

```

while l1 ≠ ERROR do
  tcl ← tuple of ctcl corresponding to l1.tid cr
  if distance (tcl.location, ts.location) < 2 and cd
    tcl.image_id = ts.image_id then
    pi ← tuple t of physical_images caf + cr
      such that t.image_id = ts.image_id
    Display pi.raw, Print cl.semantics
  l1 ← next tuple of ctcl_loc intersecting R csn

```

**Plan P4<sub>I</sub>** Search using alphanumeric indices on class for all airfield tuples and all beach tuples.

```

cda ← first tuple t of cl_semant such that t.semantics >= "airfield" caf
ca ← tuple of classes corresponding to cda.tid cr
cdb ← first tuple t of cl_semant such that t.semantics >= "beach" caf
cb ← tuple of classes corresponding to cdb.tid cr
lca ← first tuple t of li_cl such that t.class >= ca.name caf
while lca ≠ ERROR and lca.class = ca.name do
  lia ← tuple of logical_images corresponding to lca.tid cr
  lcb ← first tuple t of li_cl such that t.class >= cb.name caf
  while lcb ≠ ERROR and lcb.class = cb.name do
    lib ← tuple of logical_images corresponding to lcb.tid cr
    if lia.location(y) > lib.location(y) and
      lib.image_id = lia.image_id then
      pi ← tuple t of physical_images caf + cr
        such that t.image_id = lia.image_id
      Display pi.raw
    lcb ← next tuple of li_cl in alphabetic order can
  lca ← next tuple of li_cl in alphabetic order can

```

**Plan P4<sub>P</sub>** Search the airfield and the beach partitions sequentially .

```

cda ← first tuple t of cl_semant such that t.semantics >= "airfield" caf
ca ← tuple of classes corresponding to cda.tid cr
cdb ← first tuple t of cl_semant such that t.semantics >= "beach" caf
cb ← tuple of classes corresponding to cdb.tid cr
cta ← get_rel_name(ca.name) cnc
ctb ← get_rel_name(cb.name) cnc
for all tuples ta of cta (access each tuple sequentially) cs
  for all tuples tb of ctb (access each tuple sequentially) cs
    if ta.location(y) > tb.location(y) and cd
      ta.image_id = tb.image_id then
      pi ← tuple t of physical_images caf + cr
        such that t.image_id = ta.image_id
      Display pi.raw

```

```

ctp ← get_rel_name(cp.name) Cnc
for all tuples tp of ctp (access each tuple sequentially) Cs
  R ← 10 × 10 rectangle with lower left corner at
    (tp.location(x) - 5, tp.location(y) - 5)
  ll ← first tuple t of cts_loc such that t.location intersects R Csf
  while ll ≠ ERROR do
    ts ← tuple of cts corresponding to ll.tid Cr
    if distance (tp.location, ts.location) < 5 and Cd
      tp.image_id = ts.image_id then
        pi ← tuple t of physical_images
          such that t.image_id = tp.image_id Caf + Cr
        Display pi.raw
  ll ← next tuple of cts_loc intersecting R Csn

```

**Plan P3<sub>I</sub>** Search for site of interest tuples using an alphanumeric index on `class` and search for points in the given range using a spatial index on `location`.

```

cde ← first tuple t of cl_semant such that t.semantics >= "site of interest" Caf
cs ← tuple of classes corresponding to cde.tid Cr
lc ← first tuple t of li_cl such that t.class >= cs.name Caf
while lc ≠ ERROR and lc.class = cs.name do
  lis ← tuple of logical_images corresponding to lc.tid Cr
  R ← 4 × 4 rectangle with lower left corner at
    (lis.location(x) - 2, lis.location(y) - 2)
  ll ← first tuple t of li_loc such that t.location intersects R Csf
  while ll ≠ ERROR do
    lic ← tuple of logical_images corresponding to ll.tid Cr
    if distance (lic.location, lis.location) < 2 and Cd
      lic.image_id = lis.image_id then
        pi ← tuple t of physical_images
          such that t.image_id = lic.image_id Caf + Cr
        cd ← tuple t of classes such that t.name = lic.class Caf + Cr
        Display pi.raw, Print cd.semantics
  ll ← next tuple of li_loc intersecting R Csn
lc ← next tuple of li_cl in alphabetic order Can

```

**Plan P3<sub>P</sub>** Search the site of interest partition sequentially, search all other partitions using the spatial index on `location`.

```

cde ← first tuple t of cl_semant such that t.semantics >= "site of interest" Caf
cs ← tuple of classes corresponding to cde.tid Cr
cts ← get_rel_name(cs.name) Cnc
for all tuples cl of classes (access each tuple sequentially) Cs
  ctcl ← get_rel_name(cl.name) Cnc
  for all tuples ts of cts (access each tuple sequentially) Cs
    R ← 4 × 4 rectangle with lower left corner at
      (ts.location(x) - 2, ts.location(y) - 2)
    ll ← first tuple t of ctcl_loc such that t.location intersects R Csf

```

**Plan P2A<sub>P</sub>:** Search both the picnic and scenic view partitions sequentially.

```

cds ← first tuple t of cl_semant such that t.semantics >= "scenic view"           caf
cs ← tuple of classes corresponding to cds.tid                                   cr
cdp ← first tuple t of cl_semant such that t.semantics >= "picnic"             caf
cp ← tuple of classes corresponding to cdp.tid                                 cr
cts ← get_rel_name(cs.name)                                                  cnc
ctp ← get_rel_name(cp.name)                                                  cnc
for all tuples tp of ctp (access each tuple sequentially)                      cs
    for all tuples ts of cts (access each tuple sequentially)                  cs
        if distance (tp.location, ts.location) < 5 and                          cd
            tp.image_id = ts.image_id then
                pi ← tuple t of physical_images
                    such that t.image_id = tp.image_id                          caf + cr
                Display pi.raw

```

**Plan P2B<sub>I</sub>:** Search for picnic tuples using an alphanumeric index on class and search for scenic view tuples using a spatial index on location.

```

cds ← first tuple t of cl_semant such that t.semantics >= "scenic view"           caf
cs ← tuple of classes corresponding to cds.tid                                   cr
cdp ← first tuple t of cl_semant such that t.semantics >= "picnic"             caf
cp ← tuple of classes corresponding to cdp.tid                                 cr
lc ← first tuple t of li_cl such that t.class >= cp.name                       caf
while lc ≠ ERROR and lc.class = cp.name do
    lip ← tuple of logical_images corresponding to lc.tid                       cr
    R ← 10 × 10 rectangle with lower left corner at
        (lip.location(x) - 5, lip.location(y) - 5)
    ll ← first tuple t of li_loc such that t.location intersects R              csf
    while ll ≠ ERROR do
        lis ← tuple of logical_images corresponding to ll.tid                  cr
        if lis.class = cs.name then
            if distance (lis.location, lip.location) < 5 and                    cd
                lis.image_id = lip.image_id then
                    pi ← tuple t of physical_images
                        such that t.image_id = lip.image_id                    caf + cr
                    Display pi.raw
            ll ← next tuple of li_loc intersecting R                            csn
    lc ← next tuple of li_cl in alphabetic order                                can

```

**Plan P2B<sub>P</sub>:** Search picnic partition sequentially, search scenic view partition using the spatial index on location for the scenic view partition.

```

cds ← first tuple t of cl_semant such that t.semantics >= "scenic view"           caf
cs ← tuple of classes corresponding to cds.tid                                   cr
cdp ← first tuple t of cl_semant such that t.semantics >= "picnic"             caf
cp ← tuple of classes corresponding to cdp.tid                                 cr
cts ← get_rel_name(cs.name)                                                  cnc

```

## Appendix A Sample Plans

The letters in *italic* at the end of each line represent the cost of executing this line in terms of the constants defined in Figure 12. Notice that the cost of the “display” operation is not included since it is not considered part of processing the query. It is only a mechanism to output the answer to the query and is always the same regardless of the selected execution plan. In addition, no cost is associated with operations that compare two values (e.g., =, <).

