Parallelising the k-Medoids Clustering Problem Using Space-Partitioning

Alejandro Arbelaez
JFLI / University of Tokyo
Tokyo, Japan
arbelaez@is.s.u-tokyo.ac.jp

Luis Quesada
CTVR, Cork Constraint Computation Centre
University College Cork, Ireland
l.quesada@4c.ucc.ie

Abstract

The k-medoids problem is a combinatorial optimisation problem with multiples applications in Resource Allocation, Mobile Computing, Sensor Networks and Telecommunications. Real instances of this problem involve hundreds of thousands of points and thousands of medoids. Despite the proliferation of parallel architectures, this problem has been mostly tackled using sequential approaches. In this paper, we study the impact of space-partitioning techniques on the performance of parallel local search algorithms to tackle the k-medoids clustering problem, and compare these results with the ones obtained using sampling. Our experiments suggest that approaches relying on partitioning scale more while preserving the quality of the solution.

1 Introduction

The k-medoids problem (i.e., the problem of selecting a subset of k points (the medoids) such that the average distance from any point to its closest medoid is minimized) is a common problem in several application domains. In Resource Allocation, we find instances of this problem when a company is planning to open a set of branches and wants to locate those branches taking into account the proximity to their clients(Mouratidis, Papadias, and Papadimitriou 2008). In the context of mobile computing, it is a concern to save communication cost when devices need to select super-nodes among them, which collect, aggregate and forward to the location server messages received from their vicinity taking into account that the wireless medium is prone to errors thus forcing the devices to be close to some super- node (Wang et al. 2003). Medoid queries also arise in the field of sensor networks. Typically, in order to prolong the battery life, only a fraction of the sensors are kept awake, and used as representatives for a particular region of the monitored area (Hassani Bijarbooneh et al. 2011). In the field of telecommunication, k-medoid queries are of primary importance in the design of optical fiber deployment strategies. Telecommunication companies are concerned with the optimal positioning of their central offices such that the cable needed to connect central offices to clients is minimized (Cambazard et al. 2012).

The k-medoid problem is known to be NP-hard (Papadimitriou 1981; Megiddo and Supowit 1984). Complete approaches to solve this problem based on systematic search have been suggested. However, these approaches do not scale to the size of real-wold instances, which motivates our consideration of local search approaches. In the literature we certainly find local search approaches to the k-medoids problem (Michel and Van Hentenryck 2004; Cambazard et al. 2012), but these approaches have been mostly conceived to be run on single core architectures, thus not taking advantage of current hardware architectures such as grid based platforms and supercomputers with thousands or tens of thousands of cores. This motivates the exploration of parallel methods to tackle this problem. The most common approach to devise parallel local search algorithms consists in exploring the search space in parallel, either independently or cooperatively with some communication between the solvers. The non-cooperative approach (so-called parallel portfolio) executes different algorithms (or the same one with different random seeds) independently with no cooperation and the global search is stopped when a solution is observed or a given timeout is reached. The parallel portfolio is usually equipped with a knowledge sharing in order to allow the solvers in the portfolio to compete and cooperate to solve a given instance. However, cooperation must be limited due to an important communication overhead, which might degrade the overall solving time. It is worth noticing that parallel portfolio solvers have shown a great scalability up to tens of hundreds of cores to solve classical CSP (Caniou et al. 2011) and SAT (Arbelaez and Codognet 2012) benchmarks, and even up to 8000 cores for the Costas array problem (Diaz et al. 2012). However, these solvers have been mainly devoted to tackle decision problems and limited attention have been given to optimization problems.

In this paper, we are interested in solving very large instances of the k-medoids problem. We show that using parallel local search helps to improve the scalability of the sequential algorithm, which is measured in terms of the quality of the solution within the same time (wall clock time) with respect to the sequential algorithm.

This paper is organized as follows: Section 2 presents the general scheme of the local search algorithm used to tackle the k-medoids problem; Section 3 depicts a set of search space partitioning methods which will be used to devise a parallel local search algorithm; Section 4 reports extensive experimental results of the parallel algorithms; and Section 5 presents general conclusions and perspectives of future work.

2 Local search

The parallel approaches discussed in Section 3 builds on top of the GEO approach (Cambazard et al. 2012). GEO is a local search approach to the k-medoid problem, which is NP-hard (Papadimitriou 1981; Megiddo and Supowit 1984).

Definition 1 *In the k-medoid problem (also known as the Euclidean p-median problem), we are given a set* $S = \{(\boldsymbol{x}_1, y_1), (x_2, y_2), \dots (x_m, y_m)\}$ *of points in a plane and we want to find a set* $W = \{(z_1, t_1), (z_2, t_2), \dots, (z_k, t_k)\}$ *of points, subset of S, so as to minimise:*

$$\sum_{i=1}^{m} \min_{1 \leq j \leq k} \sqrt{(x_i - z_j)^2 + (y_i - t_j)^2}$$

In GEO, the objective function is incrementality maintened by exploiting computational geometry through the computation of Voronoi cells. The main observation in GEO is that, in practice, very few points in S (the set of points) are likely to have a new closest or second closest facility. The left part of Figure 1 shows an example of opening a new facility p_j. Facilities are indicated by plain circles and points by crosses. The points for which p_j is the new closest facility are shown in squares whereas the points for which p_j is the new second closest facility are shown in circles. Only a very small number of points of the m points of S are affected. In (Cambazard et al. 2012) the authors focus on approaches that do not maintain a priority queue of facilities per point. The two closest facilities of each point are maintained by using computational geometry techniques since points affected by the opening and closing of a facility can be characterised through the notion of Voronoi cell. The right part of Figure 1 shows the Voronoi cell of point p_j.

