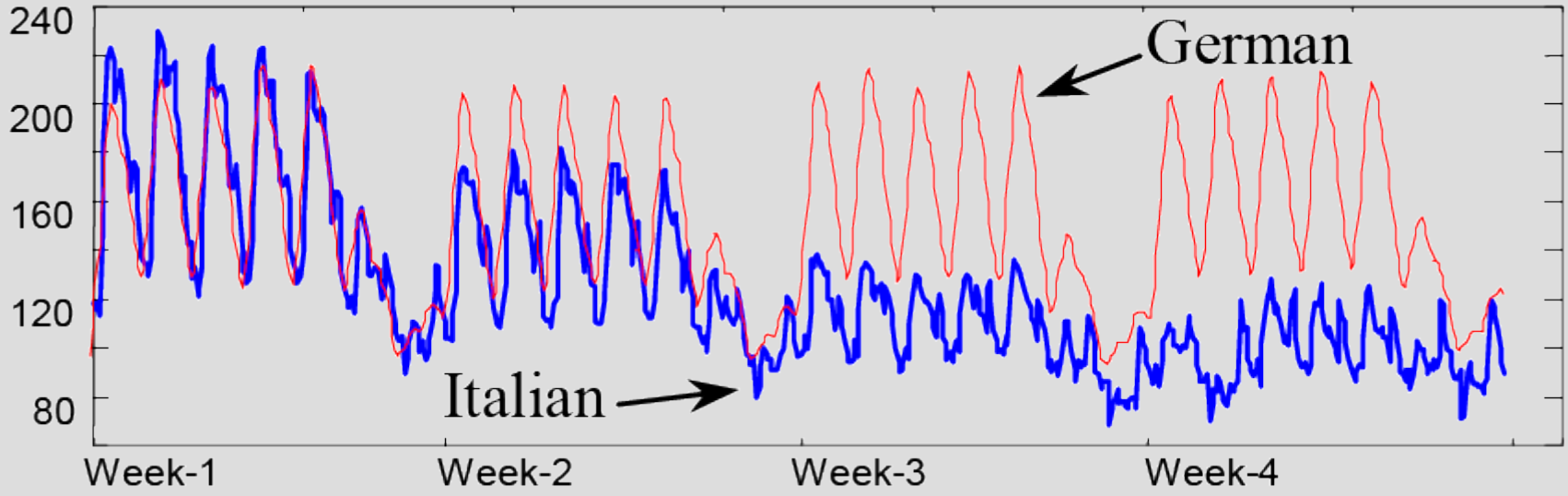
T (Sorted)

Approximation of Inner

- Want to compare *t* to a similar item in S, so that the item can be ruled out quickly
- Compute hash key for SAX word
- Check items in *Hash_S* with same hash key
- Other items in S visited in random order

