# Scalable Spatial GroupBy Aggregations Over Complex Polygons

Laila Abdelhafeez[a,b] Amr Magdy[a,b] Vassilis J. Tsotras[a,b]
[a]Department of Computer Science and Engineering [b]Center for Geospatial Sciences
University of California, Riverside
labde005@ucr.edu {amr,tsotras}@cs.ucr.edu

## ABSTRACT

This paper studies a *spatial group-by query* over complex polygons. Groups are selected from a set of non-overlapping complex polygons, typically in the order of thousands, while the input is a large-scale dataset that contains hundreds of millions or even billions of spatial points. Given a set of spatial points and a set of polygons, the spatial group-by query returns the number of points that lie within boundaries of each polygon. This problem is challenging because real polygons (like counties, cities, postal codes, voting regions, etc.) are described by very complex boundaries. We propose a highly-parallelized query processing framework to efficiently compute the spatial group-by query. Our experimental evaluation with real data and queries has shown significant superiority over all existing techniques.

## CCS CONCEPTS

• **Information systems** → Data management systems; *MapReduce-based systems*; **Join algorithms**; **Query operators**.

## KEYWORDS

spatial, big data, group by, join, polygon, query processing

**ACM Reference Format:**
Laila Abdelhafeez[a,b] Amr Magdy[a,b] Vassilis J. Tsotras[a,b]. 2020. Scalable Spatial GroupBy Aggregations Over Complex Polygons. In *28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '20), November 3–6, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397536.3422222

## 1 INTRODUCTION

Spatial data is readily available in very large quantities through the emergence of various technologies. Examples include user-generated data from hundreds of millions of users, fine-granularity satellite data from both public and private sectors, ubiquitous IoT applications and traffic management data. Such excessively large spatial datasets are rich in information but also come with new challenges for data scientists who try to explore and analyze them efficiently in various applications. These challenges span the whole stack of spatial data management starting from revisiting fundamental queries and their variations so they are efficiently supported on the current volume scale.

In this paper, we address a spatial group-by query to efficiently support large-scale datasets that contain hundreds of millions or even billions of data points on real polygons with very complex perimeter geometries that have tens of thousands of points. Such overwhelming polygon complexity combined with large volume data is beyond any currently available techniques in the mainstream spatial data management systems. Given a set of spatial points and a set of complex polygons, our group-by query counts the number of spatial points within the boundaries of each polygon. So, it groups points by polygon boundaries. This query is a composition of the fundamental spatial range query using sets of polygons as spatial group-by conditions. These real polygons are heavily used by social scientists in various spatial statistical analysis applications, such as spatial regionalization, spatial harmonization, segregation analysis, join-count analysis, hot-spot and cold-spot analysis, and spatial autocorrelation analysis [29].

Traditionally, count aggregations over polygons are performed using filter-refine approaches [13, 15]. The filter phase retrieves a subset of data based on the polygon minimum bounding rectangle (MBR), then this subset is refined where each data point is tested against the exact polygon geometry using point-in-polygon checks. This approach is still used in modern distributed systems, e.g, GeoSpark [32]. However, it incurs significantly expensive computations on large data since spatial containment checks are highly complex and consume significant processing overhead. For example, running a single query for 100 million points over only 255 country borders takes an hour to finish, using a twelve-nodes GeoSpark cluster with a total memory of 1TB. Such inefficient runtime limits spatial data scientists from performing large-scale analysis on modern spatial datasets.

Existing approaches face a main challenge to handle modern large datasets efficiently. This challenge arises from the prohibitive computations of point-in-polygon checks on real complex polygons, due to the excessive number of points on the polygon perimeter.

To overcome this challenge, we propose the Spatial GroupBy Polygon Aggregate Counting (*SGPAC*), a highly-parallelized query processing framework to efficiently support spatial group-by queries in mainstream spatial data management systems. The *SGPAC* framework is able to efficiently aggregate counts for large-scale datasets over a large number of highly complex polygons. To this end, *SGPAC* crumbles both data points and query polygons into fine-granular pieces based on two-level spatial indexing. For real polygons, a two-level *clipper* significantly downsizes the number of perimeter points to considerably speed up computing group-by aggregates, while still ensuring exact results. The counting process

[0]This work is partially supported by the National Science Foundation, USA, under grants IIS-1849971, IIS-1901379 and SES-1831615.

over the new clipped polygons is highly-parallelizable and makes great use of the distributed computation resources.