Algorithm 1 TABUSEARCH()

```
Initialize $W$ randomly, $C = S - W$,
While (end condition not reached)
  $p^*_j = -1$, $bestDelta = \infty$, $cobj = obj$
  For each $p_j \in W - T$ and as long as $bestDelta > 0$
    CloseFacility($p_j$)
    // updates obj and all $\Delta^+_i$
    $p_{ibest} = \arg\min_{\{p_i \in C - T\}}(\Delta^+_i)$
    If $(\Delta^+_{ibest} + (cobj - obj)) < bestDelta$
      $p^*_j = p_j$, $bestDelta = \Delta^+_{ibest} + (cobj - obj)$
    OpenFacility($p_j$)
    // updates obj and all $\Delta^+_i$
If ($bestDelta$ > 0)
  $p^*_j$ = a random point in $W - T$
CloseFacility($p^*_j$)
OpenFacility ($\arg\min_{\{p_i \in C - T\}}(\Delta^+_i)$)
update tabu list $T$
```

Algorithm 1 presents the general scheme of the tabu search used in GEO. The initial p facilities are chosen randomly. The tabu mechanism is very simple. It prevents a point that was a facility in the last t iterations, where t is the length of the tabu-list, from becoming a facility again. The tabu-list is denoted T in this algorithm. The first improving move found is performed. If no improving move exists, the facility to close is chosen randomly and the new best location for this facility is opened. After a number of non-improving iterations, the search is restarted from p random facilities.

Algorithm 1 assumes that two methods are available for opening and closing a facility (resp. OpenFacility and CloseFacility) while incrementally maintaining the value of the objective function (denoted obj) and Δ^+. This algorithm is very like the PAM algorithm (Kaufman and Rousseeuw 1990); the only difference being that PAM is selecting the best move rather than the first improving one. However this algorithm is enhanced with the incremental mechanisms, and the tabu metaheuristic, introduced in warehouse location for a similar neighborhood.

We summarize the time complexity of GEO in Table 1. As explained in (Cambazard et al. 2012), the time complexity is dominated by the time spent in the initialisation of the data structures (which takes place each time the algorithm is restarted) and the time spent in opening/closing a facility. The time spent in opening/closing a facility is dominated by the time spent in maintaining the two closest facilities to every point. In Table 1, m is the number of nodes, p is the number of facilities, k is the number of nodes that have p_j as a closest or second closest, and $k^{'}$ is an upper bound on k useful to express the complexity (it is the number of points contained in the enclosing rectangle of the extended Voronoi cell) as we still have $k^{'} \leq m$. In (Cambazard et al. 2012) the authors also show that the space complexity of GEO is $O(m + p)$.

3 Parallel Algorithms

3.1 CLARA based approach

Clustering Large Applications (CLARA) (Kaufman and Rousseeuw 1990) reduces the burden of computing the average distance by generating random samples from the set of points and executing PAM on those. In (Zhang et al. 2004) the authors use the parallel portfolio architecture to speedup the search by executing multiple copies of CLARA without cooperation. In this paper, we exploit the same principle by executing multiple copies of Algorithm 1 on a random sample of the points. The performance of the algorithm is proportional to the sample size. The larger the sample the better the quality of the solution. Finally, the process with an overall best solution cost indicates the global solution.

3.2 Space partitioning

In this section we review two main space partitioning techniques to divide the problem space into smaller independent spaces; each core (or processor) is assigned a subspace of the problem space. A local search algorithm subsequently explores its subspace until the stopping criteria is met. In addition, the number of facilities is also divided proportionally to the size of the sub-spaces. The space partitioning technique has been previously used in the literature (see (Crainic and Toulouse 2003) for a complete de-

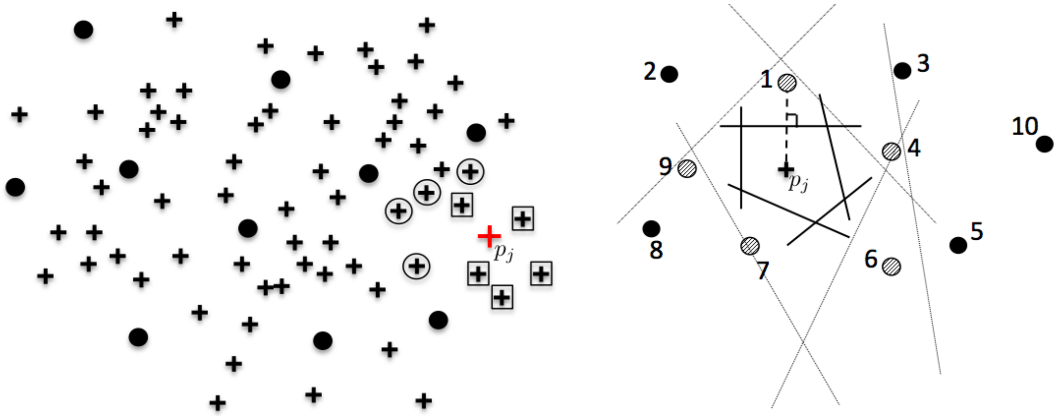


Figure 1: Example of opening a facility p_j on left. Facilities are shown as plain circles, points as crosses and the points having p_j as their closest (resp. second closest) facility are shown in a square (resp. a circle). Example of the Voronoi cell of p_j ($\mathcal{V}(p_j)$) on right. The boundary of the cell is indicated by the dashed nodes so $\mathcal{B}(p_j) = \{1, 4, 6, 7, 9\}$.

