Fast Inference with Min-Sum Matrix Product

(or how I finished a homework assignment 15 years later)

Pedro Felzenszwalb
University of Chicago

Julian McAuley
Australia National University / NICTA
Active contour models (snakes) for interactive segmentation

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

$m$ control points

$n$ possible locations for each point (blue regions)

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

\[ x_i = \text{location of } i\text{-th control point} \]

Many reasonable choices for $V$

\[ V(p, q) = \frac{1}{\text{grad}(I, p, q)} + \|p - q\|^2 \]
Dynamic programming for open snakes

Shortest path problem

$m$ tables with $n$ entries each

$T_i[p] = \text{cost of best placement for first } i \text{ points with } x_i = p$

- $T_i[p] = \min_q T_{i-1}[q] + V_i(q, p)$
- Pick best location in $T_m$, trace-back

$O(mn^2)$ time (optimal in a reasonable sense)
CS664 Homework assignment: 
Implement closed snakes

pff’s solution:

• Consider one control point \( x_i \)
• Fixing its location leads to open snake problem
• Try all \( n \) possibilities for \( x_i \): \( O(mn^3) \) time total

Is this a good solution?

• pff: I think this is the best possible
• rdz: Are you sure?
An alternative solution

Single DP problem

$m$ tables with $n^2$ entries

$T_i[p, q] = \text{cost of best placement for first } i \text{ points with } x_1 = p, x_i = q$

- $T_i[p, q] = \min_r T_{i-1}[p, r] + V_i(r, q)$
- compute $T_i$ from $T_{i-1}$ in $O(n^3)$ time
- Optimal position for $x_1$ minimizes $T_n[p, p]$
- still $O(mn^3)$ time total...

But, we can write: $T_i = T_{i-1} \ast V_i$

Min-sum matrix product (MSP), a.k.a. distance product
MSP (min-sum product) / APSP (all-pairs-shortest-paths)

\[ C = A \times B \quad C_{ik} = \min_j A_{ij} + B_{jk} \]

MSP reduces to APSP and vice versa

SP distance matrix in graph with \( n \) nodes

\[ E \times E \times E \times E \ldots = E^n \] (transitive closure of \( n \) by \( n \) adjacency matrix)

\[ C_{ik} = d((1, i), (3, k)) \]
MSP algorithms

$O(n^3)$ brute force algorithm, $O(n^3 / \log n)$ via APSP

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 are independent samples from a uniform distribution

With tweaks this really works in practice

- On inputs with significant structure from real applications in vision and natural language
Basic algorithm

MSP(A, B)

1: \( S := \emptyset \)

2: \( C_{ik} := \infty \)

3: Initialize \( Q \) with entries of \( A, B, C \)

4: while \( S \) does not contain all \( C_{ik} \) do

5: \( \text{item} := \text{remove-min}(Q) \)

6: \( S := S \cup \text{item} \)

7: if \( \text{item} = A_{ij} \) then

8: \( \text{for } B_{jk} \in S \text{ relax}(C_{ik}, A_{ij} + B_{jk}) \)

9: end if

10: if \( \text{item} = B_{jk} \) then

11: \( \text{for } A_{ij} \in S \text{ relax}(C_{ik}, A_{ij} + B_{jk}) \)

12: end if

13: end while

relax(\( C_{ik}, v \))

1: if \( v < C_{ik} \) then

2: \( C_{ik} := v \)

3: decrease-key(\( Q, C_{ik} \))

4: end if
Correctness

Assume entries in $A$ and $B$ are non-negative
Let $j = \operatorname{argmin} A_{ij} + B_{jk}$
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 $\text{relax}(C_{ik}, A_{ij} + B_{jk})$
When $C_{ik}$ comes off the queue it equals $A_{ij} + B_{jk}$
Implementation

MSP($A$, $B$)

1: $S := \emptyset$
2: $C_{ik} := \infty$
3: Initialize $Q$ with entries of $A$, $B$, $C$
4: while $S$ does not contain all $C_{ik}$ do
5: 
6: 
7: if item = $A_{ij}$ then
8: 
9: end if
10: if item = $B_{jk}$ then
11: 
12: end if
13: end while

relax($C_{ik}$, $v$)

1: if $v < C_{ik}$ then
2: $C_{ik} := v$
3: 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 $N = \# \text{ pairs } A_{ij}, B_{jk} \text{ that are combined before we stop}$