Evaluation – Power Usage

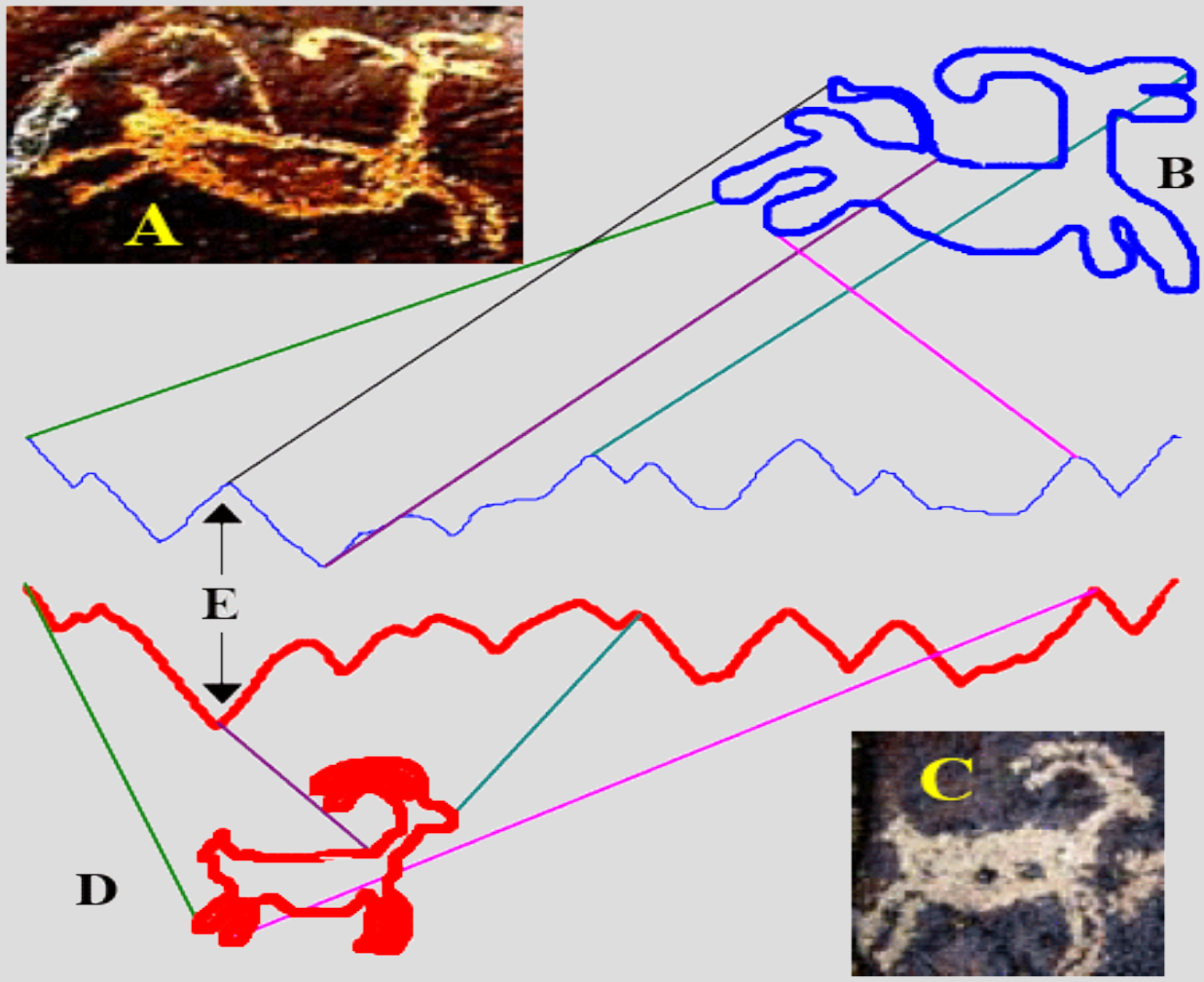
- Analyzed German vs. Italian daily power use
- Time window was 4 weeks



- Due to August lull in hot Italian summer

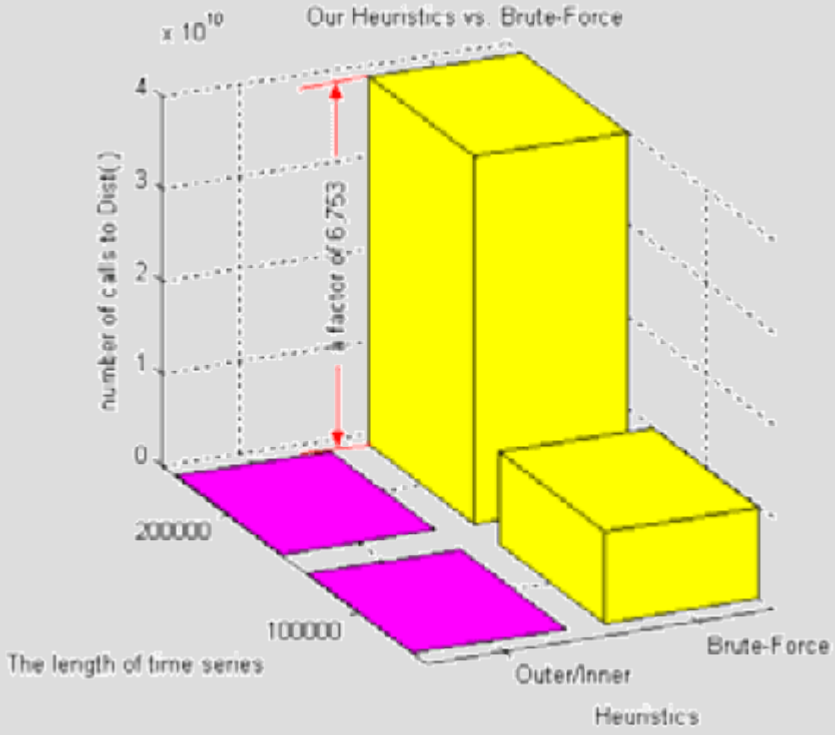
Evaluation - Petroglyphs

- Analyzed two sets of rock paintings
- Converted images into time series
- Considered orientation, rotation
  - added mirror, circular shifts to database
- Were able to identify key difference among 100,000 images at two sites


A
B
E
C
D

Performance

- Compared algorithm with brute force
- Used random walk data sets of lengths 100,000 and 200,000
- Measured number of times distance function was called
- Algorithm almost 7,000 times faster than BF



Future Work

- Authors suggest extending algorithm to:
  - multidimensional time series
  - streaming data
  - other distance measures
- Combine TS-Diff(T,S,n) and TS-Diff(S,T,n)
  - Reduce repeated calculations

Questions?