Table 1: Summary of time complexities of the different schemes

operation	GEO
initialization	$O(pm + p\log(p) + m\log(m))$
open a facility p_j	$O(plog(p) + \sqrt{m} + k^{'})$
close a facility p_j	$expected$: $O(klog(p))$, worst-case: $O(kp)$

scription) to improve the performance of local search algorithms to solve optimization problems. However, unlike the topologies mentioned in (Crainic and Toulouse 2003), in which sub-processes have access to the entire search space, in this work the search space is heuristically divided and sub-processes have access to a limited portions of the search space.

In this paper, we employ two space partitioning techniques to divide the search space into sub-spaces. Thus, the parallel algorithm works in three phases.

Phase 1 (Pre-processing phase) The master processor divides the problem space into several partitions, one for each processor;

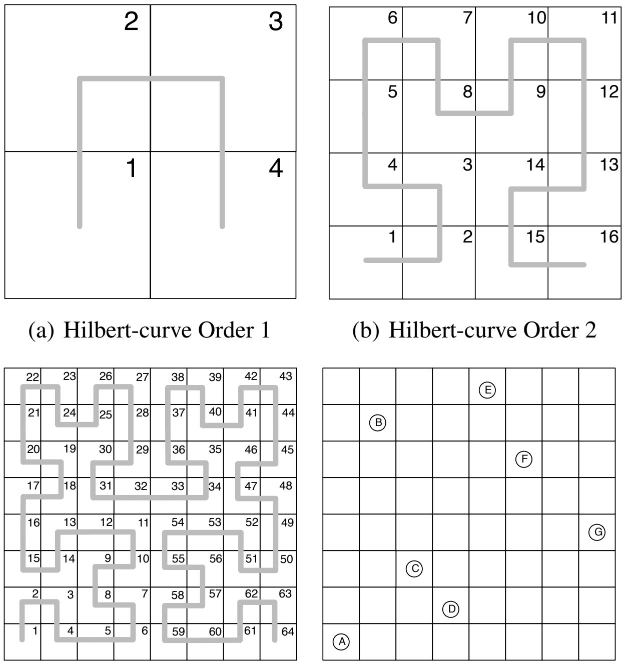
Phase 2 (Solving phase) solve in parallel each sub-problem until a given time cutoff is reached;

Phase 3 (Ending phase) aggregate partial solutions and compute the global solution. This step requires computing the closest facility for each node, the time complexity of this phase is $O(pm)$. However, it is important to notice that the partial solution obtained in the previous phase can be observed as an upper bound of the global solution.

Space-filling curves Space-filling curves (Sagan 1994) are widely used to reduce dimensionality from a N-dimensional space into a one-dimensional space. An important feature of space-filling curves is that they preserve locality, that is, points that are close in the N-dimensional space are also close in the one-dimensional space. Informally speaking, a space-filling curve can be obtained by drawing an "imaginary" line that passes through every point in the two dimensional space, starting in the initial point and ending in the final point. Each point is then characterized by a unique index, which imposes a linear ordering of the elements. Examples of space-filling curves include: Hilbert (Butz 1971), Peano (Rosenkrantz and Fagin 1984), and RBG (Faloutsos 1988) curves. In this paper, we focus our attention in the dimension sort curve, which sorts the elements according to a given axis and the Hilbert curve, one of the most popular space-filling curves (Mokbel, Aref, and Kamel 2002).

Figure 2 illustrates the Hilbert-curve with orders 1 (Figure 2(a)), 2 (Figure 2(b)) and 3 (Figure 2(c)). In general, the n-th order Hilbert curve consists of four concatenated copies of the Hilbert curve of the previous (n-1) order located in the upper left quadrant, upper right quadrant, bottom left quadrant (rotated 90 degrees counter-clockwise), and bottom right quadrant (rotated 90 degrees anti-clockwise). This way, the n-th order Hilbert curve contains a grid of 2^{2n} cells. Each cell features the so-called Hilbert-value, starting in the position (0,0) and ending in the position (n,n). In practice, several libraries provide efficient implementations to obtain the Hilbert value of a given point, see (den Bercken, Dittrich, and Seeger 2000; Mehlhorn and Näher 1999) just to name a few.

Figure 2(d) depicts a set S = $\{A, B, C, D, E, F, G\}$ of points of interest. In this example, the ordering imposed by the dimension sort curve (w.r.t x-axis) is A, B, C, D, E, F, and G, and the ordering imposed by the Hilbert-curve is A, D, C, B, F, E, and G. As it can be observed in the example, the Hilbert-curve preservers locality better than the dimension sort one. For instance, the dimension sort-curve



(a) Hilbert-curve Order 1

(b) Hilbert-curve Order 2

(c) Hilbert-curve Order 3

(d) Example

Figure 2: Space-filling curves

locates B three positions away of E, nearest neighbor in the original space, while B and E are only two positions away with ordering given by the Hilbert-curve.