**Plan P1<sub>I</sub>:** Search using an alphanumeric index on `class`.

```

cd ← first tuple t of cl_semant such that t.semantics >= "scenic view"           caf
c ← tuple of classes corresponding to cd.tid                                     cr
lc ← first tuple t of li_cl such that t.class >= c.name                         caf
while lc ≠ ERROR and lc.class = c.name do
  li ← tuple of logical_images corresponding to lc.tid                          cr
  pi ← tuple t of physical_images such that t.image_id = li.image_id           caf + cr
  Display pi.raw
  lc ← next tuple of li_cl in alphabetic order                                 can

```

**Plan P1<sub>P</sub>:** Search the `scenic view` partition sequentially.

```

cd ← first tuple t of cl_semant such that t.semantics >= "scenic view"         caf
c ← tuple of classes corresponding to cd.tid                                     cr
ct ← get_rel_name(c.name)                                                       cnc
for all tuples t of ct (access each tuple sequentially)                         cs
  pi ← tuple tp of physical_images such that tp.image_id = t.image_id          caf + cr
  Display pi.raw

```

**Plan P2<sub>A<sub>I</sub></sub>:** Search using alphanumeric indices on `class` for all picnic tuples and all scenic view tuples.

```

cds ← first tuple t of cl_semant such that t.semantics >= "scenic view"       caf
cs ← tuple of classes corresponding to cds.tid                               cr
cdp ← first tuple t of cl_semant such that t.semantics >= "picnic"           caf
cp ← tuple of classes corresponding to cdp.tid                               cr
lcp ← first tuple t of li_cl such that t.class >= cp.name                  caf
while lcp ≠ ERROR and lcp.class = cp.name do
  lip ← tuple of logical_images corresponding to lcp.tid                       cr
  lcs ← first tuple t of li_cl such that t.class >= cs.name                 caf
  while lcs ≠ ERROR and lcs.class = cs.name do
    lis ← tuple of logical_images corresponding to lcs.tid                     cr
    if distance (lis.location, lip.location) < 5 and                          cd
      lis.image_id = lip.image_id then
      pi ← tuple t of physical_images
        such that t.image_id = lis.image_id                                     caf + cr
      Display pi.raw
    lcs ← next tuple of li_cl (after lcs) in alphabetic order                 can
  lcp ← next tuple of li_cl (after lcp) in alphabetic order                 can

```

- [5] J. P. Cheiney and B. Kerherve. Image storage and manipulation for multimedia database systems. In *Proceedings of the 4th International Symposium on Spatial Data Handling*, pages 611–620, Zurich, Switzerland, July 1990.
- [6] C. Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, March 1985.
- [7] W. I. Grosky and R. Mehrotra. Index-based object recognition in pictorial data management. *Computer Vision, Graphics, and Image Processing*, 52(3):416–436, December 1990.
- [8] A. Gupta, T. Weymouth, and R. Jain. Semantic queries with pictures: the VIMSYS model. In G. Lohman, editor, *Proceedings of the Seventeenth International Conference on Very Large Databases*, pages 69–79, Barcelona, September 1991.
- [9] R. Jain. Proceedings of the NSF workshop on visual information management systems. 1993.
- [10] T. Joseph and A.F. Cardenas. PICQUERY: A high level query language for pictorial database management. *IEEE Transactions on Software Engineering*, 14(5):630–638, May 1988.
- [11] S. T. Lee and M. K. Shan. Access methods of image database. *International Journal of Pattern Recognition and Artificial Intelligence*, 4(1):27–44, January 1990.
- [12] R. C. Nelson and H. Samet. A consistent hierarchical representation for vector data. *Computer Graphics*, 20(4):197–206, August 1986. (also Proceedings of the SIGGRAPH’86 Conference, Dallas, August 1986).
- [13] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content using color, texture, and shape. Technical Report RJ 9203, IBM Almaden Research Center, San Jose, CA, February 1993.
- [14] J. Orenstein and F.A. Manola. PROBE spatial modeling and query processing in an image database application. *IEEE Transactions on Software Engineering*, 14(5):611–628, May 1988.
- [15] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, April 1994.
- [16] C. Chennubhotla R. Kasturi, R. Raman. Document image analysis an overview of techniques for graphics recognition. In *Proceedings of the IAPR Workshop on Syntactic and Structural Pattern Recognition*, pages 192–230, Murray Hill, New Jersey, June 1990.
- [17] D. Rotem. Spatial join indices. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 500–509, Kobe, Japan, April 1991. IEEE Computer Society, IEEE Computer Society Press.
- [18] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, May 1988.
- [19] H. Samet and A. Soffer. Automatic interpretation of floor plans using spatial indexing. In S. Impedovo, editor, *Progress in Image Analysis and Processing III*, pages 233–240. World Scientific, Singapore, 1994.
- [20] H. Samet and A. Soffer. A legend-driven geographic symbol recognition system. In *Proceedings of the 12th International Conference on Pattern Recognition*, volume II, pages 350–355, Jerusalem, Israel, October 1994.

Two different data organizations (integrated and partitioned) for storing logical images in relational tables were proposed. They differ in the way that the logical images are stored. Sample queries and execution plans to answer these queries were described for both organizations. Analytical as well as empirical cost analyses of these execution plans were given and the partitioned data organization proved to be the most efficient for queries that consist of both contextual and spatial specifications. On the other hand, the integrated organization proved to be better for purely spatial specifications. Both organizations gave similar results for queries that consist of purely contextual specification.

The data set used for our experiments was not very large. A more realistic data set for this sample application would most likely have a much larger number of images and as a result a much larger number of tuples in each relation. We currently have only scanned one small portion of the map of Finland in order to test our ideas. Acquiring more data and testing on a larger data set is subject of future research. The fact that even on a relatively small data set, a significant improvement in execution time was achieved by use of spatial indexing and data partitioning is very encouraging. On a larger data set the effect of our methods should be even more pronounced.

Our definition of the class of images that we can handle is rather strict. Some of these restrictions can be relaxed. In particular, the requirement that there exists a function  $f$  which when given a symbol  $s$  and a class  $C$  returns a value between 0 and 1 indicating the certainty that  $s$  belongs to  $C$  can be omitted. In this case, we can store the feature vectors in the database rather than the classifications. More elaborate indexing methods would then be required to respond to queries such as those presented in this paper. This is also a subject of future research. All of the examples and experiments in this paper were from the map domain. However, images from many other interesting applications fall into the category of symbolic images. These include CAD/CAM, engineering drawings, floor plans, and more. Hence, the methods that we describe in this paper are applicable to them as well. Note that we have used similar methods for the interpretation of floor plans [19].

## 8 Acknowledgements

We are grateful to Karttakeskus, Map Center, Helsinki, Finland for providing us the map data.

## References

- [1] W. G. Aref and H. Samet. Optimization strategies for spatial query processing. In G. Lohman, editor, *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 81–90, Barcelona, September 1991.
- [2] S. K. Chang and A. Hsu. Image information systems: Where do we go from here? *IEEE Transactions on Knowledge and Data Engineering*, 4(5):431–442, 1992.
- [3] S. K. Chang, Q. Y. Shi, and C. Y. Yan. Iconic indexing by 2-D strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.
- [4] S. K. Chang, C. W. Yan, D. C. Dimitroff, and T. Arndt. An intelligent image database system. *IEEE Transactions on Software Engineering*, 14(5):681–688, May 1988.

Plan P2B does make use of a spatial index and here we see a significant difference between the partitioned and integrated organizations. In particular, the partitioned organization performs much better than the integrated organization. This is due to the significantly smaller number of spatial “find next” and “random access by tid” operations in the partitioned organization. In addition, plan P2B performs significantly better than plan P2A both in the integrated and partitioned organizations. As expected, using the spatial index improves the performance of such queries significantly. On the other hand, in the case of plan P3, the integrated organization performs much better than the partitioned organization. This is due mainly to the significantly larger number of spatial “find first” operations that are required in the partitioned organization since a spatial search is performed on each partition separately.

### 6.3 Discussion

In an image database, we may assume that the most prevalent operation is the retrieval of images. Images will usually be inserted in batches and should be stored in such a way that they can be retrieved efficiently. Update operations are very uncommon. The difference in the time required to respond to queries such as Q1–Q4 is the most important factor in choosing how to organize the data. As we have seen, the type of queries that are expected can also influence this choice. If most queries consist of only contextual specifications (i.e., retrieving images that contain a given object), then the additional complexity in writing plans required by the partitioned organization is probably not worthwhile. On the other hand, if most queries consist of both contextual and spatial specifications, then the significant improvement in the time required to answer such queries outweighs the overhead associated with the partitioned organization. Our results also show that for both data organizations, plans that use a spatial index perform much better than plans that do not make use of such an index. In the case of queries that consist of mostly spatial specifications, the partitioned organization proved to be a hindrance. The problem is that if the query does not specify which objects are required, then the computation needs to be repeated on each partition thereby resulting in very poor results. If these queries are predominant in the application, then the integrated organization should be chosen.

In our example application most queries consist of both contextual and spatial specifications. Thus, the partitioned organization is probably the most desirable. However, if queries such as Q3 are also expected, then some sort of hybrid approach may be necessary.

## 7 Concluding Remarks

Methods for integrating symbolic images into the framework of a conventional database management system (DBMS) have been described. Both the pattern recognition and indexing aspects of the problem have been addressed. By limiting ourselves to symbolic images, the task of object recognition was simplified, thereby enabling us to utilize well-known methods used in document processing in order to perform object extraction. The emphasis is on extracting both contextual and spatial information from the raw images. The logical image representation that we have defined preserves this information. Methods for storing and indexing logical images as tuples in a relation have been presented. Indices are constructed on both the contextual and the spatial data, thereby enabling efficient retrieval of images based on contextual as well as spatial specifications.

