Scalable Database Support for Correlation and Fusion Algorithms

J. E. Barros and J. C. French

Institute for Parallel Computation
Technical Report IPC-93-05
October, 1993

Abstract

The process of extracting useful information from noisy sensor data is an important part of applications such as environmental analysis and military surveillance. Difficulties arise with analyzing the accuracy of the reports and in processing the total number of reports. Automated methods capable of handling noisy information from multiple sensors must be employed to assist the human analyst.

This report investigates scalable database support for such systems. Region estimation and range retrieval techniques are employed to dramatically reduce retrieval and overall execution times.

This work supported in part by JPL Contract no. 957721
and by Dept of Energy Grant no. DE-FG05-88EER25063.

1 Introduction

In applications such as environmental analysis and military surveillance, sensor output is collected, correlated, fused and analyzed to form a representation of an environment. However, it is difficult to analyze the individual reports and to process the large number of reports. Sensor measurements are not exact, may contain significant error, and are consequently difficult to analyze. A large number of reports compounds this problem and the limit of an individual's ability to process many reports is quickly reached. Automated methods capable of handling a large amount of noisy information efficiently must be employed to assist the human analyst. This paper examines scalable database support for such systems.

Sensors operating in an environment can detect and report on only a subset of the entities present. Sensors produce reports on K measured attributes of the present entities. The reports are an ordered pair (X, Σ) where X is a vector of length K and Σ is a $K \times K$ covariance matrix that describes the uncertainty associated with the measurements. For this paper $K = 2$ and corresponds to the spatial position of the entity. Σ can be interpreted as the 95% confidence contour from a bivariate Gaussian whose density function describes the likelihood of the associated entity being at location X. This elliptical contour is completely described by the major axis, minor axis, orientation and position information and is known as the elliptic error probable (EEP). Figure 1 shows some EEPs from one of the simulations.

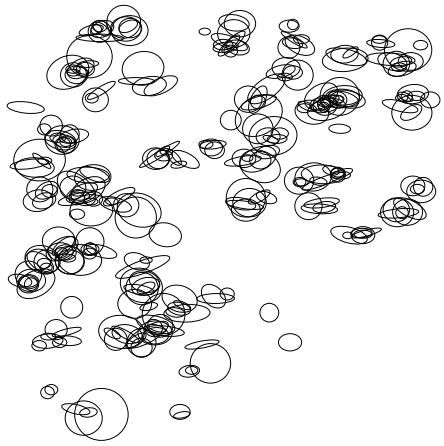

Figure 1: Sensor report elliptic error probables (EEP)

When the system receives a new sensor report it must compare it with the records of each known element. This step, known as correlation, determines to some degree of confidence if the report is about a known entity or is about a new entity. If the report is judged to be about a new entity it is added to the database. If it is judged to be about a report on a previously sensed entity, the report and the record are combined. This process, known as fusion, accepts a group of reports and forms a single entity with a smaller EEP.

Candidates for correlation are selected using a figure of merit for location (fomloc) function. Fomloc tests the hypothesis that two EEPs are reporting on the same location by calculating $e^{-d/2}$ where d is the divergence or the difference in the mean values of the log-likelihood ratio of the two densities[SJB89]. This value is interpreted as the confidence level that the two reports cover the same location. The confidence level is compared against a previously set discriminator, τ, to either accept or reject the hypothesis. If the spatial positions correlate well enough other attributes can be examined to decide whether the two reports are about the same entity. In this report only the entity positions are used in the correlation step.

An important function of the original simulator was to allow experimentation with different τ levels and to evaluate the results produced. The discriminator level, τ, is set at the beginning of the simulation. Normally, the value of τ would be chosen based on prior performance results. Since this research studied execution time and not output quality the particular value of τ was not important.

2 The "A1" Algorithm

There are many possible algorithms for correlation and fusion. For this research we used the algorithm described in [SJB89] known as "A1" (see Figure 2). When a new sensor report arrives, the A1 algorithm searches the candidate data base (CDB) for the set, F, of records with a correlation score greater than the discrimination value τ. The new report is then fused with the single most highly correlated record. The database record used for fusing is then removed from the database. These steps are repeated and a new set F is retrieved until F is empty. The entire process is repeated for each new report. At this point the report (which may or may not have fused with a previously existing record) is inserted in the database. In the pseudocode only the record with the maximum score from the set F is explicitly maintained. This algorithm is linear in the number of records in the database and the number of reports.