The pre-processing phase of the algorithm involves sorting the elements according to the Hilbert-value (resp. X or Y coordinates) and assigning equal partitions (data) to the processors. The number of facilities is also equally divided among the processors.

k-Means k-Means is one of the most popular and best understood clustering algorithms (Berkhin 2006). Unlike Algorithm 1, which associates each point to the closest facility, the k-Means algorithm associates each point to the center (or centroid) of the cluster. Broadly speaking, the algorithm works as follows:

1. Select uniformly k initial centroids;
2. Assign each node to the initial centroid;
3. Recompute the centroid of each cluster;
4. Repeat steps 2 and 3 until a given stoping criteria is met.

Typical stopping criteria are a fixed number of iterations or convergence has been reached.

The pre-processing phase of the algorithm involves the execution of the k-Means algorithm using k = number of processors. And the number of facilities for each process is proportional to the number of nodes in the partition.

3.3 Complexity of the initialisation phase

Let us now focus our attention on the complexity of the initialisation phase in the different parallel approaches. In the CLARA based approach, for each sample, we have to traverse the set of points to decide which points are in the sample and which points are not. Therefore, the complexity is linear per partition. In the space space filling curve based approaches, we have to generate the curve and then partition the problem. The generation of the curve costs nothing for the case of the dimension curves. When it comes to the Hilbert curve, computing the Hilbert value for a point is $O((log\,\eta)^2)$ where η is the number of cells (in one dimension) in the grid of points. As there are m points, the cost of generating the Hilbert values for all the points is $O(m * (log\,\eta)^2)$. Once the curve has been created, the points need to be sorted according to their curve value ($O(m * log\,m)$ and partitioned ($O(m)$). Thus, the overall complexity for the case of the Hilbert curve based approach is $O(p{*}(log\,\eta)^2{+}m{*}log\,m{+}m)$, and $O(m{*}log\,m{+}m)$ for the other cases. For the case of k-Means we pay $O(p * k * l)$ for the creation of the cluster, where l is the number of iterations.

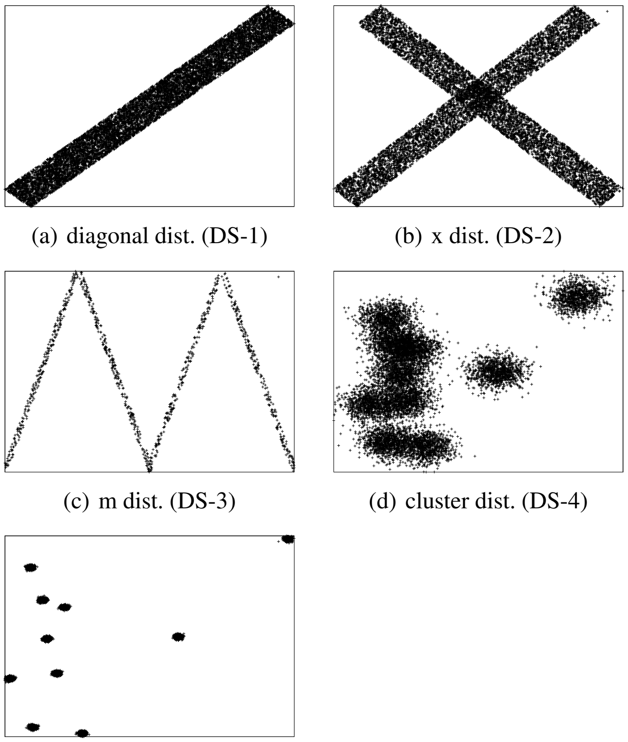
4 Experiments

4.1 Experimental setup

In this section, we present the experimental scenarios used for tests. In particular, we use the following five data sets from four different problem distributions; see (Seeger and Kriegel 1990) for a complete presentation of the problem distributions.

- *diagonal distribution* (DS1): this distribution consists of points generated, uniformly at random, in a diagonal line as shown in Figure 3(a);
- *x distribution* (DS2): this distribution consists of points generated, uniformly at random, in two diagonal lines as shown in Figure 3(b);
- *m distribution* (DS3): this distribution consists of points generated, uniformly at random, in four diagonal lines as shown in Figure 3(c);
- *cluster distribution*: this distribution consists of points generated in m clusters normally distributed around the center with a given standard deviation (std) for each cluster. We generate two set of instances: DS-4 where clusters overlap each other and DS-5 where there is a clear separation among the clusters.

In the experiments we consider a collection of 125 (5 × 5 × 5) instances. 5 instances for each problem distribution and gradually increasing the size of the problem (N1=100000, N2=120000, N3=150000, N4=200000) and an extreme case with a million points (N5=1000000). For DS-4 we use $\langle m$=10, std=1000000$\rangle$ and for DS-5 we use $\langle m$=10, std=100000$\rangle$. For all the experiments, we set the number of facilities to 1000. In addition, we executed each algorithm 5 times for each instance (using different random seeds) using a time cutoff of 300 seconds (parallel algorithms) and 1800 seconds (GEO). In order to evaluate the quality of the solutions we report the *Mean Relative Error* (MRE), which computes the distance between the solution of a given algorithm to solve a given problem instance against the best



(a) diagonal dist. (DS-1)

(b) x dist. (DS-2)

(c) m dist. (DS-3)

(d) cluster dist. (DS-4)

(e) cluster dist. (DS-5)

Figure 3: Problem distributions

known solution obtained in the experimentation of this paper.

$$MRE(a) = \frac{1}{|R|\ |\mathcal{I}|} \sum_{r \in R} \sum_{i \in \mathcal{I}} \frac{c(a,i,r) - c^*(i)}{c^*(i)}$$

where $\mathcal{I}$ represents the set of instances; R represents the set of independents runs (using different random seeds); $c(a,i,r)$ is the solution obtained by algorithm a to solve a given instance i with a random seed r; and $c^*(i)$ is the best known solution obtained to solve i. The MRE has been used before to assess the performance of local search algorithms in (Carchrae and Beck 2005; 2009)

All the experiments were performed on the Grid'5000 platform (www.grid5000.fr), the French national grid for research. In particular, we used two clusters: one with 44 nodes and another with 40 nodes. Both clusters feature 24 cores (2 AMD Opteron 6164 HE processors at 1.7 Ghz) and 44 GB of RAM per node, all nodes being interconnected on a 1 Gb network.