<i>Relation name</i>	<i>Integrated</i>	<i>Partitioned</i>
<code>classes</code>	2.6	2.6
<code>cl_semant</code>	1.9	1.9
<code>physical_images</code>	46.1	46.1
<code>pi_imid</code>	11.0	11.0
<code>pi_ll</code>	38.6	38.6
<code>logical_images / total CL_class</code>	95.3	47.2
<code>li_cl / total CL_cert</code>	124.5	58.6
<code>li_loc / total CL_loc</code>	94.3	128.27
total all relations	414.3	334.27

Figure 14: Sizes (in Kilobytes) of the relational tables in the integrated and partitioned data organizations.

from the relation name in all partitions. However, since the total difference in storage space is not very large, it should not play an important role in considering which organization is better.

## 6.2 Image Retrieval Results

Figure 15 displays various measures describing the execution time, the number of I/O operations, and the number of basic database operations (as defined in Section 5.3) that were required to process queries Q1–Q4. The performance of plans P1, P2A, and P4 are almost identical using the

<i>Measure</i>	<i>P1</i>		<i>P2A</i>		<i>P2B</i>		<i>P3</i>		<i>P4</i>	
	<i>Integ.</i>	<i>Part.</i>	<i>Integ.</i>	<i>Part.</i>	<i>Integ.</i>	<i>Part.</i>	<i>Integ.</i>	<i>Part.</i>	<i>Integ.</i>	<i>Part.</i>
time in user mode	1.02	1.05	38.71	41.95	24.49	15.21	7.40	97.67	18.63	21.13
time in system mode	0.59	0.55	10.30	4.28	12.10	7.39	3.36	47.36	7.83	3.05
total time	1.61	1.60	49.01	46.23	36.59	22.60	10.76	145.03	26.46	24.18
# of input operations	42	19	51	21	63	23	57	129	52	19
# of output operations	12	15	1138	1151	18	24	352	7721	18	24
# of page faults	127	117	167	108	173	134	169	239	151	102
# alphanumeric find first	189	188	27	26	27	26	164	82	37	36
# alphanumeric find next	187	0	15793	0	187	0	57	0	10830	0
# sequential access	0	187	0	15708	0	187	0	1276	0	10738
# random access by tid	375	188	15630	26	1635	64	308	171	10746	36
# spatial find first	0	0	0	0	187	187	56	1254	0	0
# spatial find next	0	0	0	0	1423	52	166	166	0	0
# geometric computations	0	0	15521	15521	52	52	166	166	5050	5050

Figure 15: Execution performance measures for plans P1–P4

two organizations with a slight advantage for the partitioned organization. All these plans only use alphanumeric indices; they do not use the spatial index. For these plans, the slight advantage of the partitioned organization is due to the absence of a need for an index on `class`. Thus, there is no need for dereferencing operations in order to access the tuples of the `logical_images` relation. This can be seen in Figure 15 by the larger number of “random access by tid” operations in the integrated organization compared to the number of these operations in the partitioned organization.



the logical images) were input to SAND and inserted into relations as defined in Section 4. SAND provides a programming environment where query execution plans can be created either manually or automatically. This programming environment is interfaced to Tcl (short for Tool Command Language), an interpreted scripting language developed by Ousterhout [15]. At the moment, the automatic creation of query evaluation plans is performed through a very primitive algorithm. Thus, we chose to create our plans manually. These plans are tcl script files. We created plans for each one of the queries listed in Section 5.1 following the strategies outlined in Section 5.2. These plans were executed on a Sparc 10 running UNIX, and statistics regarding the execution were recorded. We only report results for the database storage and retrieval performance here. For experimental results of the conversion process see [20].

## 6.1 Image Insertion Results

Figure 13 displays various measures describing the execution time and the number of I/O operations that were required when inserting the logical images into the relations. These results are only for inserting the data into the database. They do not include the time required for converting from physical to logical images, which is a time consuming process (i.e., it took approximately 4 hours for our data set). Time in user mode corresponds to the CPU time spent executing application

<i>Measure</i>	<i>Integrated</i>	<i>Partitioned</i>
time in user mode	4:39	6:51
time in system mode	1:23	1:34
total time	6:02	8:25
# of input operations	184	213
# of output operations	29,527	29,421
# of page faults	190	228

Figure 13: Image insertion times (min:sec) and number of I/O operations in the integrated and partitioned organizations.

code. Time in system mode corresponds to the CPU time spent processing system calls. (These are values that are reported by the UNIX `time` operation). Total time is the sum of these two quantities. Inserting the data into the partitioned organization is slightly slower than for the integrated organization. The reason for this is that some additional time is required in order to determine the particular partition (i.e., relation) that a given logical image tuple belongs too. That relation is then opened for insertion. In contrast, in the integrated organization all tuples are inserted into the same relation. However, since the difference in insertion time is relatively small, it should not play an important role in considering which organization is better.

Figure 14 gives the sizes of the various relations in the two organizations. The `classes` relation and `physical-images` relations along with their indices are identical in both organizations. Thus, their sizes are the same. The total space required for the `logical-images` relation and its indices in the partitioned organization is 25% less than that required for the relations of the integrated organization (234K vs. 314K, respectively). This is because the `class` is stored for each tuple both in the `logical-images` and the `li-cl` relations. On the other hand, in the partitioned organization, the class name is only stored once for each class in the `classes` relation. The class name is derived

site of interest whose location was the center of this circular range. This is the number of results reported when responding to query 3.  $N_{classes}$  denotes the number of tuples in relation `classes`.  $N_{CL-InR_3}$  denotes the number tuples of class `CL` in  $R_3$ .

$$C_{3I} = 2c_{af} + c_r + \tag{7}$$

$$N_{soi} \times [c_r + c_{sf} + N_{InR_3} \times (2c_r + c_d + c_{af} + c_{sn}) + N_{rslt_3} \times (c_{af} + c_r) + c_{an}]$$

$$C_{3P} = c_{af} + c_r + c_{nc} + \tag{8}$$

$$N_{classes} \times [c_s + c_{nc} + N_{soi} \times [c_s + c_{sf} + N_{CL-InR_3} \times (c_d + c_r + c_{sn}) + N_{rslt_3} \times (c_{af} + c_r)]]$$

For query 3, the integrated organization is clearly the one to use. The spatial index on `location` is used to retrieve exactly those tuples that are within the specified range. Since the class of the tuple was not used as a qualifier, there is no need for any further processing of these tuples. However, in the partitioned organization, each of the partitions needs to be searched separately. The number of spatial window queries is multiplied by the number of different classes in the application thereby resulting in a very inefficient computation.

Equations 9 and 10 estimate the cost of responding to query 4 using the integrated and partitioned organizations, respectively. Note that the plan for this query is very similar to plan A for query 2 in the case of both the integrated and partitioned data organizations. Observe that in this case, the alternative plan B is not even considered since the spatial range specified by the condition “north of” is very large. Thus, its selectivity is small and it is not worthwhile to perform a spatial search.  $N_{rslt_4}$  denotes the number of airfield tuples that are north of beach tuples and in the same image.

$$C_{4I} = 3c_{af} + 2c_r + \tag{9}$$

$$N_{air} \times [c_r + c_{af} + N_{bch} \times (c_r + c_{an}) + N_{rslt_4} \times (c_{af} + c_r) + c_{an}]$$

$$C_{4P} = 2c_{af} + 2c_r + 2c_{nc} + N_{air} \times [c_s + N_{bch} \times c_s + N_{rslt_4} \times (c_{af} + c_r)] \tag{10}$$

The main difference between these two cost estimates is that in the integrated organization, the index is scanned sequentially, whereas in the partitioned organization, the relation corresponding to the scenic view partition is scanned sequentially (as is the case of query 1). As a result, once again, there are considerably more random access operations in the integrated organization than in the partitioned organization. Notice that there is no cost associated with the “north of” operation as it is just a comparison of two numbers.

## 6 Experimental Study

The system was tested on the red sign layer of the  $GT_3$  map of Finland. This map is one of a series of 19 GT maps that cover the whole area of Finland. The red sign layer contains geographic symbols that mostly denote tourist sites. The map was scanned at 240dpi. The layer was split into 425 tiles of size  $512 \times 512$ . For the purpose of our experiment, each one of these tiles was considered as an image. See Figure 5 for an example image. The initial training set was created by giving one example symbol of each class as taken from the legend of the map. There were 22 classes in the map. The images were input in random order to the image database via the image conversion system outlined in Section 3. The first 50 tiles were processed in user verification mode. At that point, the training set contained 100 instances of symbols and the current recognition rate was determined sufficient. The remaining images were processed automatically. The results of this conversion (i.e.,

$$C_{2B_I} = 3c_{af} + 2c_r + N_{pic} \times [c_r + c_{sf} + N_{InR_2} \times (c_r + c_{sn}) + N_{sv\_InR_2} \times c_d + N_{rslt_2} \times (c_{af} + c_r) + c_{an}] \quad (5)$$

$$C_{2B_P} = 2c_{af} + 2c_r + 2c_{nc} + N_{pic} \times [c_s + c_{sf} + N_{sv\_InR_2} \times (c_r + c_d + c_{sn}) + N_{rslt_2} \times (c_{af} + c_r)] \quad (6)$$

The main difference between these two cost estimates is in the number of spatial next operations and the number of random access operations. In the integrated organization, all tuples  $\mathbf{t}$  of any class in rectangle  $R_2$  are retrieved from the spatial index. The `class` of  $\mathbf{t}$  is then retrieved from the `logical_images` relation to see if it corresponds to a “scenic view”. This requires a random access operation for each tuple in  $R_2$ . In addition, the result of the spatial query is larger. In the partitioned organization, only tuples of type “scenic view” are retrieved by the spatial index. Thus, there is no need for an additional random access to check the class of the tuple, and the result of the spatial query is smaller. The plan for the partitioned organization can be further improved by applying a *spatial join* operation to the two relations `scenic_class` and `pic_class` corresponding to “scenic view” and “picnic”, respectively. This operation uses techniques aimed at computing joins between collections of objects based on attributes corresponding to the space occupied by the individual object. The overall idea is that the join can be efficiently computed by traversing both indices in parallel in such a way as to avoid comparing tuples which cannot satisfy the join condition. This operation has not been implemented in SAND yet. Once it is added, plan  $P2B_P$  will be revised accordingly.