We performed an extensive experimental evaluation with real spatial data and world-scale polygons. Our techniques have shown significant superiority over all existing techniques for real complex polygons. In the rest of this paper, we outline related work in Section 2, while Section 3 formally defines the problem. The proposed query processing is detailed in Section 4. Sections 5 and 6 present experimental evaluation and conclusions.

## 2 RELATED WORK

**Centralized techniques**. Aggregation over spatial polygons has been studied for long and several techniques have been proposed [2–5, 12, 13, 15, 17–23, 30, 35]; the majority of them produce exact results while there are also works that produce approximate results, e.g., [3, 18, 33]. The most widely-used techniques are based on the two-phase filter-refine approach [13, 15] that filters out irrelevant data based on polygon approximations, most commonly a minimum bounding rectangle (MBR) approximation, and then refines candidate points based on polygon perimeter geometry. Several works proposed more precise polygon approximations in the filtering phase. Brinkhoff et. al. [4] studied different approximations, namely rotated minimum bounding box (RMBB), minimum bounding circle (MBC), minimum bounding ellipse (MBE), convex hull (CH), and minimum bounding n-corner (n-C). Sidlauskas et. al. [30] improved filtering through clipping away empty spaces in the MBR. Other approaches proposed multi-step filtering [5, 19, 20] and rasterization-based polygon approximation [2, 3, 12, 33, 35] to reduce the candidate set size. Although the rich approximations result in a tight candidate set, they do not reduce the computation complexity of each point-in-polygon check, which depends on the number of polygon perimeter points. Kipf et. al. [18] solves an orthogonal point-polygon join query that takes a single point and outputs polygons that contain this point. This work assumes that all polygons are known beforehand, and thus can be indexed, while the data points are streamed.

Another direction in supporting polygon aggregation is polygon decomposition [21–23, 26]. Such decomposition reduces the complexity of a polygon query by dividing it into smaller polygons. The original polygon geometry is decomposed into regular forms, e.g., convex polygons, triangles, trapezoids, combinations of rectangles and triangles, or even smaller irregular polygons using a uniform grid. However, decomposition has not yet been adopted in distributed big spatial data systems. This is confirmed by recent surveys [11, 27] that examined work on range queries in modern big spatial systems. Current big spatial systems mostly focus on rectangular ranges and have limited support for arbitrary polygons with complex shapes and high-density perimeters, which is crucial for data analysis on real datasets, e.g., in social sciences, since real-world polygons are neither rectangular nor regular.

**Distributed and parallel techniques**. There are also recent works on addressing irregular polygon range queries using distributed partitioning techniques [14, 24, 25, 28] and parallel GPU-based techniques [1, 33, 34]. These techniques mostly rely on partitioning data across a cluster of machines or GPU cores so that one query is partitioned along with data partitioning, and then the query executes on multiple nodes/cores that have relevant data. Nodarakis et. al. [25] partitions data based on either a regular grid or angle-based partitioning. However, that work only supports convex polygons. Guo et. al. [14] partitions data using a quadtree, then different partitions work in parallel using a traditional filter-refine approach. Ray et. al. [28] partitions data based on either object size or point density to improve workload balance among nodes. Malensek et. al. [24] proposed bitmap-based filtering where the global view facilitates node selection based on the intersection with query polygons. While these distributed approaches tighten the candidate set and take advantage of parallelism, they do not reduce the computational complexity of point-in-polygon checks, which are the bottleneck. Therefore, even in distributed environments, their overall computational cost is still high on large-scale datasets and large real polygon sets. Also, GPU-based techniques are not widely incorporated in the mainstream spatial systems.

Our work follows the decomposition direction in distributed environments to speed up polygon aggregations. Compared to existing literature, our work is distinguished by inherently considering real complex polygons and large-scale datasets (that contain hundreds of millions of points) by identifying and reducing the main performance bottleneck.

