Efficient Quality Threshold Clustering for Parallel Architectures

Anthony Danalis, Collin McCurdy, Jeffrey S. Vetter

IPDPS 2012, Shanghai, China
What is clustering?

Cluster Diameter:

Data Points
What is clustering?
Euclidean vs. Non Euclidean Data

\[ c = f(a, b, \phi, \theta) \]

\[ 0 < c < a+b \]
Euclidean vs. Non Euclidean Data

\[ c = f(a,b,\varphi,\theta) \]

\[ 0 < c < a+b \]
Euclidean vs. Non Euclidean Data

\[ c = f(a, b, \phi, \theta) \]

\[ 0 < c < a + b \]
Euclidean vs. Non Euclidean Data

\[ c = f(a, b, \phi, \theta) \]

\[ 0 < c < a+b \]
Graph Formulation of Problem
Quality Threshold Clustering

\[ QT\_\text{Clust}(G, d) \{ \]
\[ \text{if}(|G| \leq 1) \text{ then output } G, \text{ else do } \]
\[ \text{foreach } p_i \in G \]
\[ \text{flag} = \text{TRUE} \]
\[ C_i = \{p_i\} /\!* \text{Ci is the cluster started by element pi } */\]
\[ \text{while}(\text{flag} == \text{TRUE}) \&\& (C_i \neq G) \]
\[ \text{find } p_j \in (G - C_i) : \text{diameter}(C_i \cup \{p_j\}) \text{ is minimum} \]
\[ \text{if}(\text{diameter}(C_i \cup \{p_j\}) > d) \]
\[ \text{then flag} = \text{FALSE} \]
\[ \text{else } C_i = C_i \cup \{p_j\} /\!* \text{Add p}_j \text{ to cluster } C_i */\]
\[ \text{identify set } C \in \{C_1, C_2, \cdots, C_{|G|}\} \text{ with maximum cardinality} \]
\[ \text{output } C \]
\[ \text{call } QT\_\text{Clust}(G - C, d) /\!* \text{recurse } */\]
\[ \} \]

IPDPS 2012, Anthony Danalis
# Quality Threshold Clustering

Quality Threshold Clustering is an algorithm designed to cluster elements in a graph `G` based on a quality threshold `d`. The algorithm iteratively finds elements that do not fit into existing clusters and adds them to the cluster with the minimum increase in diameter, until all elements are assigned to clusters.

The algorithm is implemented as follows:

```
foreach (element in G)

    while( (flag==TRUE) && (Ci ≠ G) )
        find `pj` ∈ (G − Ci): `diameter(Ci ∪ {pj})` is minimum
        if( `diameter(Ci ∪ {pj})` > d )
            then flag = FALSE
        else Ci = Ci ∪ {pj} /* Add pj to cluster Ci */
    identify set `C` ∈ `{C1, C2, ..., C|G|}` with maximum cardinality
    output `C`
    call QT_Clust(G − C, d) /* recurse */

```
Quality Threshold Clustering

\[ \text{QT-Clust}(G, d) \{ \]

\hspace{1cm} \text{foreach (element in G)}

\hspace{1cm} \text{while (candidate cluster can grow)}

\hspace{1.5cm} \text{\textbf{find} } p_j \in (G - C_i) : \textbf{diameter}(C_i \cup \{p_j\}) \text{ is minimum}

\hspace{1.5cm} \text{if( } \text{diameter}(C_i \cup \{p_j\}) > d \text{ )}

\hspace{1.5cm} \text{then flag = FALSE}

\hspace{1.5cm} \text{else } C_i = C_i \cup \{p_j\} \text{ /* Add } p_j \text{ to cluster } C_i */

\hspace{1cm} \text{identify set } C \in \{C_1, C_2, \cdots, C_{|G|}\} \text{ with maximum cardinality}

\hspace{1cm} \text{output } C

\hspace{1cm} \text{call QT-Clust}(G - C, d) \text{ /* recurse */}

\}

IPDPS 2012, Anthony Danalis
Quality Threshold Clustering

\[ QT_{\text{Clust}}(G, d) \{
\]

