Seeding Strategies for Lloyd‘s kmeans

C++ implementation of Ostrovsky, Rabani, Schulman and Swamy‘s ideas

Mirko Speth

22.07.2016

Overview

Five approaches to seed Lloyd‘s k-means

- A: Random Sampling
- B: Greedy Deletion
- C: Linear time algorithm
- D: Linear time constant factor algorithm
- E: Polynomial Time Approximation Scheme (PTAS)

A: Random Sampling

$O(knd)$

1. Select two random points c_1 and c_2 with probability $||c_1 - c_2||^2$
2. Perform a ball-k-means step
3. Repeat: add another point c_{i+1} with probability $min_{j \in \{1 \ldots i\}} ||c_{i+1} - c_j||^2$

A: Random Sampling

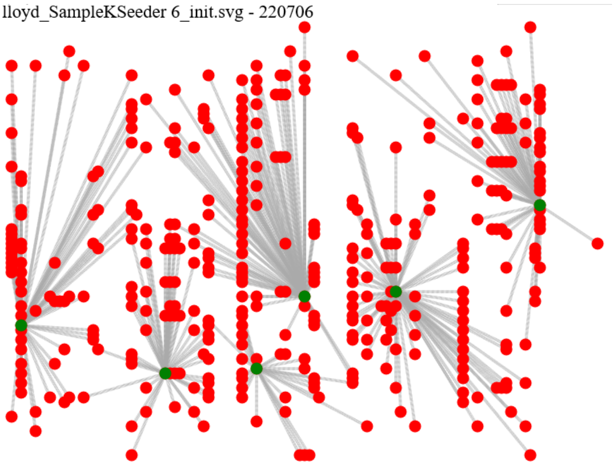

Figure: SVG-Output of SamplekSeeder on pbl395.tsp (k=6)

B: Greedy Deletion

```
Pointset GreedyDelSeeder::seed(Pointset init) const {
  Pointset sites = init;
  while (sites.size() > (unsigned int) k) {

    // B1: get best and second best center for each customer
    Partition part = Partition(&customers, sites);

    // B2: pick the center for which Tx is minimum
    int bestid = part.getMinTx();

    // B3: delete chosen partition and move points to centroid of voronoi region
    part.delete_set_from_partition(bestid);
    sites = part.centroids();
  }
  return sites;
}
```

B: Greedy Deletion

Running time: outer loop $(n - k)$ iterations $\Rightarrow O(n)$
Partitioning: $O(n^2 d)$
No speed loss for second best
$\Rightarrow$ together: $O(n^3 d)$

C: Linear time algorithm

```
Pointset LTSeeder::seed() const {

  //C1
  double e = instance.eps();
  double p1 = sqrt(e);
  int N = (int)(2 * k / (1 - 5 * p1) + 2 * log(2 / p1)
    / pow((1 - 5 * p1), 2));

  SampleKSeeder samplekseeder(instance, N);
  Pointset S = samplekseeder.seed();

  //C2
  Partition partition = Partition(&customers, S);
  Pointset sdach = partition.centroids();

  GreedyDelSeeder greedydelseeder(instance, k);

  return greedydelseeder.seed(sdach);
}
```

C: Linear time algorithm

$O(nkd + k^3d)$

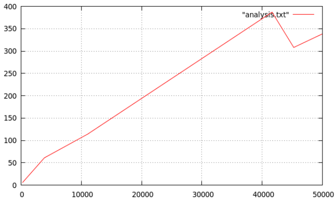

Figure: complexity in #customers

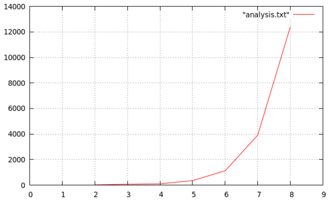

Figure: complexity in #sites

D: Linear time constant factor algorithm

```
Pointset DSeeder::seed() const {
    // D1 (obtain k initial centres using last seeding strategy)
  Pointset init = (LTSeeder(instance, k)).seed();

  // D2 (run a ball-k-means step)
  return ballkmeansstep(init);
}
```

D: Linear time algorithm

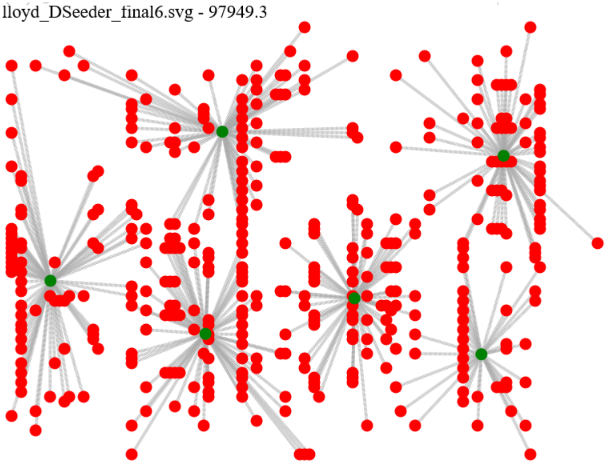

Figure: SVG-Output of DSeeder on pbl395.tsp (k=6)

E: Polynomial Time Approximation Scheme (PTAS)

```
Pointset ESeeder::seed() const {
  SampleKSeeder samplek(instance, k);
  Partition part = Partition(&customers, samplek.seed()
    );
  return part.centroid_estimation(instance.omega,
    instance.eps);
}
```

centroid estimation

```
for s in sites:
    select expanded Voronoi region V[s]
    choose a random subset R[s] of V[s]
    foreach subset A of size 1/wb:
        T[s] = T[s], centroid(A)
foreach set B in {{x1,...xk} : xi in T[i]}:
    if error(B) < error (best) then best = B
```

PTAS analysis

Ostrovski et al:
error at most $(1 + \omega) * OPT$ with probability γ^k
(for some constant γ)

E: PTAS

$$O(2^{(\frac{4k}{\beta\omega})} n * d)$$

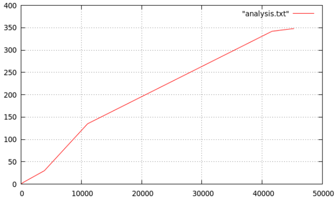

Figure: complexity in #customers

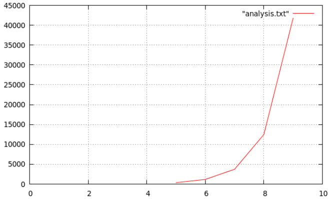

Figure: complexity in #sites