It is interesting to compare the costs of answering query 2 for one particular organization using the two different plans. For the integrated organization, we compare equations 3 and 5. In plan A, both relations are scanned sequentially via the alphanumeric index `li_cl`. For each picnic tuple, each scenic view tuple is checked to determine whether or not it is within the specified range. Thus, the total number of distance computations is  $N_{pic} \times N_{sv}$ . In addition, the same number of random access operations are also required in order to get the locations from the `logical_images` relations. In plan B, the spatial index is used and thus only tuples that are within the specified range need to be examined. The cost of this is the overhead involved in using the spatial index. In this case, this cost is  $N_{pic}$  spatial find operations, and  $N_{pic} \times N_{InR_2}$  spatial next operations. The total number of distance computations is  $N_{pic} \times N_{sv\_InR_2}$ . Whether plan A or plan B is better depends on the size of the data set and on the size of the specified spatial range. Assuming a relatively large data set and a relatively small spatial range (i.e., that the number of tuples in the spatial range is much smaller than the total number of tuples in the data set), plan B should prove to be much more efficient than plan A. Similar observations can be made about the partitioned organization by comparing equations 4 and 6. Once the spatial join operation is implemented, as mentioned above, the difference will be even larger.

Equations 7 and 8 estimate the cost of responding to query 3 using the integrated and partitioned organizations, respectively. Note that the plan for this query is very similar to plan B for query 2 in the case of both the integrated and partitioned data organizations. A plan similar to that of plan A for query 2 is not reasonable as there is no condition on an alphanumeric attribute. In other words, the only reasonable plan is to search using the spatial index.  $N_{InR_3}$  denotes the number of tuples in the search rectangles of query 3 ( $R_3$ ).  $N_{InC_3}$  denotes the number of tuples in the circular range specified in query 3 ( $C_3$ ).  $N_{rslt_3}$  denotes those tuples of  $N_{InC_3}$  that are in the same image as the

<i>Name</i>	<i>Meaning</i>
$c_s$	average time to access a tuple in sequential order
$c_r$	average time to access a tuple by tid (random order)
$c_{af}$	average time to perform a “find first” operation on an alphanumeric index
$c_{an}$	average time to perform a “find next” operation on an alphanumeric index
$c_{sf}$	average time to perform a “find first in window” operation on a spatial index
$c_{sn}$	average time to perform a “find next in window” operation on a spatial index
$c_d$	average time to perform a distance computation
$c_{nc}$	average time to perform class name to relation name conversion

Figure 12: List of constants used in the cost analysis.

Equations 1 and 2 estimate the cost of responding to query 1 using the integrated and partitioned organizations, respectively. The costs are computed based on the detailed plans given in Appendix A.

$$C_{1_I} = 2c_{af} + c_r + N_{sv} \times (2c_r + c_{af} + c_{an}) \quad (1)$$

$$C_{1_P} = c_{af} + c_r + c_{nc} + N_{sv} \times (c_s + c_{af} + c_r) \quad (2)$$

One difference between these two cost estimates is that in the integrated organization, there is one more alphanumeric find operation than in the partitioned organization. It is required in order to find the first scenic view tuple in the index. In addition, one more random access is required for each scenic view tuple in order to get the `image_id` from the `logical_images` relation. The other difference is that there are  $N_{sv}$  alphanumeric next operations in the integrated organization compared with  $N_{sv}$  sequential access operations in the partitioned organizations. The reason for this is that in the partitioned organization, the relation is scanned directly, whereas in the integrated organization the index is scanned.

Equations 3 and 4 estimate the cost of responding to query 2 with plan A using the integrated and partitioned organizations, respectively.  $N_{InR_2}$  denotes the number of tuples in the search rectangles of query 2 ( $R_2$ ).  $N_{sv\_InR_2}$  denotes the number scenic view tuples in  $R_2$ .  $N_{sv\_InC_2}$  denotes the number of scenic view tuples in the circular range specified in query 2 ( $C_2$ ).  $N_{rslt_2}$  denotes those tuples of  $N_{sv\_InC_2}$  that are in the same image as the picnic site whose location was the center of this circular range. This is the number of results reported when responding to query 2.

$$C_{2_{A_I}} = 3c_{af} + 2c_r + N_{pic} \times [c_r + c_{af} + N_{sv} \times (c_r + c_d + c_{an}) + N_{rslt_2} \times (c_{af} + c_r) + c_{an}] \quad (3)$$

$$C_{2_{A_P}} = 2c_{af} + 2c_r + 2c_{nc} + N_{pic} \times [c_s + N_{sv} \times (c_s + c_d) + N_{rslt_2} \times (c_{af} + c_r)] \quad (4)$$

The main difference between these two cost estimates is that in the integrated organization the index is scanned sequentially, whereas in the partitioned organization the relation corresponding to the scenic view partition is scanned sequentially (as is the case of query 1). As a result, once more there are considerably more random access operations in the integrated organization than in the partitioned organization.

Equations 5 and 6 estimate the cost of responding to query 2 with plan B using the integrated and partitioned organizations, respectively.

get all tuples of `logical_images` which correspond to "site of interest"  
 (use index `li_cl`)  
 for each such tuple `t`  
   get all points in a  $4 \times 4$  mile rectangular region around `t.location`  
   for each one of these points `p`  
     if `p` is within 2 miles of the "site of interest" point and in the same image then  
       display the corresponding physical image and output the class name of the point

**Plan P3<sub>P</sub>** Search the site of interest partition sequentially, search all other partitions using the spatial index on `location`.

for each partition `p`  
   for each tuple `t1` of relation corresponding to "site of interest" partition  
     get all points in a  $4 \times 4$  mile rectangular region around `t1.location` in partition `p`  
     for each one of these points `p`  
       if `p` is within 2 miles of the "site of interest" point and in the same image then  
         display the corresponding physical image and output the class name of the point

**Query Q4:** display all images that contain an airfield north of a beach

**Plan P4<sub>I</sub>** Search using alphanumeric indices on `class` for all airfield tuples and all beach tuples.

get all tuples of `logical_images` which correspond to "airfield" (use index `li_cl`)  
 for each such tuple `t1`  
   get all tuples of `logical_images` which correspond to "beach" (use index `li_cl`)  
   for each such tuple `t2`  
     if `t1`'s `y` coordinate > `t2`'s `y` coordinate and they are in the same image then  
       display the corresponding physical image

**Plan P4<sub>P</sub>** Search the airfield and the beach partitions sequentially .

for each tuple `t1` of relation corresponding to "airfield" partition  
   for each tuple `t2` of relation corresponding to "beach" partition  
     if `t1`'s `y` coordinate > `t2`'s `y` coordinate and they are in the same image then  
       display the corresponding physical image

### 5.3 Cost Analysis

In order to estimate the costs of each plan, we must make some assumptions about the data distribution and the costs of the various operations. Table 12 contains a list of basic operations that are used in processing queries, along with a constant value that denotes the the cost of this operation. Notice that the cost of the "display" operation is not included since it is not considered part of processing the query. It is only a mechanism to output the answer to the query and is always the same regardless of the selected execution plan. Let  $N_{pic}$ ,  $N_{sv}$ ,  $N_{soi}$ ,  $N_{air}$ , and  $N_{bch}$  denote the number of tuples for which `semantics` is "picnic", "scenic view", "site of interest", "airfield", and "beach", respectively.

```

get all tuples of logical_images which correspond to "picnic" (use index li-cl)
for each such tuple t1
    get all tuples of logical_images which correspond to "scenic view"
    (use index li-cl)
    for each such tuple t2
        if distance between t1 and t2 < 5 miles and they are in the same image then
            display corresponding physical image

```

**Plan P2B<sub>I</sub>** Search for picnic tuples using an alphanumeric index on `class` and search for scenic view tuples using a spatial index on `location`.

```

get all tuples of logical_images which correspond to "picnic" (use index li-cl)
for each such tuple t
    get all points in a 10 × 10 mile rectangular region around t.location
    for each one of these points p
        if p is a "scenic view" and within 5 miles of "picnic" point and in same image then
            display the corresponding physical image

```

**Plan P2A<sub>P</sub>** Search both the picnic and scenic view partitions sequentially.

```

for each tuple t1 of relation corresponding to "picnic" partition
    for each tuple t2 of relation corresponding to "scenic view" partition
        if distance between t1.location and t2.location < 5 miles
            and they are in the same image then
                display the corresponding physical image

```

**Plan P2B<sub>P</sub>** Search the picnic partition sequentially, and search the scenic view partition using the spatial index on `location`.

```

for each tuple t1 of relation corresponding to "picnic" partition
    get all points in a 10 × 10 mile rectangular region around t.location
    in the relation corresponding to "scenic view" partition
    for each one of these points p
        if p is within 5 miles of the "picnic" point and in the same image then
            display corresponding physical image

```

Notice that plan P2B, that takes advantage of the spatial index, uses the minimum bounding square for the circular region of interest to specify the desired range to the spatial index. However, not all tuples in this square will necessarily intersect the circular region. Thus, after retrieving all points in the specified square region, we still need to check if the point is within the required distance. This is necessary because the builtin primitives supported by the spatial index used here are designed to return all items that overlap a given rectangular window. If we had a primitive that enabled retrieving a point based on its Euclidean distance from another point, then the extra check may not be necessary. Such a primitive is currently under construction.

