Partition-Based Clustering in Object Bases: From Theory to Practice

Carsten Gerlhof[1], Alfons Kemper[1], Christoph Kilger[2], Guido Moerkotte[2]

[1] Universität Passau, Lehrstuhl für Dialogorientierte Systeme, Fakultät für Mathematik und Informatik, D-94030 Passau, Germany, [*gerlhof* | *kemper*]@db.fmi.uni-passau.de

[2] Universität Karlsruhe, Fakultät für Informatik, D-76050 Karlsruhe, Germany, [*kilger* | *moer*]@ira.uka.de

Abstract. We classify clustering algorithms into *sequence-based* techniques—which transform the object net into a linear sequence—and *partition-based* clustering algorithms. Tsangaris and Naughton [TN91, TN92] have shown that the partition-based techniques are superior. However, their work is based on a single partitioning algorithm, the Kernighan and Lin heuristics, which is not applicable to realistically large object bases because of its high running-time complexity. The contribution of this paper is two-fold: (1) we devise a new class of greedy object graph partitioning algorithms (GGP) whose running-time complexity is moderate while still yielding good quality results. (2) Our extensive quantitative analysis of all well-known partitioning algorithms indicates that no *one* algorithm performs superior for *all* object net characteristics. Therefore, we propose an *adaptable* clustering strategy according to a multi-dimensional grid: the dimensions correspond to particular characteristics of the object base—given by, e.g., number and size of objects, degree of object sharing—and the grid entries indicate the most suitable clustering algorithm for the particular configuration.

1 Introduction

Clustering of logically related objects on the same page is a very powerful optimization concept. Unlike other, more restricted models—e.g., the hierarchical model or the nested relational model—the object-oriented data models allow arbitrary object graphs. We distinguish between *sequence-based* and *partition-based* clustering. Under sequence-based clustering the object graph is transformed into a linear sequence of objects which is then sequentially assigned to pages. Under partition-based clustering the object graph is partitioned into object partitions that fit onto a single page. Tsangaris and Naughton [TN91, TN92] showed that the partition-based clustering is superior to sequence-based clustering. Unfortunately, the well-known partitioning algorithms—such as the Kernighan and Lin (KL) algorithm, which was the only one investigated in [TN91, TN92]—have a very high running-time complexity. This makes their application to realistically large object bases impossible.

Therefore, in this work we develop a new class of *greedy graph partitioning (GGP)* heuristics which have a moderate running-time complexity while still yielding good quality results. From the basic GGP heuristics we develop more sophisticated heuristics by incorporating a *look-ahead* with the possibility of rejecting less promising choices and a *new-chance* for those rejected choices at a later stage. Our extensive quantitative analysis—of which only a fraction could be covered in the paper—indicates that the quality of the (sophisticated) greedy partitioning is slightly inferior to the computationally complex algorithms, such as KL. However, the greedy heuristics are indispensable for two reasons:

1. They are extremely useful for pre-partitioning the cluster graph. These pre-partitions can then be improved using the computationally complex algorithms, e.g., KL.
2. For (realistically) large object bases the computationally complex algorithms cannot be employed because of their enormous running-time. In this case we can employ our greedy heuristic which is far better than the best known sequence-based algorithms—the *best-first* heuristics.

The above discussion leads to the conclusion that there is no *one* single cluster heuristics which is superior for *all* object base configurations. This motivates us to propose an *adaptable* cluster strategy according to a multi-dimensional grid: the dimensions correspond to particular characteristics of the object base—given by, e.g., number and size of objects, degree of object sharing—and the grid entries indicate the most suitable clustering algorithm for the particular configuration.

The rest of the paper is organized as follows. In Section 2 the cluster problem is defined and the related work is classified. Then, in Section 3 the well-known partitioning algorithms are reviewed and the basics of our greedy graph partitioning heuristics is introduced. In Section 4 two enhancements of the basic GGP heuristics are presented. Section 5 contains some of our extensive quantitative analysis of many different partition-based cluster heuristics. In Section 6 we derive the adaptable cluster strategy controlled by the multi-dimensional grid. Section 7 concludes the paper.

2 Clustering as a Graph Partitioning Problem

The clustering problem is closely related to the graph partitioning problem where some graph is to be partitioned into several disconnected subgraphs (partitions). The *object graph* (OG) is constructed considering objects as vertices and the inter-object references as directed edges [TN91]. Clustering algorithms partition the OG by assigning objects to equally sized pages. Instead of the OG clustering algorithms often use a more specific graph as input that is derived from the OG and/or from information about the applications' access behavior, e.g., access traces. We follow [TN91] and call this graph the *clustering graph* (CG). The vertices and edges of the CG are labeled with weights: vertex weights represent object sizes and edge weights represent the application's access behavior (higher weights denote that the start and terminal object of the edge are more often accessed in succession). For a given partitioning of the CG, the total weight of all edges crossing partition borders (i.e., page borders) are the *external costs* of this partitioning. The clustering problem is to find a partitioning of the CG such that the size of each partition, i.e., the total size of its objects, is less or equal the page size and the external costs are minimized.

There are two dimensions along which clustering algorithms can be classified: (1) the determination of the access patterns, i.e., edge weights, and (2) the algorithm that is applied to map objects into pages. Along the first dimension—the determination of the access patterns—*static* and *dynamic methods* can be distinguished. Along the second dimension we distinguish *sequence-based* from *partition-based* mapping algorithms.