## 3 PROBLEM DEFINITION

Consider a spatial dataset $D$ that consists of point objects. Each object $o \in D$ is represented by ($oid$, $lat$, $long$), where $oid$ is the object identifier and <$lat$,$long$> represent the latitude/longitude coordinates of the object's location in the two-dimensional space. Formally, an *SGPAC* query $q$ is defined by a set of polygons $L$ as follows:

***Definition 1: Spatial GroupBy Polygon Aggregate Counting (SGPAC) Query***. Given a spatial dataset $D$, a query $q$ defined by a set of polygons $L = \{l_1, l_2, ..., l_m\}$, returns a set of $m$ integers $\{c_1, c_2, ..., c_m\}$, where $c_i$ is the number of objects $o_j \in D$ so that $o_j$'s location lies inside polygon $l_i \in L$.

Each polygon $l_i \in L$ is represented by $n_i$ spatial points that define its perimeter geometry. For large values of $m$ and $n_i$ and a large-scale spatial dataset $D$ with hundreds of millions of points, scaling a spatial group-by query is highly challenging.

As with typical relational group-by queries, the query groups can be selected in an ad-hoc manner. The polygon sets that are used in different applications, e.g., social sciences, could be either pre-defined polygons, e.g., US states borders, or arbitrary polygons, e.g., polygons produced by regionalization and harmonization algorithms [29]. Here we assume the general case, where set $L$ is not known apriori (and thus cannot be indexed beforehand). Since groups in a typical group-by clause are disjoint, the polygon set consists of disjoint polygons. However, our processing algorithm can also support polygon sets with overlapping polygons.

## 4 QUERY PROCESSING FRAMEWORK

The rationale behind this framework depends on two observations. The first observation recognizes that the main performance bottleneck of polygon aggregations is the high computational cost of point-in-polygon checks. So, our framework relies on significantly reducing the complexity of these checks to minimize the overall

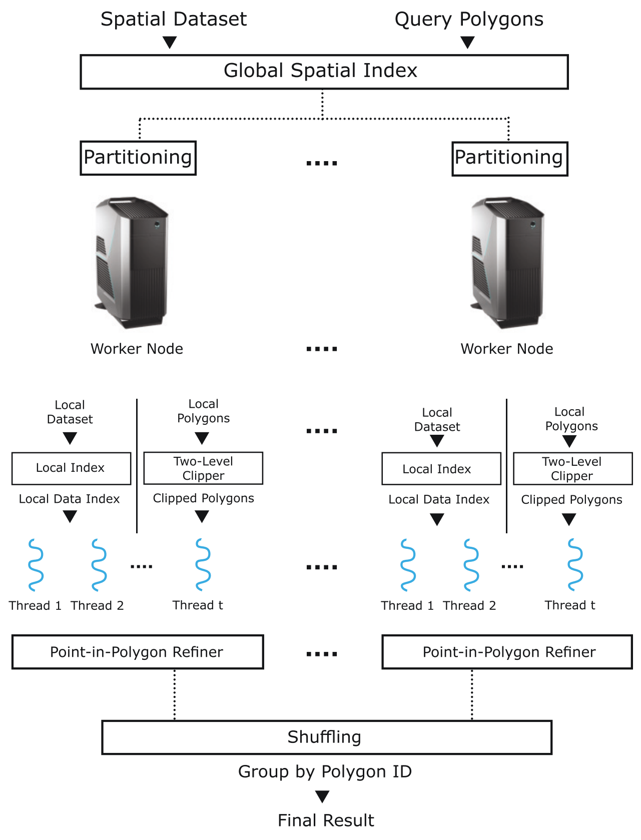


**Figure 1: Query Processing Framework Overview**

cost of large polygon sets aggregations. The second observation is that in nowadays applications, large-scale spatial datasets are usually indexed on distributed big spatial data systems, such as GeoSpark [32], Simba [31], GeoMesa [16], or SpatialHadoop [10]. Therefore, we develop a query processing framework that exploits such distributed indexing infrastructure in reducing the computational cost of spatial group-by queries to make our techniques applicable to a wide variety of existing applications and platforms.

