Fast Inference with Min-Sum Matrix Product

Pedro Felzenszwalb
University of Chicago
(Brown University)

Julian McAuley
Australia National University / NICTA

Overview

Exact Inference with Graphical Models

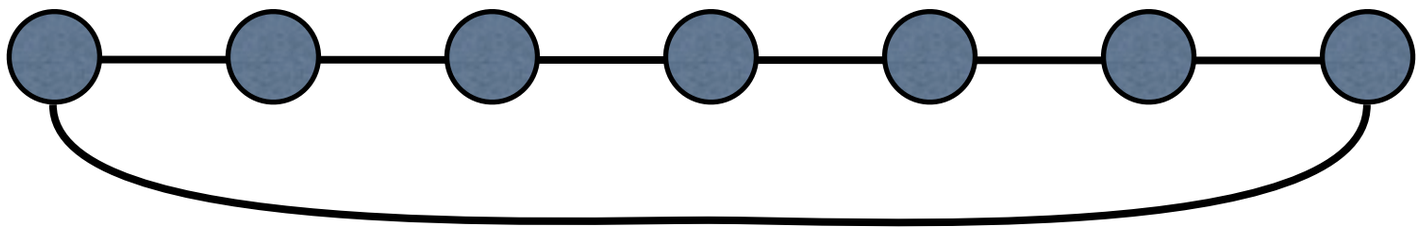
- Classical methods:
  - Dynamic programming, junction-tree, BP, etc.
  - Complexity depends (exponentially) on tree-width
- Classical methods ignore form of clique potentials
- We can do better for certain (general) classes of models
  - [McAuley, Caetano], [Felzenszwalb, McAuley]
  - Based on fast min-sum matrix multiplication

Inference on a chain

$$E(x_1, \ldots, x_m) = \sum_{i=1}^{m-1} V_i(x_i, x_{i+1})$$

- m variables
- n possible values (states) for each variable
- Goal: find minimum energy configurations
- $O(mn^2)$ time algorithm via dynamic programming
  - Best possible for arbitrary pairwise costs

Inference on a cycle



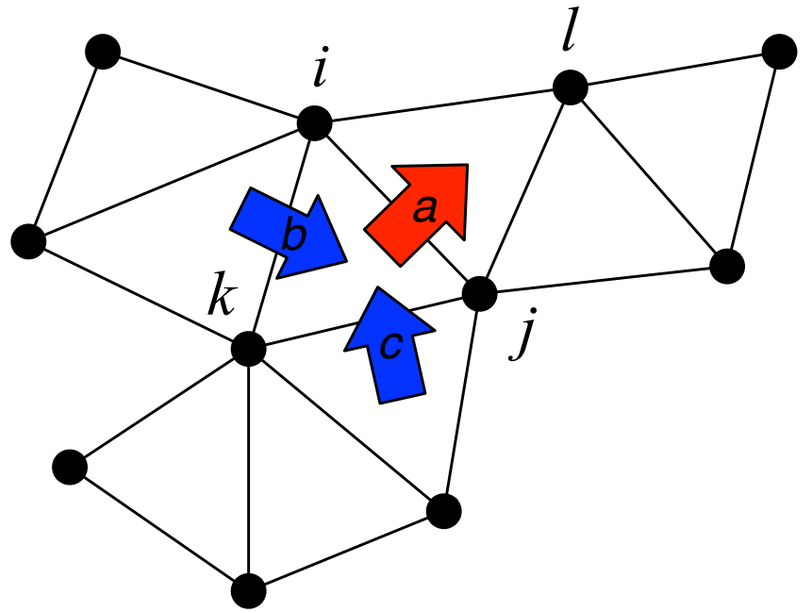
$$E(x_1, \ldots, x_m) = \sum_{i=1}^{m} V_i(x_i, x_{i+1})$$

- Add one edge to form loop
  - One more pairwise cost
- Classical methods take $O(mn^3)$ time instead of $O(mn^2)$
- Is this the best possible? No obvious reason why!
  - [McAuley, Caetano]: ~ $O(mn^{2.5})$
  - In this talk: ~ $O(mn^2 \log(n))$