We profiled a run of 300 entities, 1000 reports, and multiple τ levels. This is a small run when compared with a run of the desired operational size of 10 thousand to 15 thousand sites. For this scenario approximately half of the cpu time used was spent in calls to fomloc. This percentage will only get worse as the number of entities or the number of reports is increased. To improve performance it is critical to reduce the time spent on correlation.

3 Improving A1 with region estimation

The first approach for improving A1 is to replace the expensive fomloc test with a much less expensive one. A region estimation technique has been developed that reduces the number of points that need to be submitted to the correlation measure. This can be done by quickly selecting a set of points P where $P \supseteq F$. The exact size of P is not critical since it will be processed to yield F. However, P should be as small as possible and ideally $P = F$.

```
S = next sensor report
repeat
    maxm = 0
    for i = 1 to n
        m = fomloc(S,C_i)
        if m > maxm then
            maxm = m
            saved = i
    if maxm ≥ τ
    then
        S = fuse(S,C_saved)
        delete(C_saved)
until maxm < τ
insert(S)
```

Figure 2: A1 correlation algorithm

Given the EEP of a new message and an arbitrary record it is possible to calculate another ellipse representing the greatest distance that can exist between the two and still have a correlation score greater than the discrimination value. A single large EEP can be calculated by using the report database with the greatest uncertainty as reflected in Σ and the EEP of the message. This EEP is centered at the original position of the incoming message and will contain the centers of all the records that need to be considered for that message. The EEP of each record has been considered and can now be ignored.

Since it is easier to test whether a point falls inside a rectangular region than in an elliptical region it is preferable to find a suitable rectangular region. The new calculated EEP can be used as the basis for a rectangular region by considering a bounding rectangle (see Figure 3). The estimation process used here is described in detail in a companion paper [Fre93]. The rectangular region would be centered over the ellipse and its size could be calculated in several ways. The side of a square could be made the same length as the major axis of the ellipse. This could be calculated very quickly and would guarantee that the ellipse is contained in the square. Alternatively, the minimum bounding rectangle could be calculated. The minimum bounding rectangle would be more costly to calculate but would have a smaller area and consequently would enclose fewer entities on average. For this study we chose to use the larger, simpler, and more conservative square. If further performance improvements are required, exploring techniques for quickly calculating the minimum bounding rectangle might be fruitful. Additional gains may also be gotten by removing entities outside of the ellipse but within the corners of the rectangle.

By using the bounding rectangle each record in the database can be quickly tested to see if it falls inside of this rectangular region. Only these points then need to be scored by fomloc and fused as in the original algorithm (see Figure 4). This modified A1 algorithm

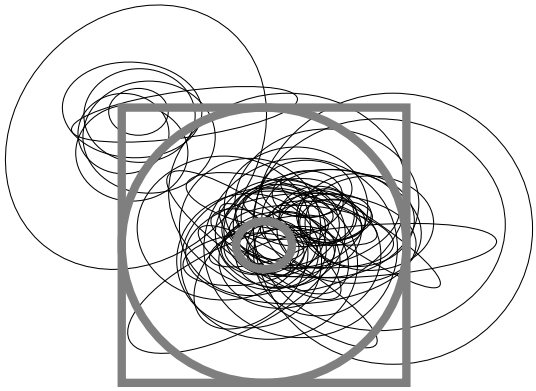

Figure 3: Estimated region

S = next sensor report
repeat
    R = estimate region((x_0,y_0),(x_1,y_1))
    fetch $P = C \in R(x_0 \leq x_c \leq x_1, y_0 \leq y_c \leq y_1)$
    $maxm = 0$
    for $i = 1$ to $|P|$
        m = fomloc(S, C_{P_i})
        if $m > maxm$ then
            $maxm = m$
            $saved = i$
    if $maxm \geq \tau$
    then
        S = fuse(S,C_{saved})
        delete(S,C_{saved})
until $maxm < \tau$
insert(S)

Figure 4: Modified A1 correlation algorithm

is also linear in the number of database records for each message. However, the estimated region inclusion test is significantly less expensive than the correlation measure and reduces the number of correlation measures that need to be performed.