We build our parallel algorithm on top of GEO (Cambazard et al. 2012). We use the XXL library (den Bercken, Dittrich, and Seeger 2000) to compute the Hilbert value for a given point in a two dimensional space. We also use WEKA (Hall et al. 2009), a machine learning library that provides an efficient implementation of the k-means algorithm.

Algorithm	Cores	N1	N2	N3	N4	N5
GEO	1	371	453	578	796	–
CLARA	12	38	38	38	38	42
	24	41	42	42	42	49
CLARA'	12	212	212	213	213	207
	24	245	259	257	247	244
Hilbert	12	9	13	15	19	163
	24	9	12	13	16	114
D-Sort X	12	11	15	18	23	239
	24	10	13	15	18	162
D-Sort Y	12	10	14	16	20	208
	24	9	12	14	17	138
k-Means	12	11	14	18	25	229
	24	8	10	14	18	136

Table 2: Mean initialization time (seconds) for each algorithm

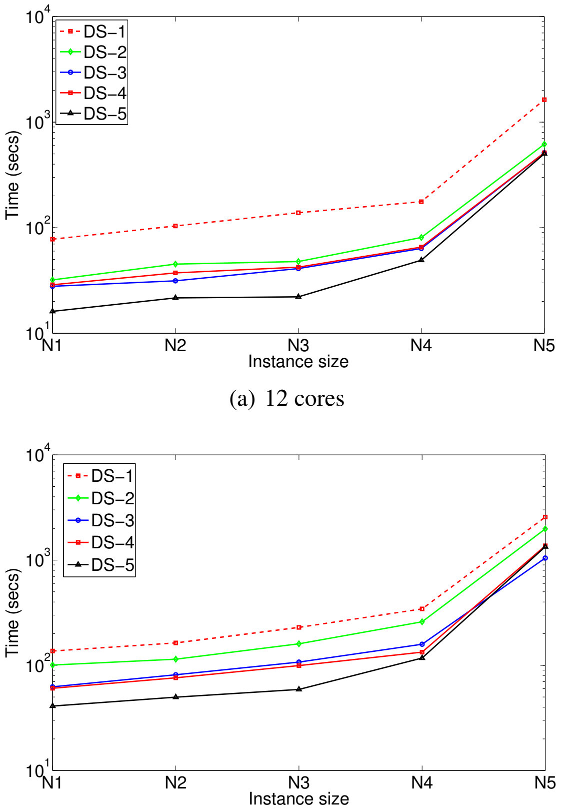
4.2 Experimental results

We start our analysis with Table 2, which depicts the initialization time for the algorithms (see Section 2). GEO represents the performance of Algorithm 1; CLARA and CLARA′ represent the performance of the CLARA-based algorithm using a sample size of 10000 and 50000; Hilbert reports the performance by partitioning the problem space using the Hilbert-curve; D-Sort X (resp. Y) reports the results w.r.t. X (resp. Y) axis coordinate; and k-Means reports the results using the k-Means algorithms to create the initial partitions. All the experiments (except GEO which uses 1 core) were performed using 12 and 24 cores. (–) is reported if a timeout was obtained in the experiments.

In this table it can be observed that the initialization time for CLARA increases with the sample size, about 5.8 times when increasing the sample size (sample size = 10000 and sample size = 50000) when using 24 cores. Furthermore, (as expected) the mean initialization time for CLARA remains nearly the same regardless the problem size, however, a small overhead is observed when doubling the number of cores (from 12 to 24). This slowdown is caused by increased cache and bus contingency when more processing cores are used at the same time. On the other hand, for the remaining parallel strategies the average initialization time for each problem distribution increases with the problem size. However, from a global perspective, partitioning the problem space reduces the initialization time. Indeed, GEO (sequential algorithm) requires more than 300 seconds to initialize the the data structures for N1, N2, N3, and N4, and for the extreme case, N5, (–) is reported since the algorithm requires more than 2 hours for initialization.

Hence, the bottleneck of the algorithms is the computation of the pre-processing phase. Figure 4 depicts the average time of executing k-Means in the k-Means based partitioning algorithm (note the log-scale) using k=12 (Figure 4(a)) and k=24 (Figure 4(b)). In general, the 12-core scenario usually requires less time. This is due to the reduced k, but as a results of this, the initialization time is larger than the 24-core scenario (see Table 2).