**Query Q3:** display all images with a site of interest, and output the semantics of anything within 2 miles of these sites of interest.

**Plan P3<sub>I</sub>** Search for site of interest tuples using an alphanumeric index on `class` and search for points in given range using a spatial index on `location`.

```

display PI.raw
  from logical_images LI1, logical_images LI2, classes C1,
       classes C2, physical_images PI
  where C1.semantics = "airfield" and C2.semantics = "beach"
        and C1.name = LI1.class and C2.name = LI2.class
        and north(LI1.location,LI2.location)
        and LI1.image_id = LI2.image_id and LI1.image_id = PI.image_id;

```

```

display PI.raw
  from air_class AC, beach_class BC, physical_images PI
  where north(AC.location,BC.location)
        and AC.image_id = BC.image_id and AC.image_id = PI.image_id;

```

The function `north` takes two geometric objects (such as two points in the example above) and returns true if the first object is north of the second object.

## 5.2 Query Processing

The following plans outline how responses to queries Q1–Q4 are computed using the two data organizations. These plans utilize the indexing structures available for each organization. Indices on alphanumeric attributes are capable of locating the closest value greater than or equal to a given string or number. Indices on spatial attributes are capable of returning all items that overlap (wholly or partially) a given rectangular window. Direct addressing of a tuple within a relation is possible by means of a tuple identifier (or *tid* for short). All index structures have an implicit attribute that stores this tid. The  $X^{th}$  plan, labeled  $Px_I$ , uses the integrated organization. The  $X^{th}$  plan, labeled  $Px_P$ , uses the partitioned organization. Appendix A contains detailed plans for these queries.

**Query Q1:** display all images that contain a scenic view.

**Plan P1<sub>I</sub>:** Search using an alphanumeric index on `class`.

```

  Get all tuples of logical_images which correspond to "scenic view" (use index li-cl)
  For each such tuple t
    display the physical image corresponding to t

```

**Plan P1<sub>P</sub>** Search the `scenic view` partition sequentially

```

  For each tuple t of relation corresponding to "scenic view" partition
    display the physical image corresponding to t

```

**Query Q2:** display all images that contain a scenic view within 5 miles of a picnic site.

We suggest two different plans for this query. The first uses only alphanumeric indices, while the second uses an alphanumeric index and a spatial index.

**Plan P2A<sub>I</sub>** Search picnic tuples and scenic view tuples using the alphanumeric index on `class`. For each picnic tuple, check all scenic view tuples to determine which ones are within the specified distance.

name of a relation given the class name. `get_class` returns the class name given a relation name. Thus, there is no need for the user to know the names assigned by the system to these relations.

**Query Q2:** display all images that contain a scenic view within 5 miles of a picnic site.

```
display PI.raw
  from logical_images LI1, logical_images LI2, classes C1,
       classes C2, physical_images PI
  where C1.semantics = "scenic view" and C2.semantics = "picnic site"
        and C1.name = LI1.class and C2.name = LI2.class
        and distance(LI1.location,LI2.location) < 5
        and LI1.image_id = LI2.image_id and LI1.image_id = PI.image_id;
```

```
display PI.raw
  from scenic_class SC, pi_class PIC, physical_images PI
  where distance(SC.location,PIC.location) < 5
        and SC.image_id = PIC.image_id and PIC.image_id = PI.image_id;
```

The function `distance` takes two geometric objects (such as two points in the example above) and returns a floating point number representing the Euclidean distance between them.

**Query Q3:** display all images with a site of interest and output the semantics of anything within 2 miles of these sites of interest.

```
display PI.raw C2.semantics
  from logical_images LI1, logical_images LI2, classes C1,
       classes C2, physical_images PI
  where C1.semantics = "site of interest" and C1.name = LI1.class
        and distance(LI1.location,LI2.location) < 2
        and LI1.image_id = LI2.image_id and LI1.image_id = PI.image_id
        and C2.name = LI2.class;
```

```
display PI.raw C.semantics
  from star_class SI, physical_images PI, X_class CL, classes C
  where distance(SI.location,CL.location) < 2
        and SI.image_id = CL.image_id and SI.image_id = PI.image_id
        and C.name=CL.class;
```

`X_class` is a variable that is assigned all possible relation names as derived from the `class` field of the tuples of relation `classes`. This is done by cycling through the `classes` relation and repeating this query using the partition that corresponds to each class.

**Query Q4:** display all images that contain an airfield north of a beach.



```

create index CL_cert for CL_class (certainty);
create index CL_loc for CL_class (location);

```

Indices `cl_semant`, `cl_name`, `pi_imid`, and `pi_ll` are identical to those in the integrated organization. Each instance of the `class` relation has an alphanumeric index on `certainty` and a spatial index on `location`. The spatial index is used to deal with queries of the type “find all images with sites of interest within 10 miles of a picnic area” by means of a spatial join operator. Figure 11 illustrates the file structures for the partitioned organization corresponding to file structures used for text data.

## 5 Retrieving Images by Contents

As mentioned above, we distinguish between contextual information and spatial information found in images. Similarly, we distinguish between query specifications that are purely contextual and those that also contain spatial conditions. A *contextual specification* defines the images to be retrieved in terms of their contextual information (i.e., the objects found in the image). For example, suppose we want to find all images that contain fishing sites or campgrounds. A *spatial specification* further constrains the required images by adding conditions regarding spatial information (i.e., the spatial relations between the objects).

In order to describe the methods that we use for retrieving images by contents, we first present some example queries. Next, we demonstrate the strategies used to process these queries. We conclude by analyzing the expected costs of these strategies (termed *plans*) and compare the two proposed data organizations (i.e., integrated and partitioned).

### 5.1 Example Queries

The example queries in this section are first specified using natural language. This is followed by two equivalent SQL-like queries. The first assumes an integrated organization and the second assumes a partitioned organization.

**Query Q1:** display all images that contain a scenic view .

```

display PI.raw
  from logical_images LI, classes C, physical_images PI
  where C.semantics = "scenic view" and C.name = LI.class
        and LI.image_id = PI.image_id;

```

```

display PI.raw
  from scenic_class SC, physical_images PI
  where SC.image_id = PI.image_id;

```

Notice that in order to write SQL-like queries for the partitioned organization, the names of the relations corresponding to each partition must be known. This can easily be overcome by having the system assign names to these relations. These names are derived from the `class` attribute of relation `classes`. Two functions that perform this name conversion are provided. `get_rel_name` returns the

star-class:	image_id	certainty	location
	image_1	0.99	(6332,1586)
	image_1	0.99	(6540,1712)
	image_1	1	(6474,1814)

P-class:	image_id	certainty	location
	image_1	0.99	(6161,1546)
	image_1	0.99	(6432,1622)
	image_2	0.99	(6858,1771)

scenic_class:	image_id	certainty	location
	image_1	0.99	(6630,1662)
	image_2	0.72	(6803,1565)

pi-class	image_id	certainty	location
	image_1	0.99	(6395,1741)
	image_2	0.99	(6849,1756)
	image_2	0.99	(6800,1807)

Figure 10: Examples instances of relations star\_class, P\_class, scenic\_class, and pi\_class in the map domain. The tuples correspond to the symbols in the images of Figures 5 and 6.