Figure 5 shows the reduction achieved in the number of fomlocs for each message. This graph was generated by counting the number of times fomloc is called for each of a thousand messages. There were roughly 200 sites in the simulation. The counts for the original algorithm (A1) are drawn with points in the graph. These points lie along two curves. This is caused by the system making one, two, or occasionally three passes through the database. Usually, a message fused with a record on the first pass and another pass was needed. Occasionally, the first pass would not find a strong enough correlation so the second pass was not necessary. Much less frequently a message would fuse with more than one record and require a third pass. The results of the modified A1 algorithm are given by the second data set and are drawn with connecting line segments. In this example the modified algorithm needed no more than fourteen fomlocs for any message and less than four fomlocs were needed on average. This results in the modified A1 algorithm being able to finish processing the message sets in approximately one third of the time of the original algorithm.

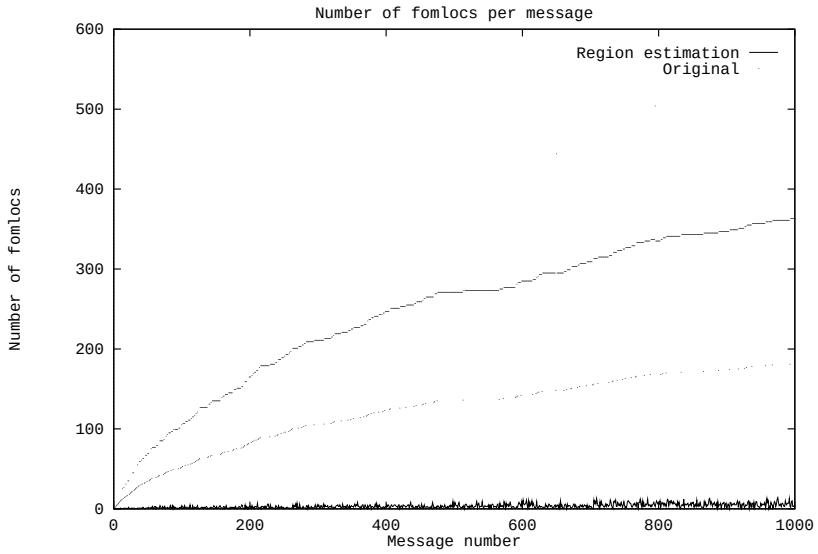

Figure 5: Reduction in number of fomlocs per message. Note original algorithm plotted as points.

In order to ensure that the output of the modified A1 algorithm remains the same the generated representations must be examined. Different correlation algorithms, fusion algorithms and sensor report sets will, in all likelihood, generate different representations for the same environment. A method is needed to evaluate the quality of the different outputs. We use the scoring methods discussed in [SJB89]. The scoring scheme compares the output representation to another representation known to be very accurate. The score generated is