Why loop is harder than chain

- Extra edge increases tree-width
  - "Measure of connectivity"
  - Chain: tree-width = 1
  - Loop: tree-width = 2
- Complexity of inference depends on tree-width
  - $O(mn^{k+1})$ time where k is tree-width
- Where is the room for improvement?
  - Loop has tree-width 2 but only pairwise costs

Inference with tree-width 2 model



- Triangulated model
  - Maximal cliques have size 3
  - One clique potential for each triangle
- Compute messages between neighboring triangles

$$m_a(x_i, x_j) = \min_{x_k}(V_{ijk}(x_i, x_j, x_k) + m_b(x_i, x_k) + m_c(x_k, x_j))$$

  - $O(n^3)$ to compute each message
  - O(mn^3) time for inference (best possible)

Pairwise costs

$$m_a(x_i, x_j) = \min_{x_k}(V_{ijk}(x_i, x_j, x_k) + m_b(x_i, x_k) + m_c(x_k, x_j))$$

- If we only have pairwise costs

$$V_{ijk}(x_i, x_j, x_k) = V_{ij}(x_i, x_j) + V_{ik}(x_i, x_k) + V_{kj}(x_k, x_j)$$

- Then

$$m_a(x_i, x_j) = V_{ij}(x_i, x_j) + \min_{x_k}(V'_{ik}(x_i, x_k) + V'_{kj}(x_k, x_j))$$

Min-Sum Product (MSP) of matrices

- $C = A * B$ (n by n matrices)
  - $C_{ik} = \min_j (A_{ij} + B_{jk})$

$O(n^3)$ brute force algorithm

No known algorithm with $O(n^{3-e})$ runtime in the worst case

- Strassen's algorithm doesn't work

Our result: $O(n^2 \log n)$ expected time, assuming values in A and B are independent samples from a uniform distribution

With tweaks this really works in practice

MSP (min-sum product) / APSP (all-pairs-shortest-paths)

- MSP reduces to APSP and vice versa
  - MSP of n by n matrices
  - APSP on dense graph with n nodes
  - If one can be solved in $O(f(n))$ time so can the other

- Solving APSP in $O(n^{3-e})$ is major open problem in TCS
  - Best known $O(n^3/\log(n))$

Basic algorithm

MSP(A, B)

1: $S := \varnothing$
2: $C_{ik} := \infty$
3: Initialize Q with entries of A, B, C
4: **while** S does not contain all C_{ik} **do**
5: $\quad item :=$ remove-min(Q)
6: $\quad S := S \cup item$
7: $\quad$ **if** $item = A_{ij}$ **then**
8: $\quad\quad$ **for** $B_{jk} \in S$ relax$(C_{ik}, A_{ij} + B_{jk})$
9: $\quad$ **end if**
10: $\quad$ **if** $item = B_{jk}$ **then**
11: $\quad\quad$ **for** $A_{ij} \in S$ relax$(C_{ik}, A_{ij} + B_{jk})$
12: $\quad$ **end if**
13: **end while**

relax(C_{ik}, v)

1: **if** $v < C_{ik}$ **then**
2: $\quad C_{ik} := v$
3: $\quad$ decrease-key(Q, C_{ik})
4: **end if**

Correctness

Assume entries in A and B are non-negative

Let $j = \text{argmin}\, A_{ij} + B_{jk}$

We always have $C_{ik} \geq A_{ij} + B_{jk}$

So A_{ij} and B_{jk} come off the queue before C_{ik}

This implies we call relax$(C_{ik}, A_{ij} + B_{jk})$

When C_{ik} comes off the queue it equals $A_{ij} + B_{jk}$

Implementation

MSP(A, B)

1: $S := \varnothing$
2: $C_{ik} := \infty$
3: Initialize Q with entries of A, B, C
4: **while** S does not contain all C_{ik} **do**
5: $\quad item :=$ remove-min(Q)
6: $\quad S := S \cup item$
7: $\quad$ **if** $item = A_{ij}$ **then**
8: $\quad\quad$ **for** $B_{jk} \in S$ relax$(C_{ik}, A_{ij} + B_{jk})$
9: $\quad$ **end if**
10: $\quad$ **if** $item = B_{jk}$ **then**
11: $\quad\quad$ **for** $A_{ij} \in S$ relax$(C_{ik}, A_{ij} + B_{jk})$
12: $\quad$ **end if**
13: **end while**

relax(C_{ik}, v)