\hspace{1cm} \text{foreach (element in G)}

\hspace{1cm} \text{while (candidate cluster can grow)}

\hspace{1.5cm} \text{find (element : min diameter)}

\hspace{1.5cm} \text{call QT_{Clust}(G - C, d) /* recurse */}

\}

IPDPS 2012, Anthony Danalis
Quality Threshold Clustering

foreach (maximal cluster)

    foreach (unclustered element)

        while (cand. cluster can grow)

            find (element : min diameter)
Worst Case Asymptotic Complexity

O(1) clusters of O(N) elements

foreach (maximal cluster)

foreach (unclustered element)

while (cand. cluster can grow)

find (element : min diameter)

O(N) clusters of O(1) elements
Worst Case Asymptotic Complexity

\[ O(1) \text{ clusters of } O(N) \text{ elements} \quad O(N) \text{ clusters of } O(1) \text{ elements} \]

\[ O(1) \quad \text{foreach (maximal cluster)} \]

\[ O(N) \quad \text{foreach (unclustered element)} \]

\[ O(N) \quad \text{while (cand. cluster can grow)} \]

\[ O(N \cdot Fd) \quad \text{find (element : min diameter)} \]
Worst Case Asymptotic Complexity

\[ O(1) \text{ clusters of } O(N) \text{ elements} \]

\[ O(1) \text{ foreach (maximal cluster)} \]

\[ O(N) \text{ foreach (unclustered element)} \]

\[ O(N) \text{ while (cand. cluster can grow)} \]

\[ O(N \times Fd) \text{ find (element : min diameter)} \]

\[ O(N) \text{ clusters of } O(1) \text{ elements} \]
Worst Case Asymptotic Complexity

- **O(1) clusters of O(N) elements**
  - O(1) foreach (maximal cluster)
  - O(N) foreach (unclustered element)
  - O(N) while (cand. cluster can grow)
  - O(N*F_d) find (element : min diameter)

- **O(N) clusters of O(1) elements**
  - O(N) foreach (maximal cluster)
  - O(N) foreach (unclustered element)
  - O(1) while (cand. cluster can grow)
  - O(N*F_d) find (element : min diameter)

- **O(N^3 * F_d)**
  - O(N^3 * F_d)
Worst Case Asymptotic Complexity

O(1) clusters of O(N) elements

O(1) foreach (maximal cluster)

O(N) foreach (unclustered element)

O(N) while (cand. cluster can grow)

O(N*Fd) find (element : min diameter)

O(N^3 * Fd)

O(f(N)) clusters of O(N/f(N)) elements:
O(f(N) * N * N/f(N) * N * Fd) = O(N^3 * Fd)

O(N) clusters of O(1) elements

O(N)

O(N)

O(1)

O(N*Fd)
Worst Case Asymptotic Complexity

\[ O(1) \text{ clusters of } O(N) \text{ elements} \]
\[ O(N) \text{ clusters of } O(1) \text{ elements} \]

\[ O(1) \text{ foreach (maximal cluster)} \]
\[ O(N) \text{ foreach (unclustered element)} \]
\[ O(N) \text{ while (cand. cluster can grow)} \]
\[ O(N \times Fd) \text{ find (element : min diameter)} \]

\[ O(N^3 \times Fd) \]

\[ O(f(N)) \text{ clusters of } O(N/f(N)) \text{ elements:} \]
\[ O(f(N) \times N \times N/f(N) \times N \times Fd) = O(N^3 \times Fd) \]

O(1) clusters of O(N) elements + O(N) clusters of O(1) elements:
\[ O(N^3 \times Fd) + O(N^3 \times Fd) = O(N^3 \times Fd) \]
Diameter Function Complexity: Fd

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements
Diameter Function Complexity: $F_d$

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements

Naïve: maximum pairwise distance in resulting cluster
$O(F_d) = O(M^2) \Rightarrow O(F_d) = O(N^2) \Rightarrow O(QTC) = O(N^5)$
Diameter Function Complexity: \( F_d \)

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements

Better: maximum distance between new element and any element in cluster:
\[ O(F_d) = O(M) \Rightarrow O(F_d) = O(N) \Rightarrow O(QTC) = O(N^4) \]
Diameter Function Complexity: $F_d$

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements

Best: remember the distances between iterations in a “diameter cache”
Diameter Function Complexity: $F_d$

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements

Best: remember the distances between iterations in a “diameter cache”
Diameter Function Complexity: Fd

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements

Best: remember the distances between iterations in a “diameter cache”
Diameter Function Complexity: $F_d$

Non-Euclidean cluster diameter: Maximum pairwise distance between cluster elements

Best: remember the distances between iterations in a “diameter cache”
Diameter Function Complexity: $F_d$