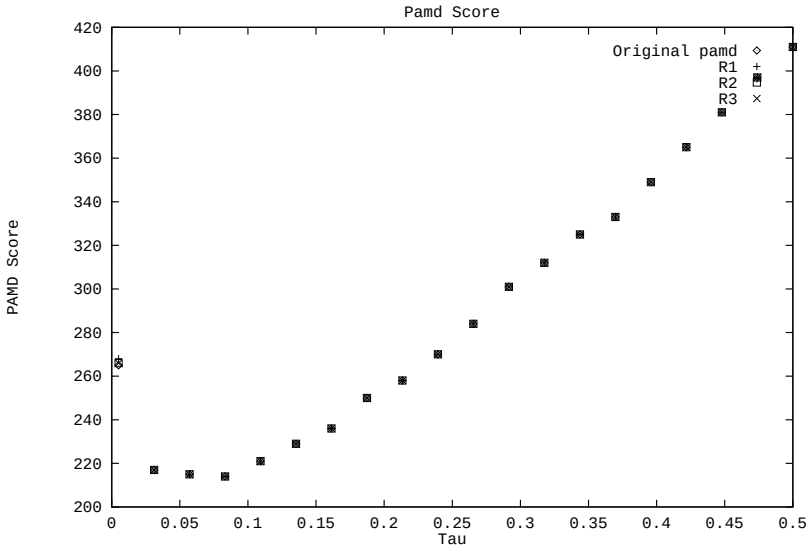

Figure 6: PAMDS are identical except for most liberal τ

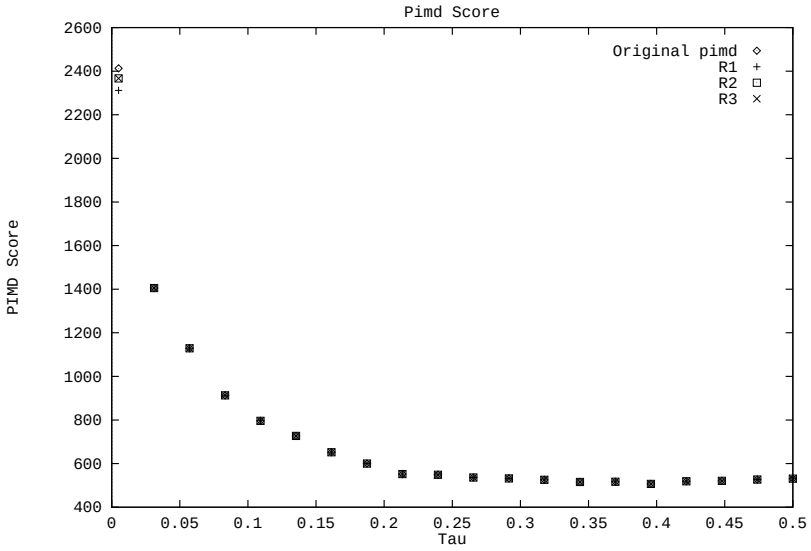

Figure 7: PIMDS are identical except for most liberal τ

based on the distance between the true location of the entities and the reported locations. The representations produced are also penalized for having too many or too few entities. There are two representations known to be accurate that are used to score the results of the algorithms. During the generation of the report sets, the true state of the environment is saved and is known as the *ground truth*. The Perfect Information Minimum Distance Score (PIMDS) compares an output representation and ground truth. During simulation the Perfect Association Representation (PAR) representation generated by processing the report set and making perfect correlations. PAR is, in general, less accurate than the ground truth but is the best possible representation that can be extracted from the given report set. The Perfect Association Minimum Distance Score (PAMDS) compares an output representation and PAR.

The results generated by the original and modified algorithms were compared across different τ levels. As shown in Figures 6 and 7 only the lowest, most lenient, τ level used showed any difference between the original and the three region estimation methods. This difference occurred only because of a shortcoming in the implementation. Our implementation uses a fixed discriminator level. If the implementation were changed to work with variable discriminators we would expect to see no difference in the representation scores. The extremely low discriminator level at run time caused the original algorithm to accept and fuse with records that had not been included in the estimated region. This τ would probably not be used in practice since it does not produce the best overall score.

4 Improving A1 with retrieval strategies

In the original implementation the entire database needed to be scanned sequentially since neither the set F nor P could be retrieved directly. However, by using the estimated region we can now take advantage of range search methods. *Range searching* is the ability to search a database for records that satisfy specific range restrictions. Given a search point (x_c, y_c), range searching provides the ability to extract the set P directly. This can be done by indexing each of the dimensions and requesting all records that match $(x_0 \leq x_c \leq x_1, y_0 \leq y_c \leq y_1)$. Each of the dimensions could be indexed and searched independently. The result of the query would be the intersection of the results of each of the individual dimension queries. By indexing only one dimension it is possible to retrieve the set of records falling in a thin slice or slab. Indexing each of the dimensions may be useful but it is not clear whether the overhead of the second search and the required intersection operation would be an improvement over the linear scan of the results of the first query. For this research the database was indexed under a single dimension.

Range searching is an important well studied problem that has many excellent solutions [BF79] for static sets only. These solutions are usually based on sorting the points and performing efficient searches, or hashing the points into bins, areas, or grids. These algorithms usually assume all the points are known in advance and efficient indexes can be built before any queries are executed. Approaches that require a preprocessing step work efficiently with static data. These approaches would not work well in our application. In correlation and

fusion algorithms points are constantly being added and shifted in the database. This problem requires a dynamic data structure that can provide efficient range retrievals and has low maintainance overhead. Different types of balanced trees have been considered. The Btree was chosen because of its ability to satisfy these requirements [Com79].