1: **if** $v < C_{ik}$ **then**
2: $\quad C_{ik} := v$
3: $\quad$ decrease-key(Q, C_{ik})
4: **end if**

Maintain $2n$ lists

$I[j]$: list of i such that A_{ij} in S

$K[j]$: list of k such that B_{jk} in S

Running time determined by number of additions and priority queue operations

Runtime Analysis

- Let N = # pairs (A_{ij}, B_{jk}) that are combined before we stop
  - Both A_{ij} and B_{jk} come off the queue
- Main Lemma: $E[N] = O(n^2 \log n)$
- Running time:
  - N additions
  - $3n^2$ insertions
  - at most $3n^2$ remove-min
  - at most N decrease-key
- Using a Fibonacci heap the expected time is $O(n^2 \log n)$

Main lemma

Let N = # pairs (A_{ij}, B_{jk}) that are combined

If entries in A and B are iid samples from a uniform distribution over [0,1] then $E[N] = O(n^2 \log n)$

- Basic idea:
  - Let M be maximum value in C
  - A_{ij}, B_{jk} come off queue if both are at most M
  - Probability that M is large is low
  - Probability that both A_{ij}, B_{jk} are small is low

Improvements - normalizing the inputs

1) Subtract min value from each row of A and column of B
   (add back to C in the end)

2) Remove entries from I/K if we finish a row/column of C

3) (A* search)

Let $a(j)$ be minimum value in column j of A

Let $b(j)$ be minimum value in row j of B

- Put A_{ij} into Q at priority $A_{ij} + b(j)$
- Put B_{jk} into Q at priority $B_{jk} + a(j)$

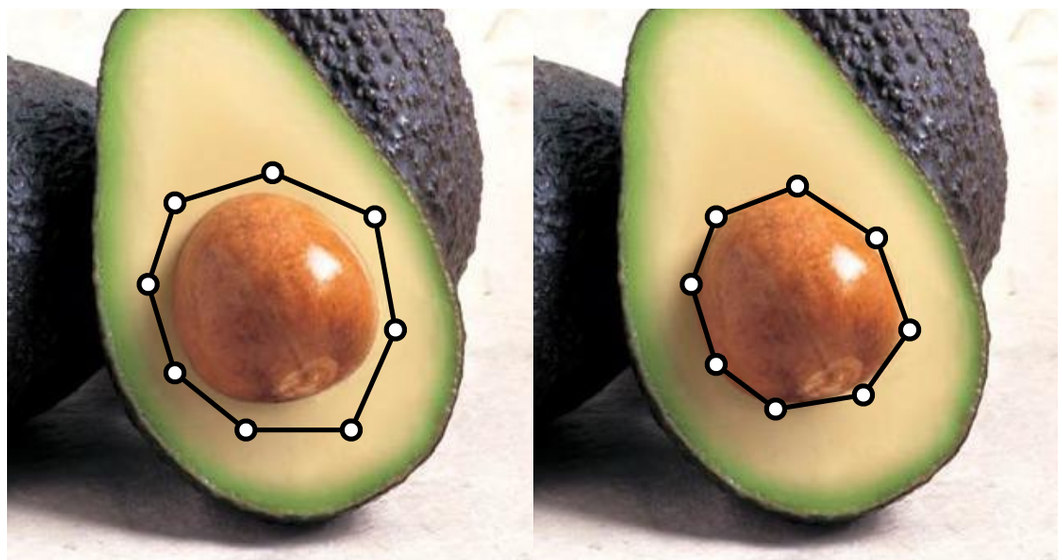
Practical issues

Fibonacci heap not practical (believe me, we tried)

Alternatives:

- Integer queue
  - In principle could introduce rounding errors but can be made exact without increasing running time
- Scaling method
  - No data structures, very simple to implement