Figure 1 shows an overview of our query processing framework. The framework exploits partitioning, by facilitating a global distributed spatial index to partition both data points and query polygons across different machines (distributed worker nodes). Each worker node $j$ covers a specific spatial area represented with a minimum bounding rectangle (MBR) $B_j$. Then, on each worker node, the local portion of data points is indexed with a local spatial index, which does not necessarily have the same structure as the global index. The local index further divides data into small chunks. Meanwhile, when a new query polygon set $L$ arrives, each worker node receives a subset $L_j$ of query polygons that overlap with its partition MBR $B_j$; that is, for all $l_i \in L_j$, $l_i \cap B_j \neq \phi$. Each polygon $l_i \in L_j$ goes through a *Two-level Clipper* module that significantly reduces the complexity of its perimeter through two phases of polygon clipping. The first phase is based on the global index partition boundaries $B_j$. This phase replaces $l_i$ with $l_i \cap B_j$, its intersection with the partition MBR, as any part of the polygon outside $B_j$ will not produce any results from the data points assigned to node $j$. The newly clipped polygon $l_i$ is passed as an input to the second level of clipping, which further clips $l_i$ based on the local index partitions to produce multiple smaller polygons, each of them corresponding to one of the local index partitions and clipped with its MBR boundaries.

After the two-level clipping of input polygons, the query input turns into small crumbles of both local data partitions and simple query polygons that are fed to a multi-threaded *Point-in-Polygon Refiner* module. This module takes pairs of data partitions and clipped query polygons with overlapping boundaries, where each pair follows one of two cases. The first case is that the boundaries of both the local partition and the clipped query polygon are the same. This means that the local partition is wholly contained inside the query polygon and all data points are counted in the result set without further refinement. The second case is that the clipped query polygon intersects with part of the local partition boundaries. In this case, the refinement module iterates over all the points within the local partition and simply uses the standard point-in-polygon algorithms to filter out points that are outside the polygon boundaries. Such point-in-polygon operation is much less expensive on the clipped polygon compared to the original polygon, with up to an order of magnitude cost reduction as shown in our experiments. Each thread maintains a list of <polygon id, count> pairs that record the count of points in each polygon. Lists of pairs from different threads and partitions are forwarded to a shuffling phase that aggregates total counts of each input polygon, based on polygon ids, in a similar fashion to the standard map-reduce word counting procedure.

## 5 EXPERIMENTAL EVALUATION

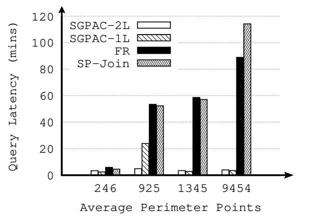
We evaluate the performance of *SGPAC* in terms of query latency using a real implementation based on GeoSpark [32].

**Experimental setup**. Our parameter is the average number of points per polygon perimeter (representing the complexity of polygon geometry). The global index is a quadtree with default capacity of 30K points and the local index is a uniform grid index with grid granularity of 10KMx10KM.

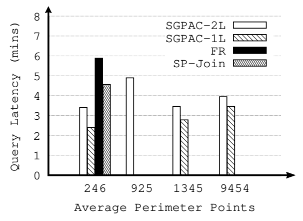
**Evaluation datasets**. Our evaluation data is a Twitter dataset that contains 100 million real geotagged tweets spatially distributed worldwide. For query polygons, we use real polygons sets that represent four different multi-scale spatial layers representing borders of continents, countries, provinces, and counties worldwide [6–9].

**Evaluated alternatives**. We evaluate our *SGPAC* technique (denoted as *SGPAC-2L*) against three alternatives: (1) A variation of *SGPAC* (denoted as *SGPAC-1L*) that employs only one-level clipping based on global index partitions and ignores local index clipping. (2) A distributed filter-refine approach (denoted as *FR*) that uses the popular MBR-based filter-refine (as discussed in Section 2) on each worker node in parallel. (3) A spatial join based approach (denoted as *SP-Join*) that partitions both data points and polygons based on the global index, and then performs nested loop spatial join on each worker node in parallel.

