Early Introduction to Parallel Computing via Applications in Data Analytics

Sukhamay Kundu
Computer Science Department
Louisiana State University, Baton Rouge, LA 70803
kundu@csc.lsu.edu

***Abstract*—Data analytics is an important application-area for parallel computation and is suitable for early introduction to parallel computation for undergraduate and graduate students. We present a simple, important clustering problem in Data Analytics and the parallelization of an efficient sequential solution algorithm for it. We have used this example successfully in a 1st yr. graduate Algorithm course in Fall-2018 to teach the application of parallel prefix-sum and other related techniques for an early introduction to parallel computing. We plan to use it for a 4th yr. undergraduate course in Spring-2019.**

***Index Terms*—clustering, data analytics, parallel computing**

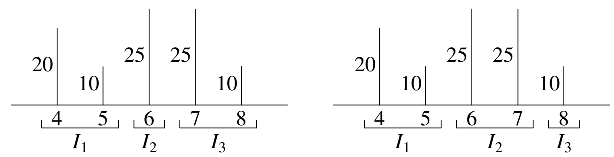
I. The clustering problem

Clustering of large data-sets into a small number of clusters is a common and important problem in Data Analytics. We consider here clustering of 1-dimensional data for three reasons: (1) it has many important applications, (2) it has a specialized efficient sequential algorithm, and (3) it offers multiple opportunities to use parallel computation. These make it a suitable path-way for an early introduction to parallel computation for senior undergraduates and beginning graduates.

Problem-OptC: Let $S = \{s_i : 1 \leq i \leq N\}$, $s_i < s_j$ for $i < j$, be the scores in a measurement (say, a class-test) and let $0 < f_i = f(s_i)$ the frequency of s_i. We want to find an optimal partition of S into $n \geq 2$ disjoint clusters or intervals $I_1, I_2, \cdots, I_n$ (in the left to right order) of consecutive scores for assigning letter-grades to each s_i so that the scores in each I_m are "close" to each other. We measure the closeness of scores in I_m by $E_m = \sum_k f_k(s_k - \mu_m)^2$, where $\mu_m = (\sum_k f_k s_k)/(\sum_k f_k)$ = the average score in I_m, and all the sums are over $s_k \in I_m$. An optimal n-clustering or n-partition is one that minimizes $E = \sum_m w_m E(I_m)$ for a given cluster-weights $w_m > 0, 1 \leq m \leq n$, which can be proportional to the grade-points associated with the letter-grades.

Example. Fig. 1 shows the optimal 3-clustering of 5 equally spaced scores for two different weights. In Fig. 1(i), s_3 = 6 and s_4 = 7 are not in the same I_j because that would make $E(I_j)$ and E too large because of large f_3 and f_4. In Fig. 1(ii), the large w_3 requires $E(I_3)$ to be small, forcing $I_3 = \{s_5\}$ and $I_2 = \{s_3, s_4\}$. Fig. 2 shows that modifying s_5 to 9 we get the same optimal 3-clustering for both weights W_1 and W_2 as in Fig. 1. The extra gap between s_4 and s_5 makes s_5 form a cluster by itself and the other two clusters are as in Fig. 1(ii).

In what follows, we write $I_{i,j}$ (or, in short I_{ij}) for $\{s_k : i < k \leq j\}$, $0 \leq i < j \leq N$; the associated average and error are denoted by μ_{ij} and E_{ij}, respectively.



(i) $W_1 = \langle 1/3, 1/3, 1/3 \rangle$. (ii) $W_2 = \langle 1/5, 1/5, 3/5 \rangle$.

Fig. 1. The optimum 3-clusterings of S = {4(20), 5(10), 6(25), 7(25), 8(10)}, with frequencies shown in parentheses, for weights W_1 and W_2.

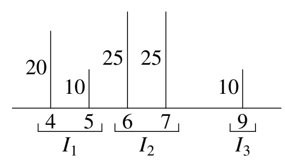


Fig. 2. The optimum 3-clustering of S' = {4(20), 5(10), 6(25), 7(25), 9(10)} for both weights W_1 and W_2 as in Fig. 1.

II. An Efficient Sequential Algorithm

We solve OptC by converting it into a special shortest-path problem in the complete acyclic digraph G on nodes $\{x_0, x_1, \cdots, x_N\}$ and links $(x_i, x_j), i < j$. A link (x_i, x_j) represents the cluster-interval I_{ij} and has cost $c(x_i, x_j) = E(I_{ij})$. See Fig. 3. The n-step x_0x_N-paths correspond to n-clusterings of S, and vice-versa; likewise, for the shortest n-step x_0x_N-paths the optimal n-clusterings of S. If a link (x_i, x_j) is the k^{th} step of an x_0x_N-path, it contributes $w_k E_{ij}$ to the path-cost.

