Paper Reading Seminar

Yan Wang

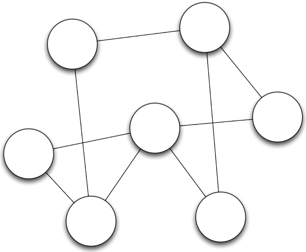
October 3, 2012

Distributed Computer Vision Algorithms Through Distributed Averaging

- Intuition: make linear algebra algorithms distributed $+$ CV applications
- Outline
  - Basic distributed algorithms
  - Linear algebra algorithms
  - Applications in CV

Basic distributed algorithm

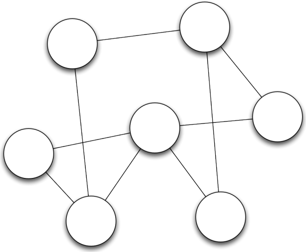
- Basic assumption
  - Several computation nodes with limited computation power as well as battery
  - Not fully connected: info can only propagated among neighbors



Basic distributed algorithm

- Averaging a number x
  - Iterative algorithm: averaging among neighbors

$$x(t+1) = x(t) + \epsilon \sum_{j \in N_i} \Big(x_j(t) - x_i(t) \Big)$$



  - Easily extended to vectors/matrices

Basic distributed algorithm

- Get the min/max

$$x_i(t+1) = \min_{j \in \{N_i \bigcup i\}} x_j(t)$$

- Covariance
  - $A = \begin{bmatrix} A_1^T & A_2^T & \cdots & A_N^T \end{bmatrix}^T \in \mathbb{R}^{n \times m}$
  - Compute $C = \frac{1}{N} A^T A$

$$C = \frac{1}{N} \sum_i A_i^T A_i$$

  - Compute $C_i = A_i^T A_i$, and then average among all the nodes
  - Only meaning for when $\mathrm{rank}(C) \ll n$

Linear algebra algorithms

- SVD

$$A = U\Sigma V^T$$

  - Compute $C = A^T A$
  - Locally compute C's SVD: $C = V(\frac{1}{N}\Sigma^2)V^T$
  - Recover U_i locally: $U_i = A_i V \Sigma^{-1}$

Linear algebra algorithms

- PCA
  - Compute the average of $A \Rightarrow$ do data centralization
  - Compute covariance matrix $C = A^T A$
  - Local SVD decomposition: $C = U \Sigma U^T \Rightarrow$ get the basis
  - Compute the new representation: $\hat{A} = A^T U$

Linear algebra algorithms

- Least square estimation
  - $x_0 = \arg\min_x \|Ax - b\|^2$
  - $x_0 = \arg\min_x \|A^T Ax - A^T b\|^2$ (is this true?)
  - $A^T A$ and $A^T b$ easy to compute distributedly
  - Can be solved locally
  - (Not very clear)

Applications in CV

- Point triangulation
- Linear pose estimation
- Structure from Motion

Coherency Sensitive Hashing

- A simple introduction
- Find ANN for local patches
- Naive baseline: brute-force match in the feature space
- PatchMatch: like Genetic Algorithm
  - Hold a candidate pool
  - If two patches in two images are similar, include nearby patches in the pool
  - Randomly add patch pairs in the pool to get rid of (too) local minima.

Approach

- Hashing + more sophisticated candidate expansion
  - Hashing: LSH + 2D Walsh Hadamard Kernel
  - Candidate expansion
    - $g_A(a) = g_B(b) \Rightarrow$ include b in Cand(a)
    - $b \in \text{Cand}(a_1), g_A(a_1) = g_A(a_2) \Rightarrow$ include b in Cand(a_2)
    - $b_1 \in \text{Cand}(a_2), g_B(b_1) = g_B(b_2) \Rightarrow$ include b_2 in Cand(a_2)
    - Include spatial neighbor pairs in the candidates
  - Still need to do verifications on the candidate set