Btrees are balanced, rooted trees with the following properties [CLR90].

- Every node has a restricted number of keys kept in sorted order.
- Given t, the branching factor or *minimum degree* of the tree, each node has:
  - a minimum of $t - 1$ keys
  - a maximum of $2t - 1$ keys
- Every leaf is at the same depth. The height of the tree is $h \leq log_t \frac{n+1}{2}$
- Internal nodes have at most $2t$ pointers to other nodes

In this implementation the interior nodes contain keys and pointers to other nodes. There is no data kept in the interior nodes. The leaves contain the data, indexes, or pointers to the data and a pointer to the next leaf. This structure is usually called a B+ tree and can be thought of as a sorted linked list with an index structure built on top 8. Range searches are implemented by searching for the lower bound using the imposed Btree index and then sequentially searching the linked list until the upper bound is reached. This method works particularly well with limited memory systems or with disk based data bases. The height of the tree, and consequently the number of disk or page access required to reach the lower bound, is low. To find the upper bound only the records required plus one actually need to be read.

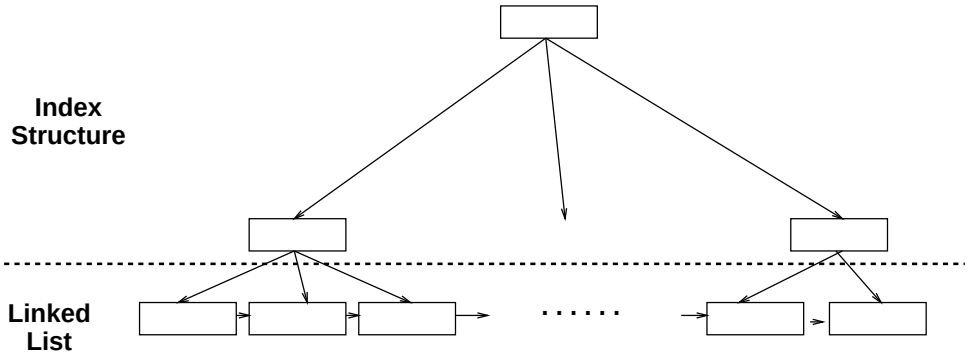

Figure 8: B+ Tree

5 Scaling the database

The desired operational size of the database is around 10 thousand to 15 thousand sites. Initial simulations were on databases of a few hundred sites since it was not feasible to run larger tests because of the length of time involved. The combination of region estimation and direct retrieval techniques improves performance so that scaling the simulations into the desired operational range is now viable.

The main advantage of the region estimation and direct retrieval combination is the low constant cost per message. In practical systems entities are not packed into an area without restriction. As entities are added to the simulation the area in which the entities are located is scaled to maintain a near constant density. Sensor performance parameters also remain consistent resulting in estimated regions of similar size. Since the area of the estimated region does not increase and the density remains consistent the number of records retrieved per range query remains approximately constant. This results in a near constant cost per message that is a function of the region density and sensor performance.

Two simulators were used to test the results of the implemented changes. Output of the sensor report generator, `simterr`, is affected by parameters such as the number of sites, number of targets per site, probability of detection, simulated observation time, maximum number of messages generated, sensor type and performance, sensor operation times, total area in which sites are generated, and terrain information.

`simterr` was used to generate five different placements and a corresponding set of sensor reports for environments of 300 – 3000 (in increments of 300), 6000, and 9000 sites. The same sensor performance parameters were used in all the simulations. This resulted in the error ellipses all being of similar size. The area in which the entities are placed was scaled to provide a constant entity density. Since no terrain information was used the entity distribution was uniform.

The second simulator, `testa1`, implements the A1 algorithm and the modified A1 algorithms. It operates on the message sets given parameters such as the discrimination level, τ, type of region estimation and retrieval strategy. Larger τ values affect the quality of the output representation and increase the simulation times. In order to reduce execution time τ was held constant at a very small value of .005,

The execution times for the simulations were gathered using the Unix `getrusage` facility. They reflect the CPU usage of the different algorithms and do not include wall clock time or time spent on system calls. This represents the processing costs of an entirely memory resident database which is the best case for the original algorithm. If it were necessary to move the database to a virtual memory or disk based system page fault or disk access times would also need to be considered. In that case the two linear algorithms would incur a greater cost while the Btree algorithm would incur a much smaller (near constant) cost.