Application: snakes



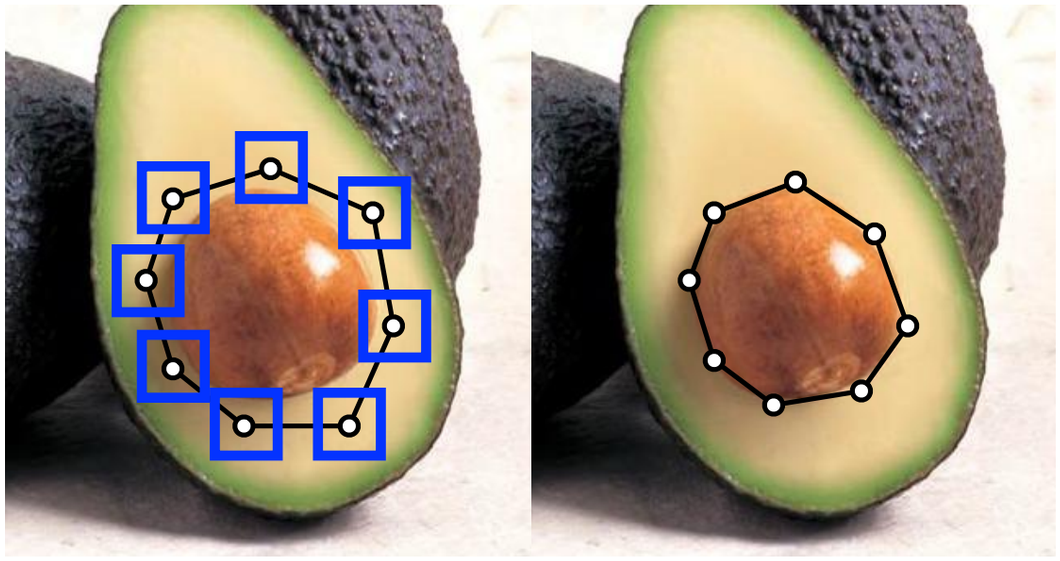
Goal: trace the boundary of an object

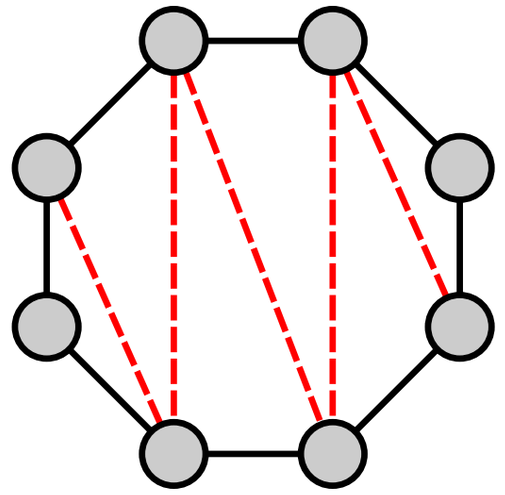
User initializes a contour close to an object boundary

Contour moves to the boundary

- Attracted to local features (intensity gradient)
- Internal forces enforce smoothness