2.1 Determination of the Access Patterns

Static methods are based on analyzing either the structure of the object base—e.g. static reference counts of objects[3] [Sta84]—or the access behavior of operations (that are part of the schema) [GKKM92a], or they require input from the database programmer, e.g., [Ben90, CDRS86]. *Dynamic methods* analyze the access trace of former database applications by monitoring the system [Sta84, HK89]. Usually, it is far more expensive to gather dynamic information on the access behavior of applications since this requires monitoring the applications. For a thorough discussion of the pros and cons of static versus dynamic access information see [GKKM92a]. In the remainder of this paper we assume the weights of the edges of the CG to be given—either by dynamic or static analysis.

2.2 Sequence-Based Mapping Algorithms

Sequence-based algorithms partition the CG by applying a graph traversal algorithm producing a linear sequence of all objects from the CG. This sequence of objects is then segmented—from left to right—into partitions. Sequence-based algorithms can be denoted in the style of a UNIX pipe consisting of the two steps *PreSort* and *Traversal*:

$$PreSort \mid Traversal$$

The *Traversal* component starts with some object and traverses all objects that are reachable from the start object and have not been visited before. After the current traversal is finished a new (unvisited) object is selected to start the next traversal. The *PreSort* method is used for sorting all objects, e.g. ordering by type [HZ87, Sta84, Ben90], ordering by decreasing dynamic reference counts[4] [HK89], and using some arbitrary ordering [Sta84, BKKG88]. The simplest *Traversal* algorithm maps objects into pages according to their position in the presort order [Sta84, HZ87]. Depth-first and breadth-first traversal algorithms for graphs were applied in [Sta84, BKKG88]. In Cactis [HK89] the

[3] The *static reference count* (SRC) of an object equals the number of references pointing to that object.

[4] The *dynamic reference count* (DRC) of an object equals the number of times the object is referenced by a sampling application.

usage of a best-first (BSTF) traversal algorithm is proposed. BSTF was the best *Traversal* algorithm in many comparison studys of clustering algorithms, e.g. [TN91, GKKM92a]. Therefore, in the remainder of this paper, we investigate only BSTF as the representative for sequence-based clustering.

2.3 Partition-Based Mapping Algorithms

Partition-based clustering algorithms segment the CG utilizing graph partitioning algorithms. In the literature graph partitioning algorithms for clustering were only considered in the work of Tsangaris and Naughton [TN91, TN92], where the algorithm by Kernighan and Lin (KL) was applied [KL70]. The clustering results obtained by the KL algorithm have been far superior to those obtained by sequence-based algorithms. Although the work of Tsangaris and Naughton lead the way to partition-based clustering their results were based on benchmarks run on a rather small object base where all objects were of uniform size. In practice there is a large diversity of object nets whose characteristics are essential for the performance and the cluster quality of partitioning algorithms: (1) The size of the object net which may range from a few hundred to hundreds of thousands of objects. (2) The size of the objects which may strongly vary. (3) The degree of sharing: there are tree-like object nets and others which exhibit a highly connected network strucure. It turns out that no one algorithm can be identified as superior in all (or even most of the) cases. Therefore, we investigate a wide range of partitioning algorithms for the clustering problem.

3 The Graph Partitioning Algorithms

According to [SM91] graph partitioning algorithms can be divided into two classes: *constructive partitioning* and *iterative improvement*. Constructive algorithms build a partitioning of a graph from the scratch; iterative algorithms start with some initial partitioning and repeatedly try to improve this partitioning. Iterative algorithms usually produce better results than constructive algorithms, but have a very large running-time compared to constructive algorithms. In the remainder of this section we sketch four graph partitioning algorithms known from the literature. Further, we describe the basics of our new greedy heuristics for graph partitioning—more sophisticated heuristics are developed in Section 4.

3.1 Iterative Algorithms

The Kernighan-Lin Heuristics. The Kernighan-Lin algorithm (KL) [KL70] was designed for the placement of VLSI chips—that is why KL is not well suited for clustering in object bases. The KL algorithm starts with an arbitrary partitioning of the CG and iterates over all pairs of partitions, trying to improve the partitioning by exchanging objects between the pair of partitions currently considered. Thus, the object sizes must not be strongly diverging for allowing objects to be swapped. While this may be true for VLSI applications, it is in general not true for object bases.

The Fiduccia-Mattheyses Heuristics. The graph partitioning algorithm by Fiduccia and Mattheyses (FM) [FM82] is derived from the KL algorithm. The main modification is that objects are *moved* across partition borders instead of being *swapped*. This modification allows for handling unbalanced partitions and non-uniform object sizes. On the other hand, swapping as employed in the KL heuristics, is advantageous in case of uniform size and nearly full pages—in this case no objects can be moved by the FM heuristics. Further, the FM heuristics is able to adapt the number of objects of the partitions, and even the number of partitions (if all objects are removed from some partition, the partition can be deleted). KL remains always stuck with the number of objects per partition and the number of partitions that was initially chosen.