Each generated message set was tested. The results of the five sets of each ground truth were averaged. Figure 9 and Table 1 show the results. The timing results show that the first two algorithms are linear in the size of the database and the Btree implementation results in near constant times. For the two linear algorithms there is a very slight leveling

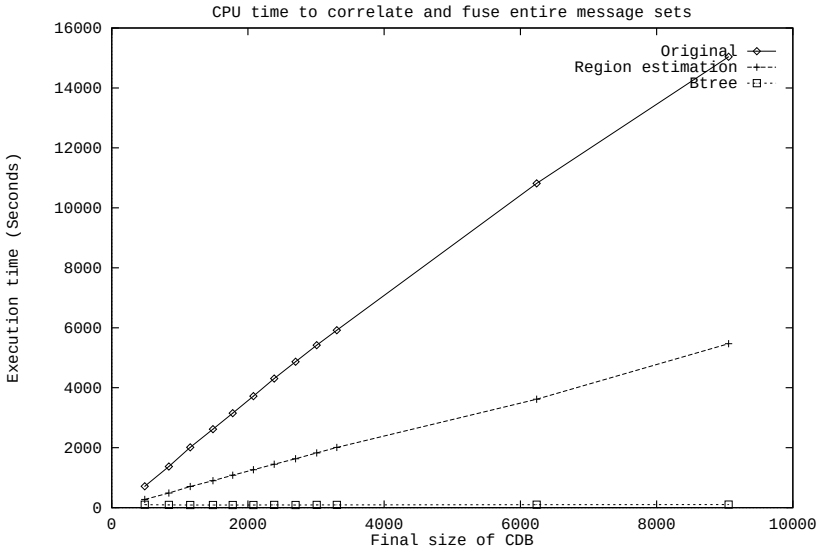

Figure 9: Total CPU time

# of sites	Db size	stdev	Original	stdev	Region Est.	stdev	Btree	stdev
300	485	12.1	714	11.8	273	4.5	99	0.4
600	839	17.6	1374	21.8	489	8.4	90	1.0
900	1153	12.7	2012	17.6	707	7.7	88	0.5
1200	1486	20.0	2619	33.0	901	8.3	89	0.5
1500	1777	17.9	3154	17.1	1080	6.0	89	0.8
1800	2079	17.7	3718	17.4	1264	5.9	89	1.0
2100	2384	10.9	4307	20.3	1450	6.3	90	0.9
2400	2698	4.3	4865	6.4	1631	3.0	90	1.0
2700	3009	19.3	5420	14.4	1825	5.5	91	1.0
3000	3303	11.5	5921	8.7	2010	4.0	92	0.8
6000	6238	16.8	10818	22.3	3616	5.8	97	0.5
9000	9058	8.6	15040	12.2	5469	5.4	104	0.9