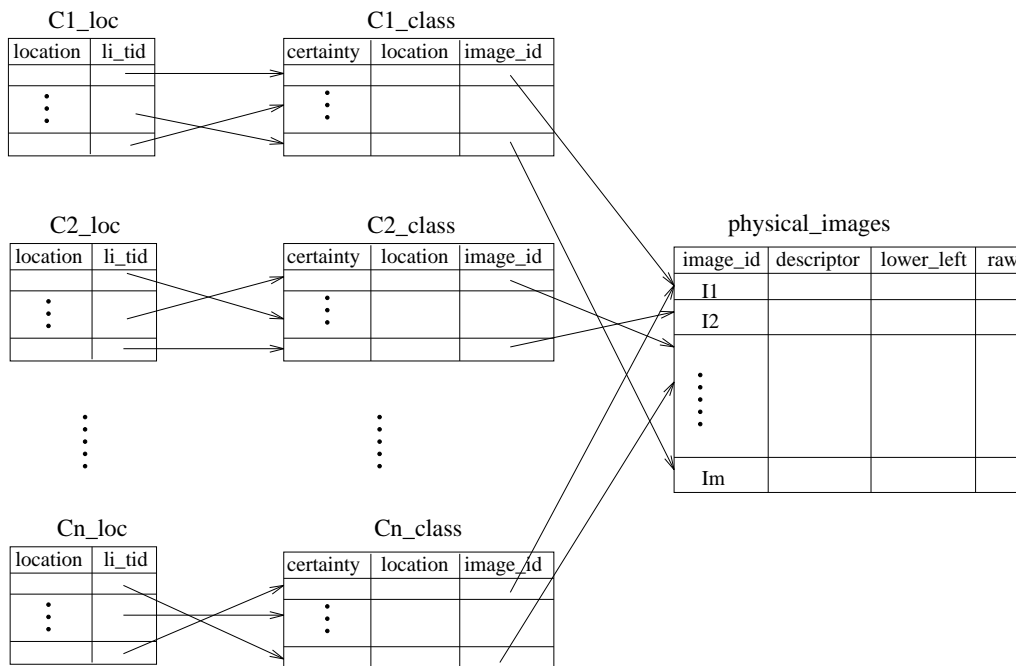


Figure 11: File structures for logical and physical images using the partitioned organization.

## Constructing Indices

Indices are defined on the schemas of the partitioned organization as follows (in SQL-like notation):

```

create index cl_semant for classes (semantics);
create index cl_name for classes (name);
create index pi_imid for physical_images (image_id);
create index pi_ll for physical_images (lower_left);
for each class CL in application

```

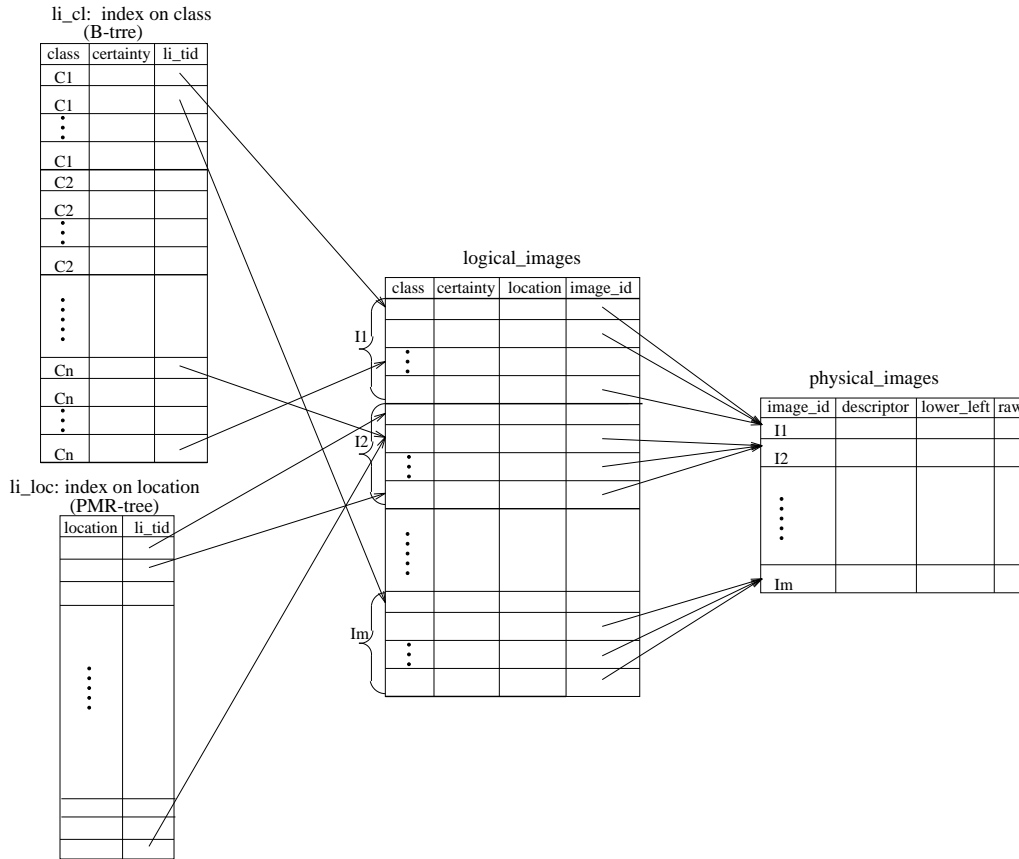


Figure 8: File structures for logical and physical images using the integrated organization.

## 4.2 Partitioned Organization

```

(create table classes      (create table physical_images   for each class CL in application
name CHAR[30],           image_id INTEGER,              (create table CL_class
semantics CHAR[50],      descriptor CHAR[50],          image_id INTEGER,
bitmap IMAGE);          lower_left POINT,           certainty FLOAT,
                        raw IMAGE);      location POINT);

```

Figure 9: schemas for the relations classes, physical\_images, and CL\_class.

The schema definitions given in Figure 9 define the relations in the partitioned organization. Both the `classes` and `physical_images` definitions are identical to those in the integrated organization. The only difference between these two organizations is in the way the logical images are stored. In the partitioned organization, there is one relation, `CL_class` for each class  $CL$  in the application. Each relation `CL_class` contains the logical images tuples  $(C, certainty, (x, y))$  for which  $C = CL$ . This is equivalent to the result of a selection operation, `class = CL`, on relation `logical_images`. See Figure 10 for example instances of relations `star_class`, `P_class`, `scenic_class`, and `pi_class` in the map domain for the images given in Figures 5 and 6.

image_id	class	certainty	location
image_1	M	1	(6493,1544)
image_1	P	0.99	(6161,1546)
image_1	H	0.99	(6513,1566)
image_1	U	1	(6167,1583)
image_1	star	0.99	(6332,1586)
image_1	P	0.99	(6432,1622)
image_1	K	1	(6416,1636)
image_1	fish	1	(6411,1661)
image_1	scenic	0.99	(6630,1662)
image_1	square	1	(6422,1693)
image_1	star	0.99	(6540,1712)
image_1	pi	0.99	(6396,1741)
image_1	triangle	1	(6475,1784)
image_1	star	1	(6474,1814)
image_1	cross	0.79	(6291,1854)
image_1	box	0.74	(6357,1862)
image_1	inf	1	(6226,1937)
image_1	box	1	(6280,2011)
image_2	arrow	0.99	(6861,1544)
image_2	scenic	0.72	(6803,1565)
image_2	pi	0.99	(6849,1756)
image_2	R	0.71	(6849,1756)
image_2	P	0.99	(6858,1771)
image_2	H	0.99	(6827,1775)
image_2	U	0.79	(6827,1775)
image_2	pi	0.99	(6800,1807)
image_2	R	0.99	(6800,1807)

Figure 7: Example instance for the `logical_images` relation in the map domain. The tuples correspond to the symbols in the images of Figures 5 and 6.

alphanumeric index. It is used to search the `logical_images` relation by `class`. It has a secondary index on attribute `certainty`. Thus, tuples that have the same class name are ordered by certainty value within this index. `li_loc` is a spatial index on points. It is used to search the `logical_images` relation by location (i.e., to deal with spatial queries regarding the locations of the symbols in the images such as distance and range queries). The spatial indices are implemented by a PMR quadtree for points [12].

Observe that the file structures resulting from the integrated organization are very similar to the file structures that are used by inverted file methods for storing text [6]. An inverted file consists of two structures. A vocabulary list which is a sorted list of words found in the documents, and a posting file that indicates for each word the list of documents that contain this word along with information regarding the position of the word in the document. The vocabulary list is actually an index on the posting file, and is used to locate the record of the posting file corresponding to a given word on disk. In our organization, the `logical_images` relation corresponds to the posting file. The index `li_cl` on this relation serves the purpose of the vocabulary list. The main difference is that since we are dealing with 2-dimensional information rather than the 1-dimensional information inherent in text databases, we need more elaborate methods to store and index the locational information. In particular, just storing the location, as is usually done for text data, is insufficient. Figure 8 illustrates the file structures that are used for the integrated organization corresponding to similar file structures used for text data.

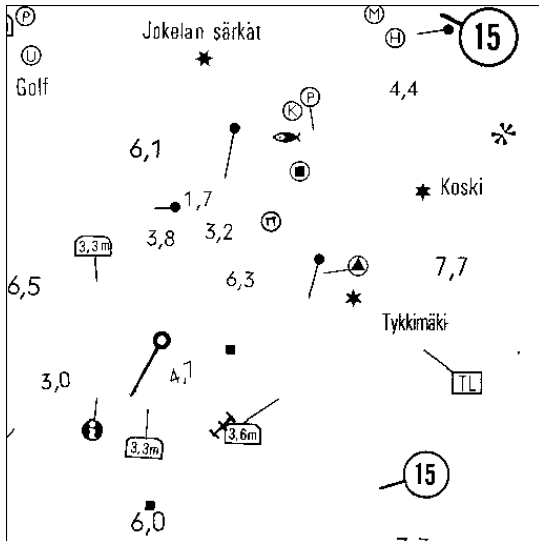


Figure 5: Example image 1 (image\_1).