Optimization problem





triangulated model

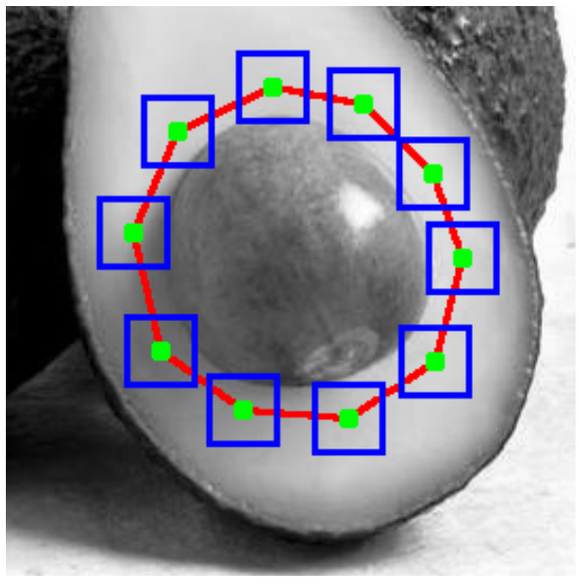
m control points

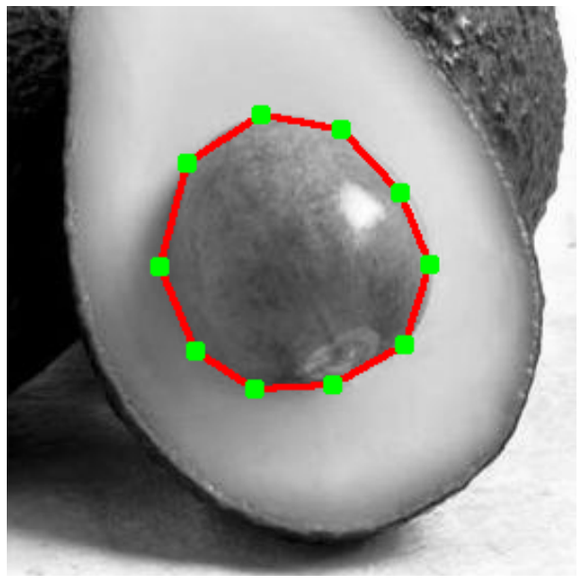
n possible locations for each point (blue regions)

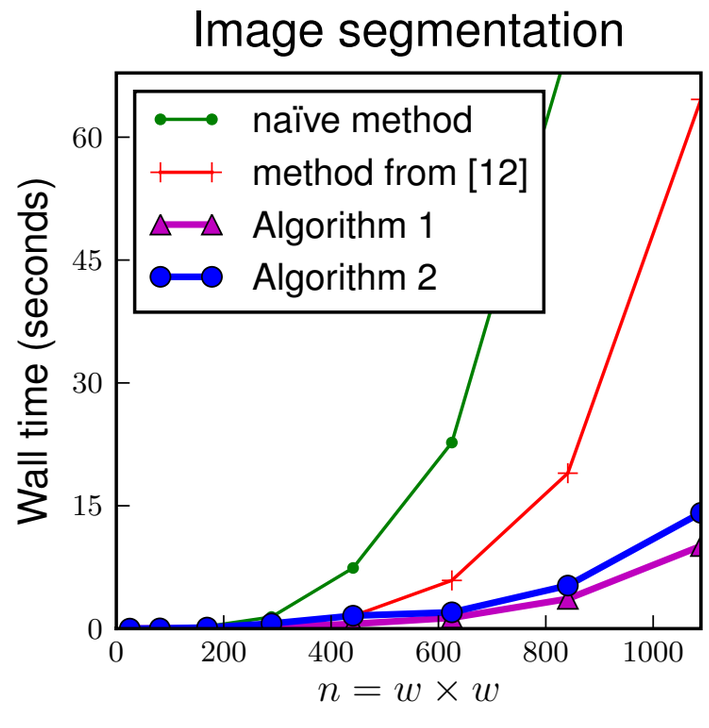
minimize:

$$E(x_1, \ldots, x_m) = \sum_{i=1}^{m} V_i(x_i, x_{i+1})$$

Experimental results with real data







naive method uses $O(n^3)$ brute-force algorithm MSP

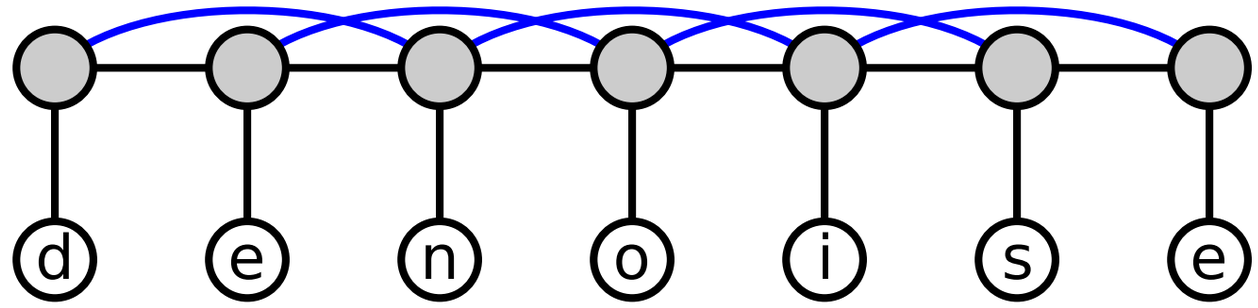
[12] gives an $O(n^{2.5})$ algorithm with (weaker) assumption that entries come in random order

Algorithm 1: integer queue

Algorithm 2: scaling method

Application: Language modeling

Something between
bigram and trigram model



- Bigram: $P(x_t \mid x_{t-1})$
- Trigram: $P(x_t \mid x_{t-1}, x_{t-2})$
- Skip-chain: $P(x_t \mid x_{t-1}, x_{t-2}) \sim q_1(x_t, x_{t-1})\ q_2(x_t, x_{t-2})$