Non-Euclidean cluster diameter: Maximum pairwise distance between cluster elements

Best: remember the distances between iterations in a “diameter cache”

$$diam(\{C^{k+1} \cup e_j\}) = \max\{dc[j], \text{dist}(e_j, e^k)\}$$
Diameter Function Complexity: Fd

Non-Euclidean cluster diameter:
Maximum pairwise distance between cluster elements

Best: remember the distances between iterations in a “diameter cache”

\[
\text{diam(}\{C^{k+1} \cup e_j\}\) = \max\{dc[j], \text{dist}(e_j, e^k)\}
\]

\[
O(Fd) = O(\text{dist}(e_i, e_j)) \Rightarrow O(\text{QTC}) = O(N^3 \times \text{dist}(e_i, e_j))
\]
Expected complexity

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>7</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>13</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>15</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
<td>12</td>
<td>13</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Graph showing connections between nodes]
### Expected complexity

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(\Delta\)

- **M**: Max cluster size
- \(O(N/M)\) clusters, \(O(M)\) elements
- \(+\)
- \(O(N)\) clusters, \(O(1)\) elements
### Expected complexity

**M**: Max cluster size

| Δ  | 1 7 | 2 3 6 8 12 13 | 3 2 6 9 12 13 14 | 4 8 10 | 5 8 9 | 6 2 3 12 13 17 | 7 1 14 | 8 2 15 18 | 9 3 5 12 13 15 | ...
|----|-----|----------------|------------------|-------|-----|----------------|------|--------|-----------|------|

**Algorithm**:

- O(N/M) foreach (maximal cluster at step “k”) O(N)
- O(N) foreach (unclustered element) O(N)
- O(M) while (cand. cluster can grow) O(1)
- O(Δ*Fd) find (element : min diameter) O(Δ*Fd)
Expected complexity

\[ O(QTC) = O(N^2 \cdot \Delta) \cdot O(Fd) \]

M: Max cluster size

O(N/M) clusters, O(M) elements

+ O(N) clusters, O(1) elements
Tighter Bound

\[ \sum_{i=0}^{k} \left( (n - \sum_{j=0}^{i-1} m_j) \times m_i \times \Delta_i \right) \]
Implementation issues

Pairwise distance is $O(1)$, but w/ huge constants

$N^2$ table: $O(1)$ dist, $O(N)$ find
Implementation issues

Pairwise distance is O(1), but w/ huge constants

elements: \( N \)

\[ \begin{array}{c|c|c|c|c}
\hline
\text{i} & \text{e}_i & \ldots \\
\hline
\end{array} \]

\( N^2 \) table: \( O(1) \) dist, \( O(N) \) find

\( N \times \Delta \) table: \( O(\Delta) \) dist, \( O(\Delta^2) \) find
Implementation issues

Pairwise distance is $O(1)$, but w/ huge constants

$$N^2 + N \Delta$$ tables: $O(1)$ dist, $O(\Delta)$ find
Implementation issues

elements: $N \quad e_i \quad \ldots$

proximity: $N^2 + N*\Delta$

\[ i \]

\[ N \]

\[ e_i \]

\[ \ldots \]
Implementation issues

unclustered elements: \( N \)

proximity: \( N^2 + N \Delta \)
## Implementation issues

<table>
<thead>
<tr>
<th>j</th>
<th>e₁</th>
<th>e₂</th>
<th>...</th>
</tr>
</thead>
</table>

- **unclustered elements**: \( N \)

- **proximity**: \( N^2 + N \Delta \)

- **clustered elements mask**: \( N \)

- **current clustered elements mask**: \( N \)

- **diameter cache**: \( \Delta \)

- **degrees**: \( N \)

### Full storage total:
\[
4 \times N^2 + 4 \times N \Delta + 4 \times 2 \times N + 2 \times N + 4 \times \Delta
\]

### Compact total:
\[
4 \times 2 \times N \Delta + 4 \times 2 \times N + 2 \times N + 4 \times \Delta
\]
GPU Implementation issues

unclustered elements: \( N \) 

proximity: \( N^2 + N\Delta \)

clustered elements mask: \( N \)

current clustered elements mask: \( N \)

diameter cache: \( \Delta \)

degrees: \( N \)

Full storage total: \( 4N^2 + 4N\Delta + N\text{TBC} + 4\times2N + N + 4\Delta\text{TBC} \)