As it can be observed in the figure, the pre-processing



(a) 12 cores

(b) 24 cores

Figure 4: Mean pre-processing time for k-Means

time varies from instance to instance, but it is important to notice that k-Means is able to quickly identify the clusters in DS-5 up to N4. Therefore, the pre-processing time reported for this problem distribution is shorter than in the other problem distributions. However, for the extreme case (N5), the runtime goes from 1046 seconds (DS-5) to 2565 seconds (DS-1) for k=24. A similar picture is observed for k=12, where the runtime goes from 512 seconds (DS-5) to 1632 seconds (DS-1). In both cases the pre-processing time for k-Means by far exceeds the time limit for the experiments. On the other hand, the pre-processing time for the Hilbert and dimension sort methods is negligible compared to the time limit. For instance, for N4 (200000 points) the time of the Hilbert-curve based method is about 2 seconds and about 5 seconds for the extreme case (1000000 points). Moreover, we recall that the CLARA-based method does not require a pre-preprocessing phase.

Let us switch our attention to Table 3, which presents the MRE for each algorithm. First, we would like to point out that GEO, a sequential algorithm, is included in the table as a baseline for comparison and it reports the overall best solution quality for N1,N2, N3, and N4. However, it employs a larger time limit, 6 times more than the parallel algorithms. We recall that GEO requires more than 300 seconds for initialization (see Table 2).

We would like to highlight that (as expected) the performance of CLARA increases with the sample size. In general, the experimental results indicate that the k-means based algorithm outperforms the other ones in terms of solution quality. However, due to the cost of the pre-processing phase this algorithm is limited to small-size instances. Indeed, the algorithm (with 24 cores) reports a timeout for 11 out of 15 instances for sizes N3, N4 and N5. In this case, Space-filling curves are an interesting alternative as the time for the pre-processing phase is negligible. In this context, the Hilbert-curve outperforms the dimension sort-curve for problem distributions DS-4 and DS-5 (N1, N2, N3, and N4), while the dimension sort is better on the remaining three problem distributions. Interestingly, although the average pre-processing plus initialization time is about 150 seconds for DS-5 on N4 (k-Means method), the algorithm reports a timeout. This phenomenon is explained by the fact that the pre-processing phase generates unbalanced partitions; therefore the initialization time for some partitions is higher than the global timeout.

It is also worth noticing that for instances N1, N2, N3, and N4 algorithms with 12 cores outperform its counterpart with 24 cores. For these instances, the time gained for initialization is not enough to improve the quality of the solution. However, for the extreme case (N5) the computational benefit gained by increasing the number of cores is observed in both initialization time and quality of the solution within the time limit. Interestingly, in this case the best performance is obtained by using 24 cores and the Hilbert-curve for partitioning the problem space. On the other hand, CLARA' reports the best results using 12 cores, we recall that the size of the partitions for CLARA' are smaller (N5) than the ones obtained by the space-filling curve methods, and as results of this, these methods require more time per iteration than CLARA'. In addition, reducing the search space also help to reduce the time required to perform an iteration of the local search algorithm.

Finally, as pointed out in (Cambazard et al. 2012) (and confirmed in this paper) the quality of the solution tend to converge rapidly. Figure 5 (note log-scale for the x-axis) shows the evolution of the quality of the solution for the parallel algorithms, GEO is omitted because the initialization time exceeded the time limit of 300 seconds. Here, it can be observed an important reduction in the initialization time by partitioning the problem space. In addition, this figure also confirms that for small instances k-Means reports the best solution quality, followed by the Hilbert-curve, Dimension sort-curves, and finally CLARA' and CLARA.

5 Conclusions and Perspectives

This paper has studied the application of space-partitioning techniques to design a parallel local search algorithm to tackle large instances of the k-medoids clustering problem. The parallel algorithm divides the problem space into several sub-spaces and explores them in parallel until a given stopping criteria is met (e.g. timeout). Two main techniques has been deeply studied to divide the problem space: space-filling curves (Hilbert-curve and Dimension sort-curve) and