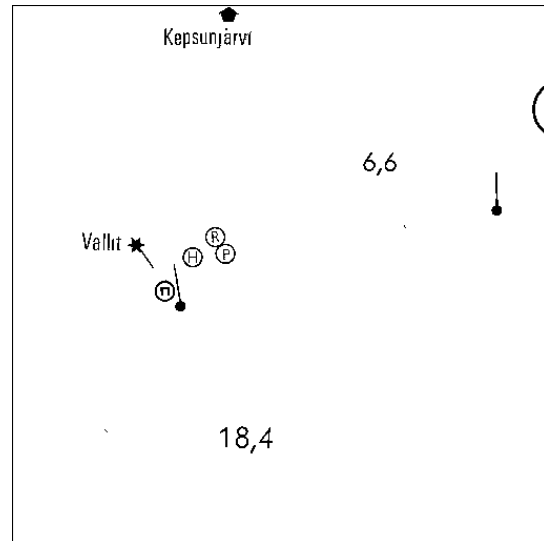


Figure 6: Example image 2 (image\_2).

example instance of the `physical_images` relation in the map domain.

The `logical_images` relation stores the logical representation of the images. It has one tuple for each candidate class output by the image input system for each valid symbol  $s$  in each image  $I$ . The tuple has four fields. The `image_id` field is the integer identifier given to  $I$  when it was inserted into the database. It is the same as the `image_id` field of the tuple representing  $I$  in the `physical_images` relation. The `class` and `certainty` fields store the name of the class  $C$  to which the image input system classified  $s$  and the certainty that  $s \in C$ . The `location` field stores the  $(x, y)$  coordinate values of the center of gravity of  $s$  relative to the entire image. See Figure 7 for an example instance of the `logical_images` relation in the map domain for the images given in Figures 5 and 6.

## Constructing Indices

Indices are defined on the schemas of the integrated organization as follows (in SQL-like notation):

```

create index cl_semant for classes (semantics);
create index cl_name for classes (name);
create index pi_imid for physical_images (image_id);
create index pi_ll for physical_images (lower_left);
create index li_cl for logical_images (class certainty);
create index li_loc for logical_images (location);

```

`cl_semant` and `cl_name` are alphanumeric indices. They are used to search the `classes` relation by `semantics` and `name`, respectively. The `pi_imid` index is also alphanumeric. It is used to search the `physical_images` relation by `image_id`. `pi_ll` is a spatial index on points. It is used to search the `physical_images` relation by the coordinates of the lower left corner of the images. `li-cl` is an























class	semantics	bitmap
S	harbor	
square	hotel	
scenic	scenic view	
T	customs	
R	restaurant	
P	post office	
M	museum	
K	cafe	
waves	beach	
triangle	camping site	
B	filling station	
arrow	holiday camp	
cross	first aid station	
fish	fishing site	
H	service station	
inf	tourist information	
pi	picnic site	
air	airfield	
star	site of interest	
telephone	public telephone	
box	youth hostel	
U	sports institution	

Figure 3: Example instance for the `classes` relation in the map domain.

image_id	descriptor	raw	lower_left
image_1	tile 003.012 of Finish road map	Figure 5	(6144,1536)
image_2	tile 003.013 of Finish road map	Figure 6	(6656,1536)

Figure 4: Example instance for the `physical_images` relation in the map domain.

`classes` relation is populated using the same data that is used to create the initial training set for the image input system (i.e., one example symbol for each class that may be present in the application along with its name and semantic meaning). See Figure 3 for an example instance of the `classes` relation in the map domain.

The `physical_images` relation has one tuple per image  $I$  in the database. The `image_id` field is an integer identifier given to the image  $I$  when it is inserted into the database. The `descriptor` field stores an alphanumeric description of the image  $I$  that the user gives when inserting  $I$ . The `raw` field stores the actual image  $I$  in its physical representation. It is an attribute of type `IMAGE`. The `lower_left` field stores an offset value that locates the lower left corner of image  $I$  with respect to the lower left corner of some larger image  $J$ . This is useful when a large image  $J$  is tiled, as in our example map domain. This offset value, in addition to the location of the symbol  $s$  in one of the tile images  $I$ , will give the absolute location of  $s$  in the large image  $J$ . It is an attribute of type `POINT`. Any other information that the user may wish to store about the images such as how they were formed, camera angles, scale, etc. can be added as fields of this relation. See Figure 4 for an

recognition rate achieved is deemed adequate. At that point, the system can continue to process the input images automatically.

The output of applying the conversion process to  $I_{phys}$  is a logical image where the tuples are of the form  $(C, certainty, (x, y))$  where  $C \neq undefined$ ,  $0 < certainty \leq 1$  indicating the certainty that  $s \in C$ , and  $(x, y)$  is the location of  $s$  in  $I_{phys}$ . For each image, a set of such tuples is inserted into a spatial database as described in the following section. In addition, the raw image  $I_{phys}$  (i.e., the image in its physical representation) is also stored.

## 4 Image Storage

Images and other information pertaining to the application are stored in relational tables. The database system that we use for this purpose is SAND [1] (denoting spatial and non-spatial database) developed at the University of Maryland. It is a home-grown extension to a relational database where the tuples may correspond to geometric entities such as points, lines, polygons, etc. having attributes which may be both of a locational (i.e., spatial) and non-locational nature. Both types of attributes may be designated as indices of the relation. For indices built on locational attributes, SAND makes use of suitable spatial data structures. Attributes of type image are used to store physical images. Query processing and optimization is performed following the same guidelines of relational databases extended with a suitable cost model for accessing spatial indices and performing spatial operations.

We propose two different data organizations for storing the images in relational tables. These two organizations differ in the way that the logical images are stored. In the *integrated organization*, all of the tuples of the logical images are stored in one relation. In the *partitioned organization*, the tuples are partitioned into separate relations resulting in a one-to-one correspondence between relations and classes of the application. For example, tuples  $(C, certainty, (x, y))$  of a logical image for which  $C = C_1$  are stored in a relation corresponding to  $C_1$ . The motivation for the partitioned organization is that it enables efficient use of spatial indices while processing spatial queries by using a spatial join operator (e.g., [17]). For more details, see Section 5.

### 4.1 Integrated Organization

```
(create table classes      (create table physical_images      (create table logical_images
name CHAR[30],            image_id INTEGER,                image_id INTEGER,
semantics CHAR[50],       descriptor CHAR[50],            class CHAR[30],
bitmap IMAGE);           lower_left POINT,               certainty FLOAT,
                           raw IMAGE);                     location POINT);
```

Figure 2: schemas for the relations `classes`, `physical_images`, and `logical_images`.

The schema definitions given in Figure 2 define the relations in the integrated organization. We use an SQL-like syntax. The `classes` relation has one tuple for each possible class in the application. The `name` field stores the name of the class (e.g., `star`), the `semantics` field stores the semantic meaning of the class in this application (e.g., `site of interest`). The `bitmap` field stores a bitmap of an instance of a symbol representing this class. It is an attribute of type `IMAGE`. The

denoted by  $I_{phys}$ . In the *logical image* representation, an image  $I$  is represented by a list of tuples, one for each symbol  $s \in I$ . The tuples are of the form:  $(C, certainty, (x, y))$  where  $C \neq undefined$ ,  $(x, y)$  is the location of  $s$  in  $I$ , and  $0 < certainty \leq 1$  indicates the certainty that  $s \in C$ .

### 3 Image Input

Conversion of input images from their physical to their logical representation is performed using methods that are common in the field of document analysis [16]. These methods use various pattern recognition techniques that assign a physical object or an event to one of several pre-specified classes. Patterns are recognized based on some features or measurements made on the pattern. A library of features and their classifications, termed the *training set library*, is used to assign candidate classifications to an input pattern according to some distance metric. Each candidate classification is given a certainty value that approximates the certainty of the correctness of this classification.