Hierarchical Partitioning. The hierarchical partitioning algorithm (HP) is a combination of the KL and FM algorithms. The HP algorithm works as follows: All objects are assigned to one partition whose size is $P = 2^x * \mathit{PageSize}$ where x is chosen to be the smallest integer such that the total size of all objects is less than or equal to P. This partition is split such that the objects are distributed equally (by size) among both partitions. The resulting partitions are improved using the KL or the FM heuristics. This method is recursively applied until the size of every partition is below the page size. The idea that lead to the HP algorithm was taken from the work of Breuer [Bre77].

3.2 Constructive Algorithms

Optimum Partitioning of Trees. Lukes [Luk74] presents a pseudopolynomial-time algorithm for computing an optimum partitioning for the vertices of a tree, based on dynamic programming. The complexity of the algorithm is $O(n * B_{\max}^2)$ where n is the number of vertices of the tree and $B_{\max}$ is the maximum number of objects per page.

The Greedy Graph Partitioning Heuristics. Because of the very good clustering results but poor running-time performance of known partition-based clustering algorithms, we have developed a new heuristics for graph partitioning, called *Greedy Graph Partitioning* (*GGP*). The GGP algorithm was first proposed in [GKKM92a]. Graph partitioning is strongly related to subset optimization problems for which greedy algorithms often find good solutions very efficiently. The GGP algorithm is based on a simple greedy heuristics that was derived from Kruskal's algorithm for computing the minimum weight spanning tree of a graph [Kru56]. First, all partitions are inhabited by a single object only, and all partitions are inserted into the list *PartList*. For all objects o_1, o_2 connected by some edge in the CG with weight w_{o_1,o_2} a tuple (o_1, o_2, w_{o_1,o_2}) is inserted into the list *EdgeList*. All tuples of *EdgeList* are visited in the order of descending weights. Let (o_1, o_2, w_{o_1,o_2}) be the current tuple. Let P_1, P_2 be the partitions to which the objects o_1 and o_2 are assigned. If $P_1 \neq P_2$ and if the total size of all objects assigned to P_1 and P_2 is less than the page size the two partitions are joined.[5] Otherwise, the edge is merely discarded—and the partitions remain invariant. If e denotes the number of edges of the CG the running-time complexity of the GGP algorithm is $O(e \log e)$—the dominating factor is here the sorting of the list *EdgeList*.

4 New Heuristics for Greedy Graph Partitioning

In [GKKM92a] we have extensively benchmarked the GGP heuristics on various forms of CG's comparing it to several clustering strategies for object-oriented database systems. The performance of GGP indicated the best cost effectiveness in terms of running-time and clustering quality. In this section we present two heuristics that further improve the results of GGP:

1. *bounded look-ahead:* Before the partitions incident to the current edge, i.e., the edge with maximum weight, are being joined, a BSTF traversal is performed locally in order to look for better candidate partitions to be joined.
2. *new-chance:* If the bounded look-ahead heuristics finds potentially better candidate partitions the current edge is rejected. Rejected edges need be re-considered (triggered) if the condition that lead to the rejection changes.

We identify the enhanced GGP algorithm by GGPla (*GGP with look-ahead*). Subsequently, both improvements are discussed in more detail. The complete description of the algorithm GGPla can be found in [GKKM92b].

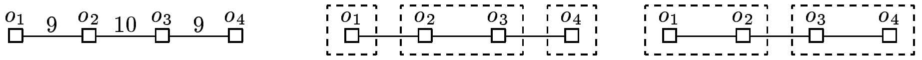
4.1 k-Look-Ahead

During each iteration the GGP algorithm takes the edge with maximum weight from the *EdgeList* and tries to join the partitions of the objects incident to that edge. Of course, this is not necessarily the best decision. Consider the example shown in Fig. 1. The CG is visualized in Fig. 1 (a); the maximum external costs are $EC_{\max} = 28$.[6] We assume uniform object sizes and a page capacity of 2 objects. Without look-ahead GGP decides to assign the objects o_2 and o_3 to the same partition resulting in a final partitioning with external costs of 18, visualized in Fig. 1 (b). Obviously, the optimum partitioning of this CG has only external costs of 10 (Fig. 1 (c)). The idea of the bounded look-ahead is to detect situations where it is advantageous to *reject* the current edge, i.e., the edge with maximum weight, and to consider other edges first. Subsequently, this rough outline of bounded look-ahead is detailed.

Consider the CG depicted in Fig. 2. The objects o_l and o_r have been assigned to the partitions P_l and P_r. Besides the edge (o_l, o_r) there are n external edges incident to o_l with weights $w_{l_1}, \ldots, w_{l_n}$ and m external edges incident to o_r with weights $w_{r_1}, \ldots, w_{r_m}$. Assume the next edge to be considered by GGP

[5] Partitions are represented as binary trees to accelerate the join operation.

[6] Squares denote objects, dashed boxes denote partitions.



(a) $EC_{\max} = 28$ (b) $EC = 18$ (c) $EC = 10$

Fig. 1. An Example for Non-Optimal GGP Clustering

is (o_l, o_r) with weight w, which implies $w \geq \max\{w_{l_1}, \ldots, w_{l_n}, w_{r_1}, \ldots, w_{r_m}\}$. Further assume that the partitions P_l and P_r could be joined because their total size being less than or equal to the page size.