Size	Distribution	GEO	Cores	CLARA	CLARA’	Hilbert	D-Sort X	D-Sort Y	k-Means
N1	DS-1	0.000533	12	0.077973	0.018530	0.008389	0.005019	0.004730	**0.003449**
			24	0.077158	0.019579	0.011465	0.009631	0.009341	**0.005417**
	DS-2	0.000708	12	0.082622	0.022615	0.015072	0.016113	0.018062	**0.006009**
			24	0.082139	0.023419	0.023478	0.030928	0.034245	**0.010718**
	DS-3	0.000778	12	0.084987	0.020326	0.009417	0.007953	0.006752	**0.001548**
			24	0.084258	0.021419	0.018065	0.018312	0.016847	**0.005064**
	DS-4	0.000716	12	0.079125	0.019735	0.008629	0.009041	0.008709	**0.003424**
			24	0.078821	0.020234	0.012410	0.016985	0.016031	**0.005780**
	DS-5	0.000846	12	0.079079	0.019050	0.010046	0.024319	**0.006511**	0.007398
			24	0.078091	0.021145	0.014585	0.038247	0.012459	**0.010801**
N2	DS-1	0.000539	12	0.078163	0.021337	0.008760	0.005835	0.005751	**0.004010**
			24	0.077510	0.024727	0.012612	0.010403	0.010141	**0.009048**
	DS-2	0.000652	12	0.082261	0.023346	0.013144	0.014645	0.013950	**0.004839**
			24	0.081627	0.028042	0.020670	0.028223	0.026937	**0.008786**
	DS-3	0.000830	12	0.084316	0.023326	0.009883	0.006699	0.006235	**0.001462**
			24	0.083743	0.025239	0.018498	0.016191	0.015875	**0.004467**
	DS-4	0.000678	12	0.078946	0.021474	0.008842	0.008717	0.008828	**0.003677**
			24	0.078342	0.024055	0.012881	0.015785	0.015836	**0.006141**
	DS-5	0.000894	12	0.079162	0.022109	0.011417	0.024473	**0.007062**	0.008175
			24	0.078217	0.025907	0.015895	0.037437	0.012806	**0.011288**
N3	DS-1	0.000921	12	0.077415	0.024061	0.009178	0.005508	0.005795	**0.004111**
			24	0.076400	0.026014	0.012358	0.009666	**0.009522**	–
	DS-2	0.000626	12	0.082389	0.026064	0.014640	0.016290	0.017003	**0.006125**
			24	0.081726	0.029143	0.023500	0.030322	0.031805	**0.010352**
	DS-3	0.000868	12	0.083658	0.025877	0.009403	0.007450	0.006754	**0.001666**
			24	0.083017	0.028274	0.017820	0.017392	0.017467	**0.005099**
	DS-4	0.000764	12	0.078130	0.023782	0.009492	0.009188	0.008919	**0.004121**
			24	0.077801	0.026156	**0.013900**	0.016366	0.015669	–
	DS-5	0.000817	12	0.077921	0.024074	0.009912	0.022630	**0.006877**	0.007376
			24	0.076941	0.027768	0.015019	0.035378	0.012566	**0.010867**
N4	DS-1	0.000642	12	0.074875	0.025378	0.008113	**0.004956**	0.005503	–
			24	0.074229	0.026180	0.011092	**0.008453**	0.009304	–
	DS-2	0.000507	12	0.081133	0.027302	0.011004	0.011571	0.012723	**0.003094**
			24	0.080017	0.029649	0.018875	0.023846	0.025677	**0.009367**
	DS-3	0.000838	12	0.082087	0.027561	0.007994	0.006103	0.005978	**0.001900**
			24	0.081173	0.028409	0.016135	0.015186	**0.015556**	–
	DS-4	0.000651	12	0.076665	0.025771	0.007658	0.009113	0.008769	**0.003876**
			24	0.075941	0.027206	**0.011837**	0.015462	0.015048	–
	DS-5	0.000683	12	0.076202	0.025789	0.009674	0.022286	**0.006339**	0.007565
			24	0.075399	0.027356	0.013562	0.033997	**0.011764**	–
N5	DS-1	–	12	0.051067	**0.009835**	0.131000	0.139430	0.169400	–
			24	0.050642	0.010090	**0.001324**	0.028146	0.007697	–
	DS-2	–	12	0.047676	**0.003650**	0.123940	–	0.132030	–
			24	0.047236	0.004639	**0.001125**	0.027820	0.010263	–
	DS-3	–	12	0.048846	**0.004606**	0.044754	0.133960	0.077337	–
			24	0.047864	0.004727	**0.001196**	0.022438	0.010011	–
	DS-4	–	12	0.051386	**0.009259**	0.123610	0.126540	0.128760	–
			24	0.050879	0.009603	**0.001009**	0.048409	0.011728	–
	DS-5	–	12	0.049396	**0.007449**	0.125420	–	0.174040	–
			24	0.048757	0.008088	**0.001109**	0.109270	0.008052	–

Table 3: The mean relative error (MRE) for each algorithm. Each reports the performance of GEO (sequential algorithm) with a time limit of 1800 seconds (wallclock time) and the Parallel methods with a time limit of 300 seconds (wallclock time) using 12 cores (top) and 24 cores (bottom).

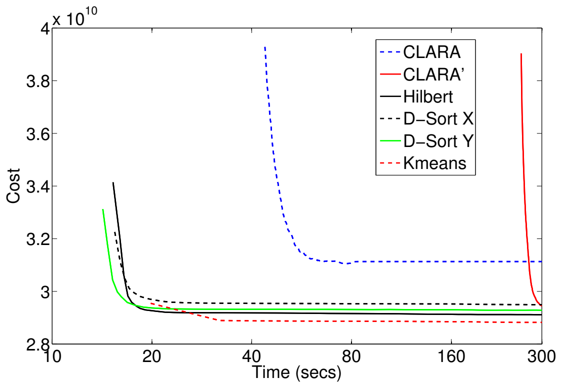


Figure 5: Evolution of the quality of the solution to solve one instance in DS-4 (size: N2)

the k-Means algorithm. Extensive experimental results suggest that the k-Means-based method usually outperforms the other ones on a subset of instances. However, for very large instances, the Hilbert-curve provides good quality solutions within a given time framework.

Currently, our approach to allocate facilities to partitions is based on the number of points within each partition. More precisely, the number of facilities allocated to a partition is directly proportional to the number of points in the partition. Certainly, the assumption here is that the partitions are homogeneous with respect to the density of points. In the future, we want to explore the option of taking the density into account when allocating the number of facilities to a partition. For instance, the number of facilities allocated to a partition could depend on the area of the minimum bounding rectangle of the partition.

We will also investigate the addition of cooperation between independent processors. To this end, we plan to divide the space into several sub-spaces with overlapping elements so that neighbour sub-spaces share facilities in order to allow the possibility of increasing or decreasing the number of facilities assigned to each sub-space.

Finally, we plan to study the application of the Hilbert-curve to reduce the complexity of opening and closing facilities in Algorithm 1. Another area of future work consists in executing the main components of the pre-processing phase in parallel, e.g. studying the impact of parallel versions of the k-Means algorithm (Kantabutra and Couch 2000). Alternatively, one could think of parallelising the basic operations in opening and closing facilities (e.g., parallelising the computation of voronoi cells).