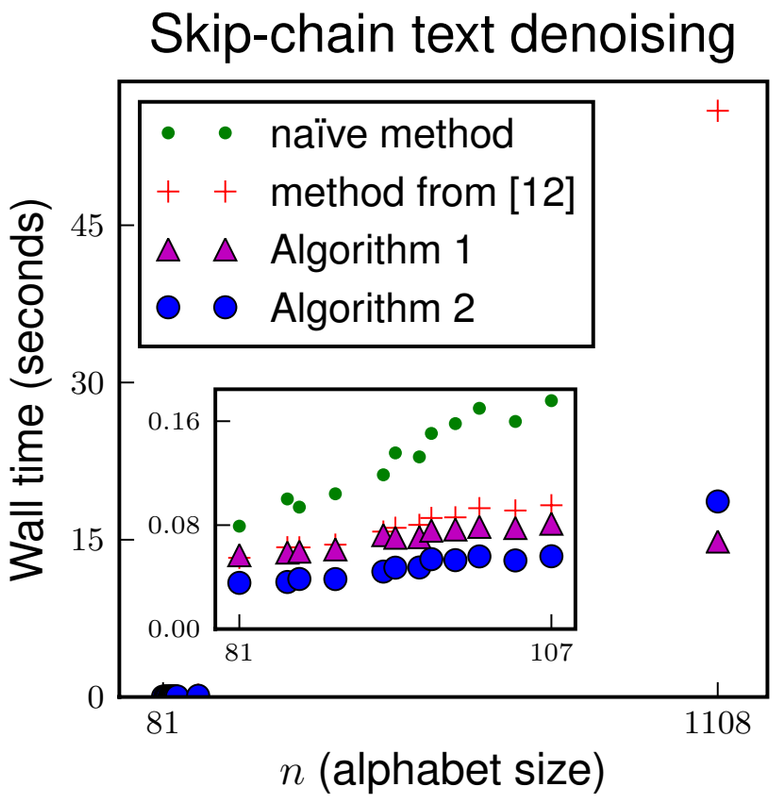
Task: recover a sentence from noisy data

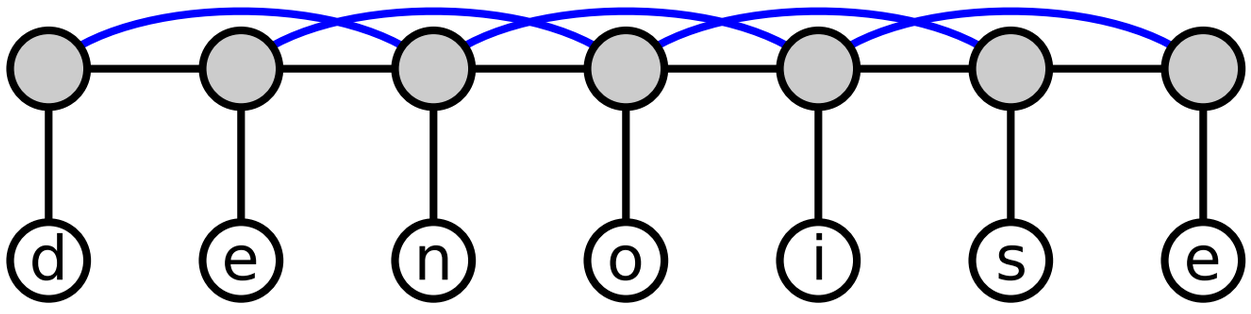
Each character corrupted with probability e