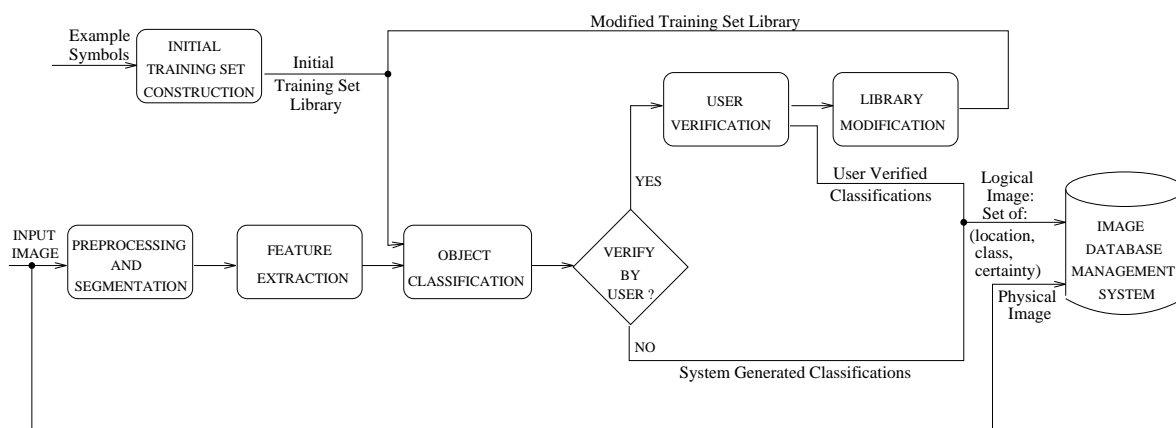


Figure 1: Image input system

We have adapted these methods to solve the problem of converting symbolic images from a physical to logical representation. Figure 1 is a block diagram of the image input system that we have developed for this purpose. It is driven by the symbolic information conveyed by the image. That is, rather than trying to interpret everything in the image, it looks for those symbols that are known to be of importance to the application. Any other symbol found in the image is labeled as belonging to the undefined class. We only give a short overview of this system here. See [20] for more details. A symbolic image  $I_{phys}$  is input to the system in its physical representation. It is converted into a logical image by classifying each symbol  $s$  found in  $I_{phys}$  using the training set library. An initial training set library is constructed by giving the system one example symbol for each class that may be present in the application. In the map domain, the legend of the map may be used for this purpose.

The system may work in two modes. In the *user verification mode*, the user verifies the classifications before they are input to the database. The training set is modified to reflect the corrections that the user made for the erroneous classifications. In the *automatic mode*, the classifications are generated by the system and input directly to the database. The user determines the mode in which the system operates. In general, the system should operate in user verification mode until the



conventional database management system (DBMS). In our application, we make use of a relational DBMS although our ideas are applicable to other types of DBMS's as well. These methods offer solutions for both the pattern recognition and indexing aspects of the problem. We describe how to incorporate the results of these methods into an existing spatial database based on the relational model. Our emphasis is on extracting both contextual and spatial information from the raw images. The logical image representation that we define preserves this information. The logical images are stored as tuples in a relation. Indices are constructed on both the contextual and the spatial data, thus enabling efficient retrieval of images based on contextual as well as spatial specifications. It is our view that an image database must be able to process queries that have both contextual and spatial specifications, in addition to any query that a traditional DBMS could answer.

We propose two different data organizations for storing images in relational tables. They differ in the way that the logical images are stored. All of the examples and experiments in this paper are from the map domain. However, images from many other interesting applications fall into the category of symbolic images. These include CAD/CAM, engineering drawings, floor plans, and more. Hence, the methods that we describe in this paper are applicable to them as well.

The rest of this paper is organized as follows. Section 2 presents some definitions as well as the notation used. Section 3 outlines an image input system that is used to convert images from their physical representation to their logical representation as they are input to the database. Section 4 describes how images are stored in a database management system including schema definitions and example relations. Section 5 gives sample queries along with execution plans and cost estimates for these plans. Section 6 describes our experimental study and results along with a quantitative comparison of the two proposed data organizations. Section 7 contains concluding remarks.

## 2 Definitions and Notations

Below we define some terms and the notation used in the remainder of the paper. A *general image* is a two-dimensional array of picture elements (termed *pixels*)  $p_0, p_1, \dots, p_n$ . Each pixel is represented by its  $x$  and  $y$  coordinate values. A *binary image* is a general image where each pixel has one of two possible values (usually 0 and 1). One value is considered the foreground and the other the background. A general image is converted into a binary image by means of a threshold operation. A *symbol* is a group of connected pixels that together have some common semantic meaning. In a given application, symbols will be divided into *valid symbols* and *invalid symbols*. A *valid symbol* is a symbol whose semantic meaning is relevant in the given application. An *invalid symbol* is a symbol whose semantic meaning is irrelevant in the given application. A *class* is a group of symbols all of which have the same semantic meaning. All invalid symbols belong to a special class called the *undefined class*.

A *symbolic image* is a general image  $I$  for which the following conditions hold: 1) Each foreground pixel  $p_i$  in  $I$  belongs to some symbol. 2) The set of possible classes  $C_1, C_2, \dots, C_n$  for the application is finite and is known a priori. 3) Each symbol belongs to some class. 4) There exists a function  $f$  which when given a symbol  $s$  and a class  $C$  returns a value between 0 and 1 indicating the certainty that  $s$  belongs to  $C$ .

Images can be represented in one of two ways. In the *physical image* representation, an image is represented by a two-dimensional array of pixel values. The physical representation of an image is

## 1 Introduction

Images (or pictures) serve as an integral part in many computer applications. Examples of such applications include CAD/CAM (computer aided design and manufacturing) software, document processing, medical imaging, GIS (geographic information systems), computer vision systems, office automation systems, etc. All of these applications store various types of images and require some means of managing them. The emerging field of *image databases* deals with this problem [9]. Several such databases have been described in the literature [4, 5, 10, 13, 14, 18]. One of the major requirements of an image database system is the ability to retrieve images based on queries that describe the contents of the required image(s), termed *retrieval by contents*. An example query is “find all images containing camping sites within 3 miles of fishing sites”.

In order to support retrieval by contents, the images should be interpreted to some degree when they are inserted into the database. This process is referred to as converting an image from a *physical* representation to a *logical* representation. The logical representation may be a textual description of the image, a list of objects found in the image, a collection of features describing the objects in the image, a hierarchical description of the image, etc. It is desirable that the logical representation also preserve the spatial information inherent in the image (i.e., the spatial relation between the objects found in the image). We refer to the information regarding the objects found in an image as *contextual information*, and to the information regarding the spatial relation between these objects as *spatial information*. Both the logical and the physical representation of the image are usually stored in the database. An index mechanism based on the logical representation can then be used to retrieve images based on both contextual and spatial information in an efficient way.

In many commercial image database systems, the user provides the logical representation in the form of keywords. These keywords are used to search the database for specific images based on their contents using traditional alphanumeric search techniques. There has been some work in recent years on automatic conversion of images from a physical representation to a logical representation. This is also referred to as *automatic image indexing* [2]. Most of this work has concentrated on the pattern recognition aspect of the problem [7] and on constructing efficient indexing structures [3, 8, 11]. However, it is not clear how to integrate the results of these efforts into a standard database system that can also support the traditional database operations in addition to those specific to images. Furthermore, most of these techniques do not preserve all of the spatial information found in the images.

In our work, we have chosen to focus on images in which the set of objects that may appear are known a priori. In addition, the geometric shapes of these objects are relatively primitive and they convey symbolic information. For example, in the map domain, the information found is mainly symbolic rather than an accurate graphical description of the region covered by the map. Thus, the width of a line representing a road has little to do with its true width. Instead, most often it is determined by the nature of the road (i.e., highway, freeway, rural road, etc.). Many graphical symbols are used to indicate the location of various sites such as hospitals, post offices, recreation areas, scenic areas etc. We call this class of images *symbolic images*. Other similar terms found in the literature are *graphical documents*, *technical documents*, and *line drawings*. By limiting ourselves to symbolic images, the task of object recognition is simplified, and we can utilize well-known methods used in document processing.

In this paper, we present methods for integrating symbolic images into the framework of a

## Integrating Images into a Relational Database System <sup>1</sup>

Hanan Samet

Aya Soffer

Computer Science Department and  
Center for Automation Research and  
Institute for Advanced Computer Science  
University of Maryland at College Park  
College Park, Maryland 20742

E-mail: hjs@umiacs.umd.edu, aya@umiacs.umd.edu

Telephone: (301) 405-1755 Fax: (301) 314-9115

### Abstract

A method is presented for integrating images into the framework of a conventional database management system (DBMS). It is applicable to a class of images termed *symbolic images* in which the set of objects that may appear are known a priori. The geometric shapes of the objects are relatively primitive and they convey symbolic information. Both the pattern recognition and indexing aspects of the problem are addressed. The emphasis is on extracting both contextual and spatial information from the raw images. A logical image representation that preserves this information is defined. Methods for storing and indexing logical images as tuples in a relation are presented. Indices are constructed for both the contextual and the spatial data, thereby enabling efficient retrieval of images based on contextual as well as spatial specifications. Two different data organizations (integrated and partitioned) for storing logical images in relational tables are proposed. They differ in the way that the logical images are stored. Sample queries and execution plans to respond to these queries are described for both organizations. Analytical as well as empirical cost analyses of these execution plans are given. A quantitative comparison of the two data organizations is presented.

**Categories:** image databases, spatial databases, image indexing, query optimization, retrieval by contents

---

<sup>1</sup>The support of the National Science Foundation under Grant IRI-9017393, and the National Aeronautics and Space Administration under Grant NGT-30130 is gratefully acknowledged.