Compact total: \( 4\times2N\Delta + N\text{TBC} + 4\times2N + N + 4\Delta\text{TBC} \)
Parallelization

\texttt{QT\_Clust}(G, d)\{
    \text{if}(|G| \leq 1) \text{ then output } G, \text{ else do}
    \textbf{foreach } p_i \in G
        \text{ flag = TRUE }
        C_i = \{p_i\} /* Ci is the cluster started by element pi */
    \textbf{while} (flag == TRUE) \&\& (C_i \neq G)
        \textbf{find } p_j \in (G - C_i) : \textbf{diameter} (C_i \cup \{p_j\}) \text{ is minimum}
        \text{if} (\text{diameter}(C_i \cup \{p_j\}) > d)
            \text{then flag = FALSE}
        \text{else } C_i = C_i \cup \{p_j\} /* Add pj to cluster } C_i */
        \text{identify set } C \in \{C_1, C_2, \cdots, C_{|G|}\} \text{ with maximum cardinality}
    \text{output } C
    \text{call } QT\_Clust(G - C, d) /* \text{recursion} */
\}
Parallelization

QT_Clust(G, d){
    if(|G| ≤ 1) then output G, else do
        foreach \( p_i \in G \)
            flag = TRUE
            \( C_i = \{ p_i \} \) /* \( C_i \) is the cluster started by element \( p_i \) */
        while( flag==TRUE) && (\( C_i \neq G \))
            find \( p_j \in (G - C_i) : \) diameter(\( C_i \cup \{ p_j \} \)) is minimum
            if( diameter(\( C_i \cup \{ p_j \} \) > d )
                then flag = FALSE
                else \( C_i = C_i \cup \{ p_j \} \) /* Add \( p_j \) to cluster \( C_i \) */
            identify set \( C \in \{C_1, C_2, \ldots, C_{|G|} \} \) with maximum cardinality
        output \( C \)
        call QT_Clust(G - C, d) /* recurse */
}

Reduction
while (flag == TRUE) && (Ci ≠ G) 
find pj ∈ (G – Ci) : diameter(Ci ∪ {pj}) is minimum
GPU register usage
poor man's unrolling

while( Ci can grow ){
    while( off + tid < degree ){
        e = table[off + tid]
        d = dist_to_clust(e)
        we = reduce_distances_in_TB()
        add_winner_to_cluster( we )
    }
}
GPU register usage

poor man's unrolling

while( Ci can grow ){
    while( off + tid < degree ){
        e = table[off + tid]
        d = dist_to_clust(e)
        we = reduce_distances_in_TB()
        add_winner_to_cluster( we )
    }
}

IPDPS 2012, Anthony Danalis
GPU register usage

poor man's unrolling

\[ r1 = \text{table}[\text{off} + \text{tid}] \]

\[
\text{while( Ci can grow )}\{
\text{dist_to_clust}( r1 )
\]

\[ ... \]

\[
\text{while( off + TC + tid < degree )}\{
\text{e} = \text{table}[\text{off} + \text{TC} + \text{tid}]
\]

\[
\text{d} = \text{dist_to_clust}(\text{e})
\]

\[
\text{we} = \text{reduce_distances_in_TB}()
\]

\[
\text{add_winner_to_cluster}(\text{we})
\]

\}
Performance (CPU vs GPU)

Execution Time (sec)

Number of Proteins

- CPU, Threads: 1
- CPU, Threads: 3
- CPU, Threads: 6
- GPU
Performance (Time vs Size, GPU)

QTC on Nvidia c2070 GPU (Euclidean data on 20x20 plane)

Time to Solution (min)

Problem Size (element count)
Performance (Time vs Threshold, GPU)

![Graph showing Performance](graph.png)
Performance (Time vs Threshold, GPU)

- Compact, N=16K
- Full Storage, N=16K
  - $\Delta^2$ (scaled)
  - $\Delta$ (scaled)
Performance (Time vs Threshold, GPU)

![Graph showing performance comparison between Compact and Full Storage with various thresholds and complexities.](image-url)
Performance (Parallel Efficiency & Time, GPU)
N=80K, d=1
Conclusions

- Worst case complexity: $O(N^3)$
- Expected complexity: $O(N^2 \cdot \Delta \cdot Fd)$
- Choice of threshold has huge impact
- GPU implementation faster than CPU
- FLOPS are cheap, but far from free
- Strong scaling is good for large problems