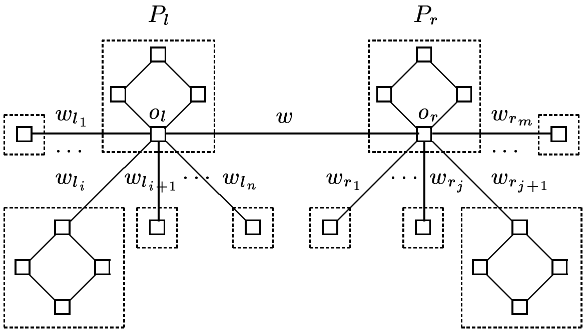


Fig. 2. Clustering Graph Before Analyzing the Edge (o_l, o_r)

To decide whether it is beneficial to join the partitions P_l and P_r (or not) we proceed in two steps:

1. We evaluate the potential gain in the case P_l and P_r are being joined and the resulting partition is being filled up to the maximum size. For that, a BSTF traversal is performed on the external edges of the CG starting at the objects $\{o_l, o_r\}$.[7] The traversal is terminated if either (1) the depth of the traversal exceeds some integer bound k, or (2) the total size of all partitions encountered (including P_l and P_r) exceeds the page size. Let $w_{l_1}, \ldots, w_{l_i}, w_{r_1}, \ldots, w_{r_j}$ be the weights of all edges accessed during the BSTF traversal. Then, $W_{\text{join}} = w + w_{l_1} + \ldots + w_{l_i} + w_{r_1} + \ldots + w_{r_j}$ denotes the potential gain in the case that the partitions P_l and P_r are being joined and the resulting partition were filled up completely using a BSTF traversal starting at the objects $\{o_l, o_r\}$.
2. We evaluate the effect of not joining P_l and P_r. To do so, we perform two BSTF traversals (on the external edges of the CG) starting at o_l (*left traversal*) and o_r (*right traversal*). The left (right) traversal is terminated as soon as the depth of the traversal exceeds k or when the total size of all partitions

[7] Note that the BSTF search inspects only paths that emanate from o_l and o_r—instead of inspecting all objects already in partitions P_l and P_r. Our experiments showed that the running-time becomes prohibitively large, otherwise.

encountered including P_l (P_r) exceeds the page size. Let W_{left} denote the total weight of all edges traversed by the left traversal, and let $\mathcal{P}_{\text{left}}$ denote the set of partitions encountered during the left traversal (including P_l). W_{left} denotes the gain that can potentially be achieved by GGP on the left part of the CG if the partitions P_l and P_r are *not* joined in the current stage of GGP. W_{right} and $\mathcal{P}_{\text{right}}$ are defined analogously for the right traversal.

Based on the definitions made above we can define two heuristics H1 and H2:

Heuristics H1: The edge (o_l, o_r) is **rejected**, i.e., P_l and P_r are not joined in the current stage of GGP, if either
- $W_{\text{left}} > W_{\text{join}}$ and the total size of all objects on pages in $\mathcal{P}_{\text{left}} \cup \{P_r\}$ exceeds the page size, or
- $W_{\text{right}} > W_{\text{join}}$ and the total size of all objects on pages in $\mathcal{P}_{\text{right}} \cup \{P_l\}$ exceeds the page size.

Heuristics H1 rejects the edge (o_l, o_r) if either the gain on the left part of the CG (rooted at o_l) or on the right part of the CG (rooted at o_r) is larger than the gain achieved by joining the partitions P_l and P_r.
Note, however, that by this heuristics the optimal partitioning of the CG shown in Fig. 1 is not found. This observation leads to the definition of

Heuristics H2: The edge (o_l, o_r) is **rejected**, i.e., P_l and P_r are not joined in the current iteration of GGP, if
- $W_{\text{left}} + W_{\text{right}} > W_{\text{join}}$ and the total size of all objects on pages in $\mathcal{P}_{\text{left}} \cup \mathcal{P}_{\text{right}}$ exceeds the page size.

Under heuristics H2, the total gain that can be potentially achieved by not joining the partitions P_l and P_r is compared to the gain potentially achieved by joining P_l and P_r. It is easy to see that the condition of H2 is always true if the condition of H1 is fulfilled—thus, H2 is a strengthening of H1. A similar heuristics that introduces more look-ahead into the Fiduccia-Mattheyses algorithm was proposed by Krishnamurthy [Kri84].

4.2 New-Chance

In this section we discuss the "fate" of edges that were rejected by the k-look-ahead heuristics. Rejected edges must not be removed from the *EdgeList* because they should be re-considered as soon as the condition leading to the rejection *changes*. Technically, in our implementation, edges are given a new chance by moving them to the end of the *EdgeList* when they are rejected.

The set of the external edges which has been visited in the k-look-ahead for some rejected edge e_r is called the *trigger-set* of e_r. Now let us assume that some edge e is considered and is not being rejected during the current iteration of GGP. Further assume that e is in the trigger-set of the rejected edge e_r. In this case the edge e_r is said to be *triggered*. After the join of the partitions incident to e the local situation in the CG that was responsible for rejecting the edge e_r has changed. In this new situation the edge e_r would possibly survive (i.e., not being rejected). Thus, we propose to retry any rejected edge when it is triggered.[8] Triggered edges are moved to the first position of the *EdgeList* as their weight is greater than or equal to the maximum of the weights of all edges that have not yet been considered.