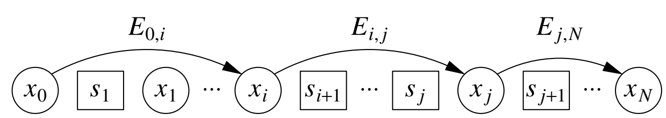


Fig. 3. A 3-step x_0x_N-path; its cost is $w_1E_{0,i} + w_2E_{i,j} + w_3E_{j,N}$.

Let $d_k(x_j)$ = the length of a shortest k-step x_0x_j-path for $j \geq k$ and $k \leq n$. If we know all E_{ij}, then for each fixed $k, 1 \leq k < n-1$ we can compute the row of $d_{k+1}(x_j), j \geq k+1$ from the row of $d_k(x_i), i \geq k$, from equns. (1)-(3) below.

$$d_1(x_j) = w_1 E_{0j} \quad (1)$$

$$d_{k+1}(x_j) = \min\{d_k(x_i) + w_{k+1}E_{ij} : k \leq i < j\} \quad (2)$$

$$d_n(x_N) = \min\{d_{n-1}(x_i) + w_n E_{iN} : n-1 \leq i < N\} \quad (3)$$

For n = 2, we need only E_{0j} and E_{jN} for $1 \leq j < N$; we need all E_{ij}'s for $n > 2$. Fig. 4, where we use the short notation d_{kj} for $d_k(x_j)$, is a graphical representation of equns.

(2)-(3) for the case $n = 4$. Here, a link from $d_{k,i}$ to $d_{k+1,j}$ indicates that the minimum for $d_{k+1,j}$ involves the term $d_{k,i} + w_{k+1}E_{i,j}$. It also shows that to compute $d_n(x_N)$ we need only $d_{k,j}$, $1 \le k < n, k \le j \le N - (n-k)$.

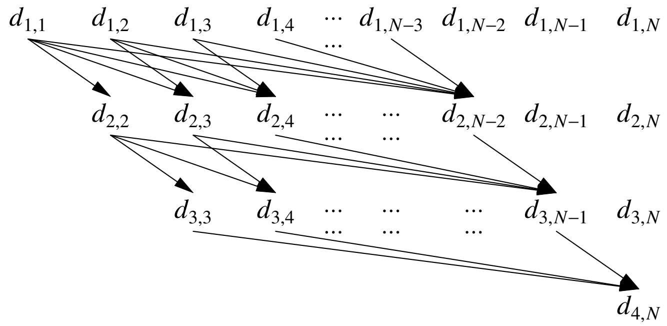


Fig. 4. A graphical representation of equns. (2)-(3) for $n = 4$.

To compute E_{ij}'s efficiently, we let $F_{ij} = \sum_k f_k$, $S_{ij} = \sum_k f_k s_k$, and $SS_{ij} = \sum_k f_k s_k^2$ for $0 \le i < j \le N$, where each sum is taken over $s_k \in I_{ij}$. Then, $\mu_{ij} = S_{ij}/F_{ij}$ and $E_{ij} = SS_{ij} - F_{ij}\mu_{ij}^2$. We can compute all F_{ij}'s in $O(N^2)$ time using equn. (4) below with the initializations $F_{i,i+1} = f_{i+1}$. A similar remark holds for S_{ij}'s, SS_{ij}'s, the equns. (5)-(6), and the initializations $S_{i,i+1} = f_{i+1}s_{i+1}$ and $SS_{i,i+1} = f_{i+1}s_{i+1}^2$. Thus, we can compute all E_{ij}'s in $O(N^2)$ time and an optimal n-clustering in $O(nN^2)$ time by the algorithm SeqOptC below.

$$F_{i,j+1} = F_{i,j} + f_{j+1}, \text{ for } i < j < N \tag{4}$$

$$S_{i,j+1} = S_{i,j} + f_{j+1}s_{j+1}, \text{ for } i < j < N \tag{5}$$

$$SS_{i,j+1} = SS_{i,j} + f_{j+1}s_{j+1}^2, \text{ for } i < j < N \tag{6}$$

Algorithm SeqOptC //sequential algorithm for OptC

1. For $i = 0, 1, \cdots, N-1$, do the following:
   (a) Let $F_{i,i+1} = f_{i+1}$, $S_{i,i+1} = f_{i+1}s_{i+1}$, $SS_{i,i+1} = f_{i+1}s_{i+1}^2$. Also, let $\mu_{i,i+1} = s_{i+1}$ and $E_{i,i+1} = 0$.
   (b) For $i+2 \le j \le N$, compute $F_{i,j}$, $S_{i,j}$, $SS_{i,j}$, $\mu_{i,j}$, and $E_{i,j}$ using equns. (4)-(6) and let $\mu_{i,j} = S_{i,j}/F_{i,j}$ and $E_{i,j} = SS_{i,j} - F_{i,j}\mu_{i,j}^2$.
2. Let $d_1(x_j) = w_1 E_{0,j}$ for $1 \le j \le N - (n-1)$.
3. For each $k = 1, 2, \cdots, n-1$, compute $d_{k+1}(x_j)$'s from $d_k(x_j)$'s for $k+1 \le j \le N - (n-k-1)$ using equn. (2).
4. Finally, compute $d_n(x_N)$ using equn. (3).