(both $A_{ij}, B_{jk}$ come off the queue)

- $N$ additions
- $3n^2$ insertions
- at most $3n^2$ remove-min
- at most $N$ decrease-key

Lemma: $E[N] = O(n^2 \log n)$

Using a Fibonacci heap the expected time is $O(n^2 \log n)$
Main lemma

Let \( N = \# \) pairs \( A_{ij}, B_{jk} \) that come off the queue

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

proof sketch:

Let \( X_{ijk} = 1 \) if \( A_{ij} \) and \( B_{jk} \) both come off the queue

\[
E[N] = \sum_{ijk} E[X_{ijk}] = \sum_{ijk} P(X_{ijk} = 1).
\]

Minimum priority in \( Q \) is non-decreasing

Let \( M \) be maximum value in \( C \)

\( X_{ijk} = 1 \) if \( A_{ij} \) and \( B_{jk} \) are at most \( M \)
$X_{ijk} = 1$ if $A_{ij}$ and $B_{jk}$ are at most $M$

The probability that $M$ is large is low

$M \geq \epsilon$ iff one $C_{ik} \geq \epsilon$

$C_{ik} \geq \epsilon$ iff all $A_{ij} + B_{jk} \geq \epsilon$

$P(A_{ij} + B_{jk} \geq \epsilon) = 1 - \epsilon^2/2 \leq e^{-\epsilon^2/2}$

$P(M \geq \epsilon) \leq n^2 e^{-n\epsilon^2/2}$ (union + independence)

The probability that $A_{ij}$ and $B_{jk}$ are both small is low

$P(A_{ij} \leq \epsilon \land B_{jk} \leq \epsilon) = \epsilon^2.$

$P(X_{ijk} = 1) \leq n^2 e^{-n\epsilon^2/2} + \epsilon^2.$

Pick $\epsilon = \frac{6 \log n}{n}$

$P(X_{ijk} = 1) \leq \frac{1+6 \log n}{n}$

$E[N] \leq n^2(1 + 6 \log n)$
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)

Practical alternatives:

• Integer queue gives approximation algorithm
• Avoid queue by sorting A and B
  – ok, but not as fast as integer queue
• Scaling method
  – Avoids sorting
  – exact, and fastest in practice
Scaling method

1: $C_{ik} := \infty$

2: $T := t-min$

3: while $\max_{ik} C_{ik} > T$ do

4: $I[j] := \{i \mid A_{ij} \leq T\}$

5: $K[j] := \{k \mid B_{jk} \leq T\}$

6: for $j \in \{1 \ldots n\}$ do

7: for $i \in I[j]$ do

8: for $k \in K[j]$ do

9: $C_{ik} = \min(C_{ik}, A_{ij} + B_{jk})$

10: end for

11: end for

12: end for

13: $T := 2T$

14: end while

Consider entries of $A$ and $B$ that are at most $T$

If maximum entry in resulting $C$ is at most $T$ we are done
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 (approximate)
Algorithm 2: scaling method (exact)
Other Applications

MAP estimation with pairwise graphical model

- $m$ variables, $n$ possible values for each variable

$$E(x_1, \ldots, x_m) = \sum_{i=1}^{m} V_i(x_i) + \sum_{(i,j) \in E} V_{ij}(x_i, x_j)$$

Tree-width 2 model

- $m$ MSP of $n$ by $n$ matrices
- $O(mn^3) \rightarrow O(mn^2 \log n)$
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**: \( P(x_t \mid x_{t-1}, x_{t-2}) \sim q_1(x_t, x_{t-1}) \cdot q_2(x_t, x_{t-2}) \)

Task: recover a sentence from noisy data

Assume each character is corrupted with probability \( c \)

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

Given corrupted text \( y \), find \( x \) maximizing \( P(x \mid y) \sim P(y \mid x)P(x) \)
Language modeling

naive method takes $O(mn^3)$
$m$ is the length of the sentence
$n$ is the alphabet size
Point pattern matching

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

2D Graph matching

Wall time (seconds)

0 1000 2000 3000 4000

n (size of target graph)

Algorithm 1
Algorithm 2
naïve method
method from [12]

(c) Point-matching model

template

target
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 actually work?
Characterize what “normalization” is doing
How does it relax assumptions on input distribution

$O(n^{3-e})$ worst case (randomized) algorithm for MSP

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