5 Quantitative Analysis

In this section we investigate the quality of clustering algorithms (1) in terms of quality, i.e., external costs and (2) in terms of running-time. We decided to use the external costs of the partitioning to evaluate the quality of the clustering because this measure does not depend on the choice of the benchmark applications, the buffer size, the replacement policy, nor the working set window size. The experiments were carried out using a graph generator—that is part of our simulation workbench TEXAS.[9] The graphs used in the experiments consist of 10 *modules*, each constituting a highly connected subgraph. Every module is connected to one randomly selected module by an edge with minimal weight 1. The weights of the intra-module edges are uniformly distributed in the range $2, \ldots, 20$. We classify the characteristics of the examined clustering graphs according to five dimensions:

database cardinality	average number of objects per page	deviation of object sizes	degree of sharing	static reference count (SRC)

The cardinality of the database is the dominant factor of the running-time of the algorithms, whereas the remaining four parameters influence the quality of the partitionings generated by the algorithms. The degree of sharing is stated in % and determines the ratio of the number of shared objects to the total number of objects. We conducted experiments on many combinations of the five parameters. However, because of space restrictions in this paper, we discuss only the influence of the first four dimensions (omitting the SRC of shared objects) on the choice of the cluster algorithm—see [GKKM92b] for a more complete treatment. The following algorithms were measured: BSTF, GGP, GGPla, HP, Lukes and KL(x), FM(x) each with pre-partitioning produced by x, for $x \in \{$ GGPla, BSTF $\}$. The (iterative) algorithms KL(x) and FM(x) were run until no further improvement was achieved (or *maxint* was reached). To compensate for one major disadvantage of Lukes' algorithm (producing memory overflow when working on high trees) we limited the keeping of the best assignments of objects to partitions only up to the average number of objects per page.

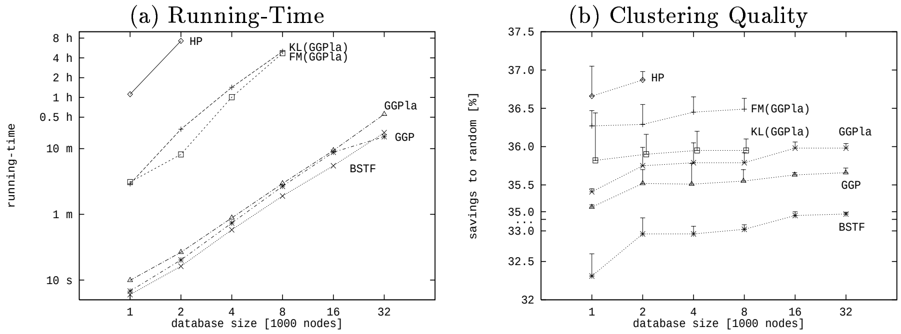
5.1 Running-Time vs. Quality of Cluster Algorithms

The first experiment evaluates the cluster algorithms BSTF, GGP, GGPla, FM(GGPla), KL(GGPla), and HP in terms of (1) their partitioning quality—

[8] To avoid exhaustive searching we employ special data structures facilitating the direct access from the current edge to the set of triggered edges.

[9] The EXtensible Access Simulator for object bases.

that is, we compute the relative savings in external costs in relation to a random placement—and (2) their running-time dependent on the database cardinality. The algorithms worked on a scalable database of $n \leq 32$ modules—each constituting a random graph with 1000 objects and 4 edges per object on the average. Fig. 3 visualizes the results where the database size is plotted against the x-axis. In Fig. 3(a) the running-time is plotted against the y-axis, in Fig. 3(b) the relative savings of the obtained clustering to a random placement is plotted against the y-axis. The vertical (error) bars in the plots of Fig. 3 (b) visualize the standard deviation of the clustering results for the different graphs.



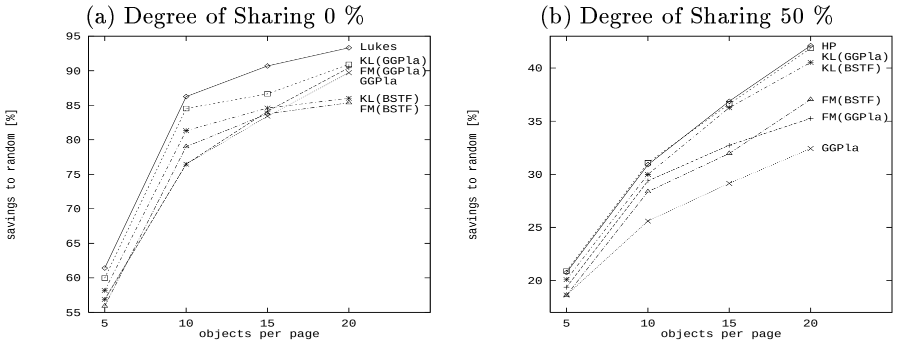
database cardinality	average number of objects per page	deviation of object sizes	degree of sharing	static reference count (SRC)
variable	15	300	100%	4