To compute the intervals of an optimal n-partition of S, we can keep track of an $i = i_{k,j}$ which gives the minimum for $d_{k+1}(x_j)$ and an $i = i_{n-1,N}$ for the minimum for $d_n(x_N)$ in steps (3)-(4) of the algorithm SeqOptC. Let $i_{n-1} = i_{n-1,N}$, $i_{n-2} = i_{n-2,j}$ for $j = i_{n-1}$, $i_{n-3} = i_{n-3,j}$ for $j = i_{n-2}$, and so on. The successive intervals of an optimal n-patition or n-clustering are $I_{0,i_1}, I_{i_1,i_2}, \cdots, I_{i_{n-1},N}$.

III. Parallelizations of Algorithm SeqOptC

A. Case of N agents or CPUs

We compute $(j+1)^{th}$ column of $F_{i,j+1}$'s in Fig. 5 in parallel from j^{th} column using equn. (4) and $F_{j,j+1} = f_{j+1}$. Alternatively, we can initialize 1^{st} diagonal items $F_{i,i+1} = f_{i+1}$, $0 \le i < N$, in Fig. 5 in parallel, then compute 2^{nd} diagonal items $F_{i,i+2} = F_{i,i+1} + f_{i+2}$, $0 \le i < N-1$ from 1^{st} diagonal items in parallel, and so on. Both methods take $O(N)$ time to compute all $F_{i,j}$'s. Likewise, we compute all $S_{i,j}$'s and $SS_{i,j}$'s in $O(N)$ time. Finally, we compute $\mu_{i,j}$'s and then $E_{i,j}$'s in parallel (by rows, columns, or diagonals) in time $O(N)$. This completes parallelization of step (1) in algorithm SeqOptC.

Time						
1	2	3	4	⋯	$N-1$	N
$F_{0,1}$	$F_{0,2}$	$F_{0,3}$	$F_{0,4}$	⋯	$F_{0,N-1}$	$F_{0,N}$
	$F_{1,2}$	$F_{1,3}$	$F_{1,4}$	⋯	$F_{1,N-1}$	$F_{1,N}$
		$F_{2,3}$	$F_{2,4}$	⋯	$F_{2,N-1}$	$F_{2,N}$
			⋯	⋯	⋯	⋯
					$F_{N-2,N-1}$	$F_{N-2,N}$
						$F_{N-1,N}$

Fig. 5. Computing F_{ij}'s in parallel by columns.

Because it takes $O(log N)$ time to compute the minimum in equn. (2) using N agents, we can compute all $d_{k,j}$, $1 \le k < n$ and $k \le j \le N-(n-k)$, in time $O(nNlogN)$. This completes parallelization of step (3) in SeqOptC. Clearly, step (2) can be done in $O(1)$ time and step (4) in $O(log\ N)$ time. This give total time $O(nNlog\ N)$ for OptC using N agents.

B. Case of N^2 or, more precisely, $N(N+1)/2$ agents

Each row $F_{i,j}$, $j \ge i+1$, in Fig. 5 is the prefix-sum of the base array $[f_{i+1}, f_{i+2}, \cdots, f_N]$. We use $N - i$ agents to initialize the base array in $O(1)$ time and to compute the prefix-sum in $O(log\ N)$ time. Thus, we compute all rows in Fig. 5 in parallel using $N(N+1)/2$ agents in time $O(log\, N)$. (Note: we can also compute $F_{i,j}$'s columnwise as prefix-sums; for the j^{th} column bottom to top, the base array is $[f_j, f_{j-1}, \cdots, f_1]$.) We likewise compute all $S_{i,j}$ and $SS_{i,j}$ (by rows or columns) as prefix-sums in $O(log\ N)$ time. Now, we compute all $\mu_{i,j}$ and $E_{i,j}$ in parallel in $O(1)$ time.

Next, we compute $d_{1,j}, 1 \le j \le N-(n-1)$ in $O(1)$ time using N agents. Then, for each fixed $k < n-1$, we compute the min in each $d_{k+1,j}$ in equn. (2) in time $O(log\, N)$ using $j-k$ agents. Thus, computation of the row of $d_{k+1,j}$'s from its previous row of $d_{k,j}$'s takes $O(log\ N)$ time using at most $N(N+1)/2$ agents. We take additional $O(log\ N)$ time to compute $d_{n,N}$. This gives $O(n\log N)$ time for OptC using $N(N+1)/2$ agents.

IV. Conclusion

We present here several ways of parallelizing a well-known efficient sequential algorithm for optimal clustering of a set of 1-dimensional data points into $n \ge 2$ parts for an early introduction to parallel computation for senior level undergraduates and beginning graduates via applications in Data Analytics. See [1] for some other relevant works.

References

[1] Topics in parallel and distributed computing: introducing concurrency in undergraduate courses (eds. S.K. Prasad, A. Gupta, A.L. Rosenberg, A. Sussman, and C. Weems, Morgan Kaufmann, 2015.