Acknowledgements

All the parallel approaches presented in this paper were implemented on top of the GEO approach (Cambazard et al. 2012). The source code of the GEO approach was kindly provided by the developers of the approach. The first author was supported by the Japan Society for the Promotion of Science (JSPS) under the JSPS Postdoctoral Program and the *kakenhi* Grant-in-aid for Scientific Research. The second author was supported by Science Foundation Ireland Grant No. 08/CE/I523.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several universities as well as other funding bodies (see https://www.grid5000.fr).

References

Arbelaez, A., and Codognet, P. 2012. Massivelly Parallel Local Search for SAT. In *ICTAI'12*, 57–64. Athens, Greece: IEEE Computer Society.

Berkhin, P. 2006. A Survey of Clustering Data Mining Techniques. In Kogan, J.; Nicholas, C.; and Teboulle, M., eds., *Grouping Multidimensional Data*. Springer. 25–71.

Butz, A. 1971. Alternative Algorithm for Hilbert's Space-Filling Curve. *Computers, IEEE Transactions on* C-20(4):424–426.

Cambazard, H.; Mehta, D.; O'Sullivan, B.; and Quesada, L. 2012. A Computational Geometry-Based Local Search Algorithm for Planar Location Problems. In Beldiceanu, N.; Jussien, N.; and Pinson, E., eds., *CPAIOR*, 97–112. Nantes, France: LNCS.

Caniou, Y.; Codognet, P.; Diaz, D.; and Abreu, S. 2011. Experiments in parallel constraint-based local search. In Merz, P., and Hao, J.-K., eds., *11th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'11)*, volume 6622 of *LNCS*, 96–107. Torino, Italy: Springer.

Carchrae, T., and Beck, J. C. 2005. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

Carchrae, T., and Beck, J. C. 2009. Principles for the Design of Large Neighborhood Search. *J. Math. Model. Algorithms* 8(3):245–270.

Crainic, T., and Toulouse, M. 2003. Parallel strategies for meta-heuristics. In Glover, F., and Kochenberger, G., eds., *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Springer US. 475–513.

den Bercken, J. V.; Dittrich, J.-P.; and Seeger, B. 2000. javax.XXL: A Prototype for a Library of Query Processing Algorithms. In Chen, W.; Naughton, J. F.; and Bernstein, P. A., eds., *SIGMOD*, 588. Dallas, Texas, USA: ACM.

Diaz, D.; Richoux, F.; Caniou, Y.; Codognet, P.; and Abreu, S. 2012. Parallel Local Search for the Costas Array Problem. In *IPDPS Workshops*, 1793–1802.

Faloutsos, C. 1988. Gray Codes for Partial Match and Range Queries. *IEEE Trans. Software Eng.* 14(10):1381–1393.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 11(1):10–18.

Hassani Bijarbooneh, F.; Flener, P.; Ngai, E.; and Pearson, J. 2011. Energy-efficient task mapping for data-driven sensor network macroprogramming using constraint programming. In *Operations Research, Computing, and Homeland*

Defense, 199–209. Institute for Operations Research and the Management Sciences.

Kantabutra, S., and Couch, A. L. 2000. Parallel K-means Clustering Algorithm on NOWs. *NECTEC Technical journal* 1(6):243–247.

Kaufman, L., and Rousseeuw, P. J. 1990. *Finding groups in data: an introduction to cluster analysis*. New York: John Wiley and Sons.

Megiddo, N., and Supowit, K. 1984. On the complexity of some common geometric location problems. *SIAM journal on computing* 13(1):182–196.

Mehlhorn, K., and Näher, S. 1999. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press.

Michel, L., and Van Hentenryck, P. 2004. A simple tabu search for warehouse location. *European Journal of Operational Research* 157(3):576–591.

Mokbel, M. F.; Aref, W. G.; and Kamel, I. 2002. Performance of Multi-Dimensional Space-Filling Curves. In Voisard, A., and Chen, S.-C., eds., *ACM-GIS*, 149–154.

Mouratidis, K.; Papadias, D.; and Papadimitriou, S. 2008. Tree-based partition querying: a methodology for computing medoids in large spatial datasets. *VLDB J.* 17(4):923–945.

Papadimitriou, C. 1981. Worst-case and probabilistic analysis of a geometric location problem. *SIAM Journal on Computing* 10(3):542–557.

Rosenkrantz, D. J., and Fagin, R., eds. 1984. *A Class of Data Structures for Associative Searching*. Ontario, Canada: ACM.

Sagan, H. 1994. *Space-filling curves*. Universitext Series. Springer-Verlag.

Seeger, B., and Kriegel, H.-P. 1990. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In McLeod, D.; Sacks-Davis, R.; and Schek, H.-J., eds., *VLDB*, 590–601. Brisbane, Queensland, Australia: Morgan Kaufmann.

Wang, X.; Xing, G.; Zhang, Y.; Lu, C.; Pless, R.; and Gill, C. 2003. Integrated coverage and connectivity configuration in wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, 28–39. New York, NY, USA: ACM.

Zhang, Y.-P.; Sun, J.-Z.; Zhang, Y.; and Zhang, X. 2004. Parallel Implementation of CLARANS using PVM. In *Machine Learning and Cybernetics*, volume 3, 1646–1649. IEEE.