Table 1: Execution times in seconds for entire data sets

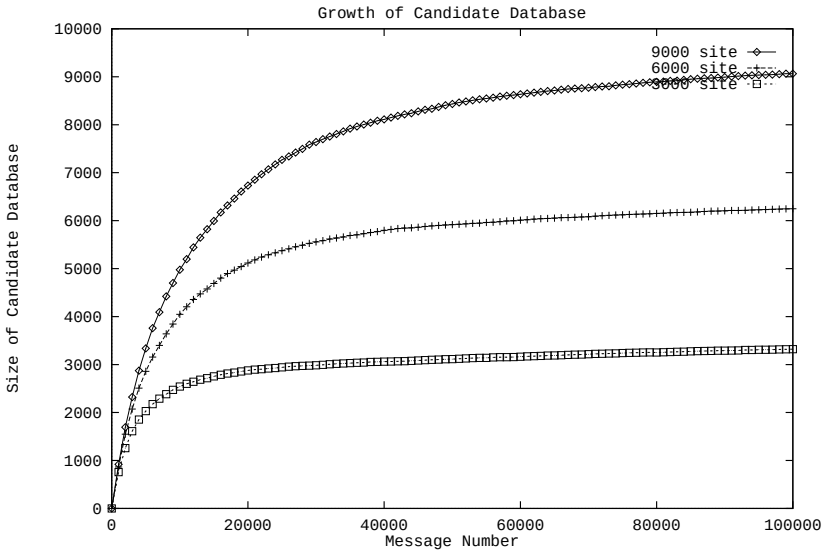

Figure 10: Number of candidates in data base

off visible in the graph. This is due to the lower cost of processing a message before the database reaches a steady state in the number of records. Since the larger databases take more time to reach a steady state this effect is more prominent in the simulations with a greater number of entities.

A constant size database would give a better estimate of the time required to process each message. Figure 10 shows how the database size grows as messages are processed and new entities are sensed and identified. Since the size of the database is constantly changing, we measured the time required to process a set of 10 thousand messages near the end of the data set. By this point in the simulation the database has achieved a near steady state. The results are shown in Figure 11. We can get a good estimate of the average cost per message per database record by dividing this time by the average size of the database during the 10 thousand messages. This is shown in Figure 12 and Table 2. This figure shows that the overhead costs of the Btree are amortized over the size of the database. The other two algorithms show a near constant cost per message per database record. However, there is a slight reduction in cost in these two algorithms for the smaller databases and it is unclear why this occurred.

6 Parallel approaches

The next phase of the project investigates the design and analysis of a parallel approach to this problem. Parallel solutions are inherently dependent on the architecture of the target system. For a shared memory machine a shared memory data structure which would allow

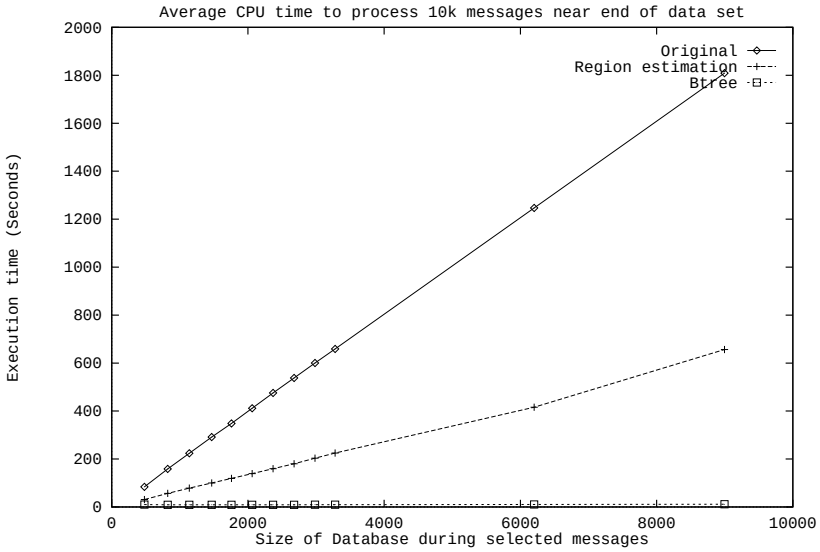

Figure 11: CPU time to process 10 thousand messages at a near steady database size