Fig. 3. Performance of Clustering Algorithms Dependent on the Database Size

The most important result of this experiment is that the running-times of the algorithms are drastically different. Whereas the constructive algorithms GGPla and GGP, and the sequenced-based technique BSTF are very fast (30 minutes for 32000 objects), the running-time of the iterative algorithms KL(GGPla), FM(GGPla), and HP is not acceptable for large databases. We limited the maximal running-time to 24 hours; Thus, HP is only feasible for small databases up to 2000 objects and KL/FM may be feasible for databases up to 8000 objects—if the long reorganization time is tolerable at all. Note, that even *small* differences in relative savings (the range is from 32% to 37%) exhibit enormous differences in absolute external costs—e.g., 40273 between BSTF and GGPla at a database cardinality of 32000 objects. The iterative algorithms HP, KL(GGPla), and FM(GGPla) exhibited the best partitioning results. Nevertheless, the results of the KL and FM heuristics strongly depend on the quality of the pre-partitioning. Utilizing a poor-quality pre-partitioning KL and FM cannot achieve such good results. Among the algorithms whose running-time is reasonably low GGP and GGPla performed best. The relative ordering of the algorithms—in running-time as well as in clustering quality—that was observed in this experiment remains stable in the majority of the experiments we conducted.

5.2 Changing the Partition Size

The number of objects per page has an important influence on the clustering results. Therefore, the page size is an effective tuning parameter in physical database design. Note, that varying the average object size instead of the page size would have the analogous effect. In the subsequent experiment we measure the performance of the clustering algorithms Lukes, HP, KL, FM, and GGPla under varying page sizes. Fig. 4 (a) visualizes the results for a tree-structured graph (sharing = 0) and Fig. 4 (b) visualizes the result for a graph with 50 % shared objects and an SRC of 11 for shared objects. All objects were of equal size. Thus, in this case our approximation of Lukes' algorithm is optimal.



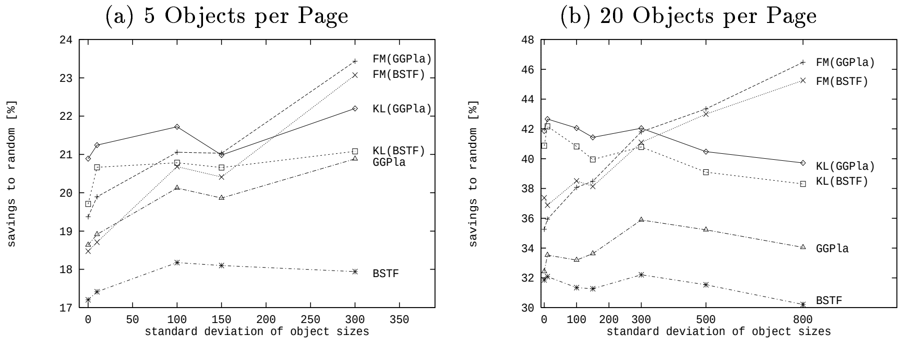
	database cardinality	average number of objects per page	deviation of object sizes	degree of sharing	static reference count (SRC)
(a)	1000	variable	0	0%	0
(b)	1000	variable	0	50%	11

Fig. 4. Influence of the Page Size on the Clustering Results

Lukes' algorithm achieves 93 % savings compared to random placement. The plot of the results of KL(GGPla) runs very close to this optimum—indicating that KL with the high quality pre-partitioning of GGPla is a good candidate for clustering trees of moderate size. The clustering results in Fig. 4 (b) range from 20 % to 40 % savings to random. Thus, if the degree of sharing is high and the page size is small, only small improvements can be achieved by clustering. With an increasing page size, the potential improvement by clustering increases, too.

5.3 Varying Object Sizes

Usually, the objects' sizes vary significantly—note, that the object net is composed of objects of many different types. E.g., there may be tuple-structured objects of just 50 bytes, whereas the size of set- or list-structured objects easily may even reach the page size. Fig. 5 (a) visualizes the results for a page capacity of 5 objects, and Fig. 5 (b) visualizes the results for a page size of 20 objects, both with 50 % sharing and SRC=11.



	database cardinality	average number of objects per page	deviation of object sizes	degree of sharing	static reference count (SRC)
(a)	1000	5	variable	50%	11
(b)	1000	20	variable	50%	11

Fig. 5. Performance of Clustering Algorithms Under Varying Object Sizes

The main observation is that the relative saving increase upon increasing the deviation of object sizes. The second result of this experiment is that FM performs better than KL if the object sizes vary substantially. Below a page capacity of 20 objects the break even point between KL and FM is at a standard deviation of object sizes of about 300 (Fig. 5 (b)). If the page capacity is only 5 objects per page on the average, the break even point is moved to a standard deviation of 150 (Fig. 5 (a)). To understand this result, consider Fig. 6. Neither KL nor FM can improve the partitioning of the graph depicted in Fig. 6 (a), where strong internal edges prevent from moving or swapping objects between the two partitions. If the number of objects decreases the probability of finding clustering subgraphs similar to those in Fig. 6 (b) and (c) increases. FM can improve the partitioning by moving a single object while KL cannot improve the partitioning as there are no pairs of objects to be swapped.