**Query evaluation**. Figure 2 evaluates the query performance of the different techniques on the four real polygon sets of continents, countries, provinces, and counties. The figure presents the query performance in ascending order of the average number of perimeter



(a) Varying average perimeter points



(b) Magnified view for Figure 2(a)

**Figure 2: Query Performance on Real Polygon Layers.**

points $n$ per polygon (in Figure 2(a) and magnified in Figure 2(b)). Both *FR* and *SP-Join* perform much worse than all *SGPAC* variations in three of the four layers, where their query latency is 2-23 times slower than *SGPAC*. All *SGPAC* variations take under 4 minutes to process all query polygons on the 100 million data points, except in one case where *SGPAC-1L* takes 24 minutes. For *FR*, latency ranges from 6-89 minutes, and for *SP-Join* latency ranges from 4.5-114 minutes, depending on the query polygons characteristics. In particular, Figure 2 gives the following major insights:

(1) The first insight is the high variability in query speedup for different polygon layers. For continents and countries, *FR* and *SP-Join* are significantly much slower than all *SGPAC* variations, with 12-23 times slower latency. However, for counties, they have latency that is only twice slower than *SGPAC* variations, while for provinces the improvement is minor with 1.3-1.7 times slower latency. This is interpreted by the variability in polygon complexity, in terms of the number of perimeter points. Different techniques have increasing latency with increasing polygon complexity. The provinces set has the simplest polygon perimeters, with an average of 246 points per perimeter; here *FR* and *SP-Join* still perform reasonably with 4.5-5.9 minutes latency compared to 3.4 minutes for *SGPAC-2L*. When the polygon complexity increases, with a higher number of perimeter points, the *SGPAC* variations perform much better due to our proposed polygon complexity reduction.

(2) The second insight is the effectiveness of the second level of *SGPAC* two-level clipping for reducing perimeter complexity. As shown in Figure 2, the *SGPAC-1L* approach (that ignores second level clipping) performs comparably to *SGPAC-2L* except in case of counties that have the smallest area ($5.7km^2$), where *SGPAC-1L* performs five times slower than *SGPAC-2L*. In that case, many county polygons are small enough to fit in the local machine without being clipped by the global index; that is, the polygon complexity is not reduced by the first level clipping. In such scenarios, ignoring the second level of clipping leads to much more expensive point-in-polygon checks and much higher overall latency.

## 6 CONCLUSIONS

This paper proposes a spatial group-by query that groups spatial points based on the boundaries of a set of complex polygons. This query designed for thousands of polygons with complex perimeters and large-scale spatial datasets that contain hundreds of millions of points. We have proposed a highly efficient query processor that uses existing distributed spatial data systems to support scalable spatial group-by queries. Our technique depends on reducing query polygons complexity by clipping them based on global and local spatial indexes. The experimental evaluation showed significant superiority for our techniques over existing competitors.