Use skip model as prior over sentences $P(x)$

Given corrupted text y, find x maximizing $P(x|y) \sim P(y|x)P(x)$

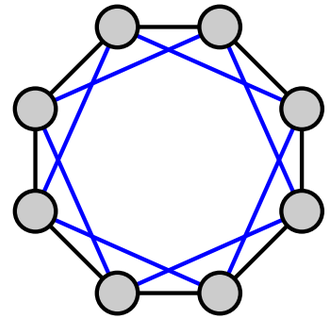
Language modeling





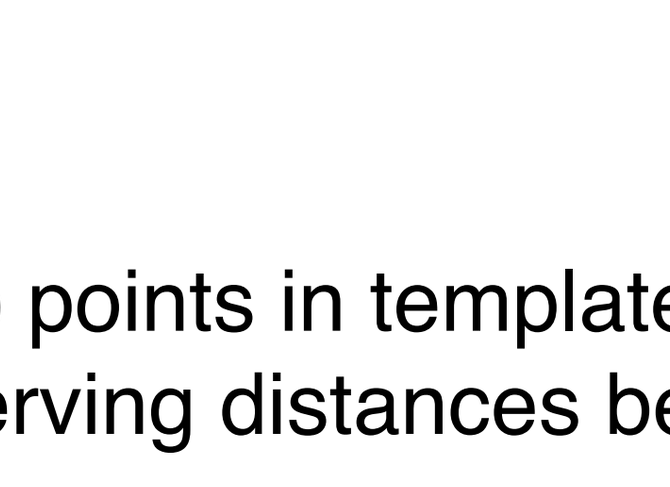
Application: Point pattern matching

Map points in template to points in target preserving distances between certain pairs



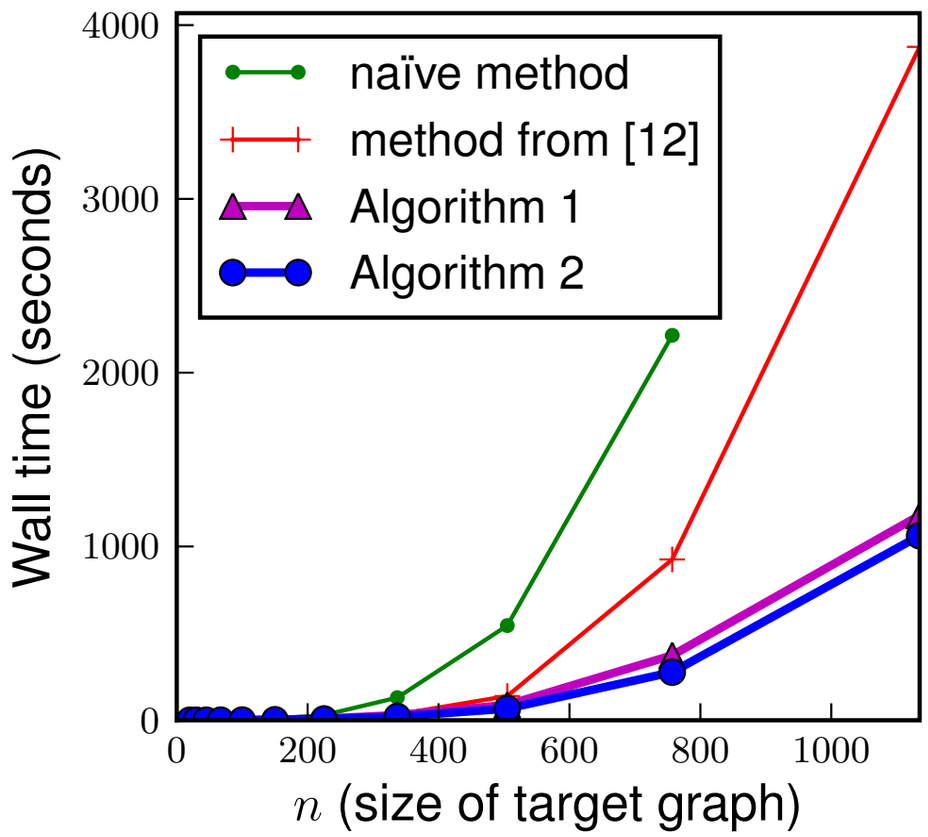
(c) Point-matching model

template



target

2D Graph matching



Application: Parsing

Parsing with stochastic context-free grammars

- $O(n^3)$ with dynamic programming (CKY)
- Reduces to MSP with Valiant's transitive closure method

RNA Secondary structure prediction

- $O(n^3)$ dynamic programming
- Reduces to parsing with special grammar

Some open questions

- Why does it work on non-random inputs?
- Characterize what "normalization" is doing
  - How does it relax assumptions on input distribution?

- Can we get an $O(n^{3-e})$ worst case algorithm for MSP? (randomized)

- Can we get a practical parsing method?
  - Avoid transitive closure machinery