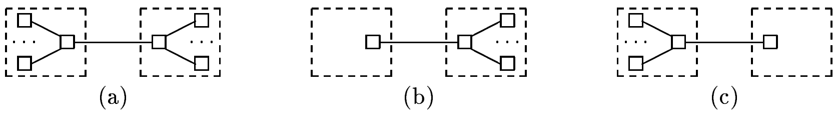
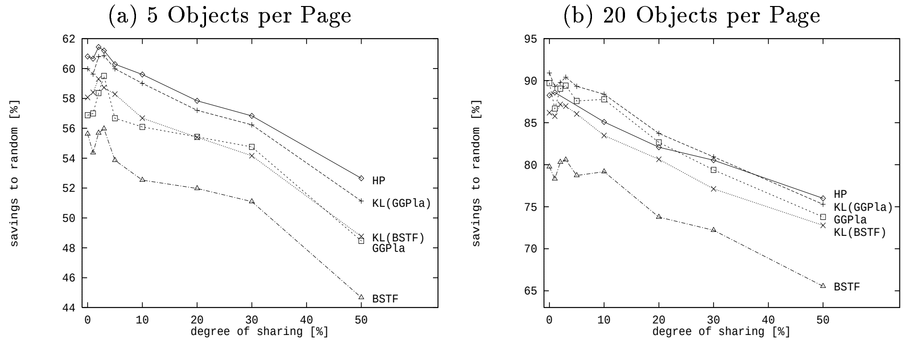


Fig. 6. Three Partitioning Problems

5.4 Degree of Sharing

The experiment in Section 5.2 already indicated that object sharing makes clustering more difficult. The next experiment is designed to investigate the influence of the degree of sharing on the clustering results. We measure the algorithms HP, KL(GGPla), KL(BSTF), GGPla, and BSTF on a graph with uniform object sizes and an SRC of 2 for shared objects. Fig. 7 (a) visualizes the results for a page capacity of 5, Fig. 7 (b) for a page capacity of 20 objects per page on the average. The degree of sharing varies from 0 % to 50 % (plotted against the x-axis).



	database cardinality	average number of objects per page	deviation of object sizes	degree of sharing	static reference count (SRC)
(a)	1000	5	0	variable	2
(b)	1000	20	0	variable	2

Fig. 7. Performance of Clustering Algorithms Under Varying Degrees of Sharing

As expected, the plots of all algorithms show a steep decline as the degree of sharing increases. With 5 objects per page on the average the savings relative to random placement drop by about 10 %, with 20 objects per page even by 15 %. In both experiments, HP and KL(GGPla) yield the best clustering results among the algorithms tested.

6 The Case for Adaptable Algorithms

The results of the quantitative analysis presented in the previous section lead to the conclusion that no *one* algorithm performs superior for *all* object base configurations. Therefore, we derive a multi-dimensional grid: the dimensions correspond to particular characteristics of the object base configurations and the grid entries determine the best clustering algorithm for the particular configuration.

The grid is shown in Fig. 8. Three dimensions are visualized by the axes of the grid, i.e., the average number of objects per page, the standard deviation of the object sizes, and the degree of sharing. The fourth dimension, i.e., the size of the database in number of objects, is visualized by the coloring of the grid entries shown on the right hand side. Here, we distinguish three configurations: (1) a

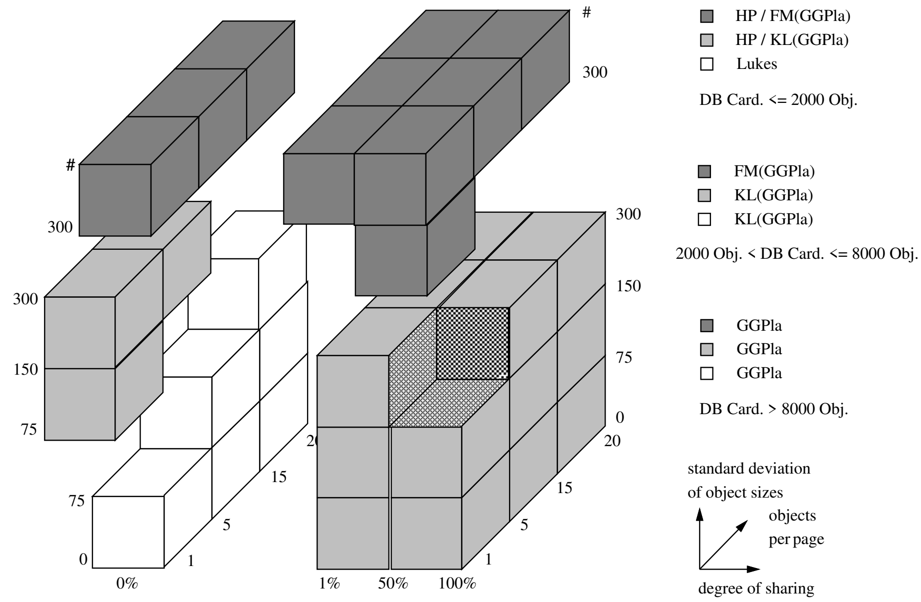


Fig. 8. Selecting the Best Clustering Algorithm

database with less than 2000 objects, (2) a database with more than 2000 and less than 8000 objects, and (3) a database with more than 8000 objects. For each configuration, the coloring (shading) of the grid indicates the most appropriate clustering algorithm. Subsequently, we briefly discuss the entries of the grid.