## REFERENCES

[1] D. Aghajarian, S. Puri, and S. Prasad. GCMF: An Efficient End-to-End Spatial Join System Over Large Polygonal Datasets on GPGPU Platform. In *SIGSPATIAL*, 2016.
[2] L. G. Azevedo, R. H. Güting, R. B. Rodrigues, G. Zimbrão, and J. M. de Souza. Filtering With Raster Signatures. In *ACM GIS*, 2006.
[3] L. G. Azevedo, G. Zimbrão, and J. M. De Souza. Approximate Query Processing in Spatial Databases Using Raster Signatures. In *Advances in Geoinformatics*. Springer, 2007.
[4] T. Brinkhoff, H.-P. Kriegel, and R. Schneider. Comparison of Approximations of Complex Objects Used For Approximation-based Query Processing in Spatial Database Systems. In *ICDE*, 1993.
[5] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-step Processing of Spatial Joins. *SIGMOD*, 23(2), 1994.
[6] ArcGIS Continents. https://www.arcgis.com/home/item.html?id=5cf4f223c4a642eb9aa7ae1216a04372, 2017.
[7] GADM Counties. https://gadm.org/download_world.html.
[8] NE Countries. https://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-0-countries/, 2009.
[9] NE Provinces. https://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-1-states-provinces/, 2009.
[10] A. Eldawy and M. F. Mokbel. Spatialhadoop: A Mapreduce Framework for Spatial Data. In *ICDE*, 2015.
[11] A. Eldawy and M. F. Mokbel. The Era of Big Spatial Data. In *ICDEW*, 2015.
[12] Y. Fang, M. Friedman, G. Nair, M. Rys, and A.-E. Schmid. Spatial indexing in microsoft sql server 2008. In *SIGMOD*, 2008.
[13] A. U. Frank. Application of DBMS to Land Information Systems. In *VLDB*, 1981.
[14] Q. Guo, B. Palanisamy, H. A. Karimi, and L. Zhang. Distributed Algorithms for Spatial Retrieval Queries in Geospatial Analysis. *STCC*, 4(3), 2016.
[15] R. H. Güting. An Introduction to Spatial Database Systems. *VLDB Journal*, 1994.
[16] J. N. Hughes, A. Annex, C. N. Eichelberger, A. Fox, A. Hulbert, and M. Ronquest. Geomesa: A Distributed Architecture for Spatio-Temporal Fusion. In *SPIE*, volume 9473, 2015.
[17] E. H. Jacox and H. Samet. Spatial Join Techniques. *TODS*, 32(1), 2007.
[18] A. Kipf, H. Lang, V. Pandey, R. A. Persa, C. Anneser, E. T. Zacharatou, H. Doraiswamy, P. A. Boncz, T. Neumann, and A. Kemper. Adaptive Main-Memory Indexing for High-Performance Point-Polygon Joins. In *EDBT*, 2020.
[19] R. K. Kothuri and S. Ravada. Efficient Processing of Large Spatial Queries Using Interior Approximations. In *SSTD*, 2001.
[20] R. K. V. Kothuri, S. Ravada, and D. Abugov. Quadtree and R-tree indexes in Oracle Spatial: A Comparison Using GIS Data. In *SIGMOD*, 2002.
[21] H.-P. Kriegel, H. Horn, and M. Schiwietz. The Performance of Object Decomposition Techniques for Spatial Query Processing. In *SSTD*, 1991.
[22] Y.-J. Lee, D.-M. Lee, S.-J. Ryu, and C.-W. Chung. Controlled Decomposition Strategy for Complex Spatial Objects. In *DEXA*, 1996.
[23] Y.-J. Lee, H.-H. Park, N.-H. Hong, and C.-W. Chung. Spatial Query Processing Using Object Decomposition Method. In *CIKM*, 1996.
[24] M. Malensek, S. Pallickara, and S. Pallickara. Polygon-based Query Evaluation Over Geospatial Data Using Distributed Hash Tables. In *UCC*, 2013.
[25] N. Nodarakis, S. Sioutas, P. Gerolymatos, A. Tsakalidis, and G. Tzimas. Convex Polygon Planar Range Queries on the Cloud: Grid vs. Angle-based Partitioning. In *ALGOCLOUD*, 2015.
[26] J. A. Orenstein. Redundancy in Spatial Databases. *SIGMOD*, 18(2), 1989.
[27] V. Pandey, A. Kipf, T. Neumann, and A. Kemper. How Good are Modern Spatial Analytics Systems? *VLDB*, 11(11), 2018.
[28] S. Ray, B. Simion, A. D. Brown, and R. Johnson. Skew-Resistant Parallel in-memory Spatial Join. In *SSDBM*, 2014.
[29] S. J. Rey and L. Anselin. PySAL: A Python Library of Spatial Analytical Methods. *The Review of Regional Studies*, 37(1), 2007.
[30] D. Sidlauskas, S. Chester, E. T. Zacharatou, and A. Ailamaki. Improving Spatial Data Processing by Clipping Minimum Bounding Boxes. In *ICDE*, 2018.
[31] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient in-memory spatial Analytics. In *SIGMOD*, 2016.
[32] J. Yu, Z. Zhang, and M. Sarwat. Spatial Data Management in Apache Spark: the GeoSpark Perspective and Beyond. *GeoInformatica*, 2018.
[33] E. T. Zacharatou, H. Doraiswamy, A. Ailamaki, C. T. Silva, and J. Freiref. GPU Rasterization for Real-time Spatial Aggregation Over Arbitrary Polygons. *VLDB*, 11(3), 2017.
[34] J. Zhang and S. You. Speeding up Large-Scale Point-in-Polygon Test Based Spatial Join on GPUs. In *SIGSPATIAL*, 2012.
[35] G. Zimbrao and J. M. De Souza. A Raster Approximation for Processing of Spatial Joins. In *VLDB*, 1998.