Let us start with the left hand side of the grid—representing a sharing of 0 %. In this case, the approximation of Lukes' algorithm is optimal if all objects are of equal size. Our experiments further indicate that the approximation of Lukes' algorithm yields good results as long as the ratio of the standard deviation of the object size to the average number of objects per page is small (white colored area). When this ratio increases other algorithms perform better than the approximation of Lukes, e.g., HP. We have found that beyond a database cardinality of 2000 objects Lukes' algorithm ran out of memory. That is why we propose to use KL/FM(GGPla) for medium cardinality and GGPla for large cardinality object bases even if the CG forms a tree. Next we investigate the right hand side of the grid, where the degree of sharing is greater than 0. If the database is of low or medium cardinality, i.e., below 8000 objects, KL(GGPla) or FM(GGPla) are the algorithms of choice: (1) Because KL is based on swapping objects, it is superior to FM if the standard deviation of the object size is small. (2) Beyond a standard deviation of around 300, FM achieves better results. For a database cardinality below 2000 objects HP can be used as an alternative to KL/FM (in the majority of the experiments we conducted HP produced the best results). When sharing is high, the page capacity is small, and the object sizes vary strongly (upper right grid corner), the FM heuristics is superior to KL because in this special case, moving objects achieves better results than swapping objects (see Section 5.3). Beyond a database cardinality of 8000 objects, the GGPla heuristics is the only method yielding acceptable cluster quality under reasonably low running-times.

7 Conclusion

Reconsider the title of this paper "Partition-Based Clustering in Object Bases: From Theory to Practice". The theory of partition-based clustering was developed by Tsangaris and Naughton [TN91]. They, in particular, proved the superiority of partition-based techniques over the sequence-based approaches. However, the hitherto known partition-based cluster approaches had to rely on the known computationally rather complex iterative partitioning algorithms, e.g., the Kernighan and Lin heuristics. Therefore, partition-based clustering was impractical for realistically large databases. Our greedy graph partitioning heuristics GGPla supplemented with the *look-ahead* and *new-chance* of once rejected choices has a much lower running-time complexity than the iterative algorithms. Nevertheless, the partitioning quality generated by GGPla is quite good compared to the computationally complex iterative partitioning heuristics and far better than sequence-based clustering algorithms. This makes the GGPla heuristics indispensable for two purposes:

1. As a pre-partitioning heuristics for small to medium cardinality object nets—on which iterative algorithms are then applied for improvement.
2. As the only applicable method on large cardinality object nets.

In conclusion, we therefore propose an adaptable clustering strategy which applies the most appropriate iterative partitioning method—in conjunction with a pre-partitioning by GGPla—on small to medium cardinality object nets and (pure) GGPla on large cardinality object nets. The adaptability is controlled by a multi-dimensional grid whose dimensions correspond to the object net characteristics which have to be established by, e.g., sampling the object base extension. The entries of the grid then determine the most appropriate cluster algorithm for the particular object base configuration.

Acknowledgement This work was partially supported by the German Research Council DFG under contracts Ke 401/6-1 and SFB 346.

We thank S. Fahrig for his help in the implementation and extensive benchmarking of the various partitioning algorithms.

References

[Ben90] V. Benzaken. An evaluation model for clustering strategies in the O_2 object-oriented database system. In *Proc. of the Int. Conf. on Database Theory (ICDT)*, pages 126–140. Springer-Verlag, 1990.

[BKKG88] J. Banerjee, W. Kim, S. J. Kim, and J. F. Garza. Clustering a DAG for CAD databases. *IEEE Trans. Software Eng.*, 14(11):1684–1699, Nov 1988.

[Bre77] M. A. Breuer. A class of min-cut placement algorithms. In *Proc. of the Design Automation Conference*, pages 284–290, 1977.

[CDRS86] M. Carey, D. DeWitt, J. Richardson, and E. Shekita. Object and file management in the EXODUS extensible database system. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 91–100, Kyoto, Japan, Aug 86.

[FM82] C. Fidducia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. of the Design Automation Conference*, pages 175–181, 1982.

[GKKM92a] C. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte. Clustering in object bases. Technical Report 6/92, Fakultät für Informatik, Universität Karlsruhe, D-7500 Karlsruhe, Jun 1992.

[GKKM92b] C. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte. Partition-based clustering in object bases: From theory to practice. Technical Report 92-34, RWTH Aachen, D-5100 Aachen, Dec 1992.

[HK89] S. E. Hudson and R. King. Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system. *ACM Trans. on Database Systems*, 14(3):291–321, Sep 1989.

[HZ87] M. Hornick and S. Zdonik. A shared, segmented memory system for an object-oriented database. *ACM Trans. Office Inf. Syst.*, 5(1):70–95, Jan 1987.

[KL70] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, Feb 1970.

[Kri84] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. on Comp.*, 33(5):438–446, 1984.

[Kru56] J. B. Kruskal. On the shortest spanning subgraph of a graph and the travelling salesman problem. *Proc. of the Amer. Math. Soc.*, 7:48–50, 1956.

[Luk74] J. Lukes. Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*, 18:217–224, 1974.

[SM91] K. Shahookar and P. Mazumber. VLSI cell placement techniques. *ACM Computing Surveys*, 23(2):143–220, Jun 1991.

[Sta84] J. Stamos. Static grouping of small objects to enhance performance of a paged virtual memory. *ACM Trans. Comp. Syst.*, 2(2):155–180, May 1984.

[TN91] M. M. Tsangaris and J. F. Naughton. A stochastic approach for clustering in object bases. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 12–21, Denver, CO, May 1991.

[TN92] M. M. Tsangaris and J. F. Naughton. On the performance of object clustering techniques. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 144–153, San Diego, CA, Jun 1992.