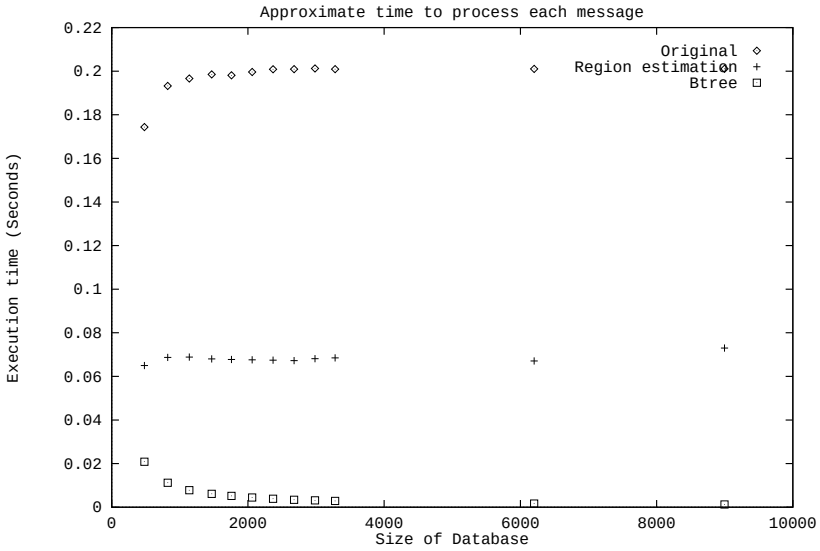

Figure 12: Average cost per message per record

Db size	Original	Region Est	Btree
480	0.174	0.065	0.021
822	0.193	0.069	0.011
1138	0.197	0.069	0.008
1468	0.199	0.068	0.006
1758	0.198	0.068	0.005
2061	0.200	0.068	0.004
2368	0.201	0.067	0.004
2677	0.201	0.067	0.003
2984	0.201	0.068	0.003
3280	0.201	0.069	0.003
6202	0.201	0.067	0.002
8997	0.201	0.073	0.001

Table 2: Average cost per message per record

multiple readers and at least one writer to any part of the tree would be an attractive approach. There are already established algorithms and protocols for manipulating this type of data structure. For a non-shared memory machine, such as a hypercube, a better approach would be to distribute the database and divide the map area into regions and assign a processor to each region.

An important question then is how to partition the area. If the partitions are defined to be rectangular and larger than any estimated message region it is possible to devise simple algorithms for processing the reports. An initial approach might be to divide the area up as shown in Figure 13. New messages would be processed by a dispatching node. The dispatch node would ensure that the estimated message regions would not overlap or interfere with each other. A message would be dispatched to the processor managing the region which contains the center of the EEP. EEPs wholly contained in a processor's region can be processed immediately. If a message's estimated region falls outside of the processor's region the processor must request relevant records from the neighboring processor. The estimated regions are guaranteed not to overlap so there is no danger of two messages trying to fuse with the same record. Since the processor regions are larger than the message regions we know that there will be at most three other processors that need to be queried.

A drawback to this approach is that there is no way to maintain a balance in the database size or processor workload. It is possible to devise a solution that resizes regions dynamically or starts up with irregularly sized regions. However, dynamic reassignment can be expensive and complicated and a scheme with statically calculated irregular regions would have the same drawbacks as the static range search approaches. Such static approaches would need to assume a knowledge of the entity distributions that we have not previously assumed.

With this type of partitioning, messages can occur that require communication with all three neighboring processors. Dividing the map area into horizontal strips reduces the

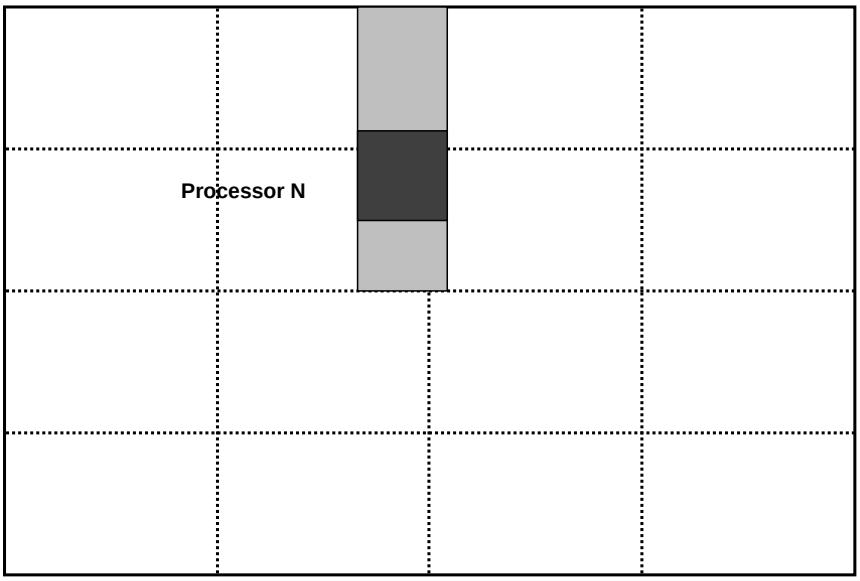

Figure 13: Grid division for parallel implementation

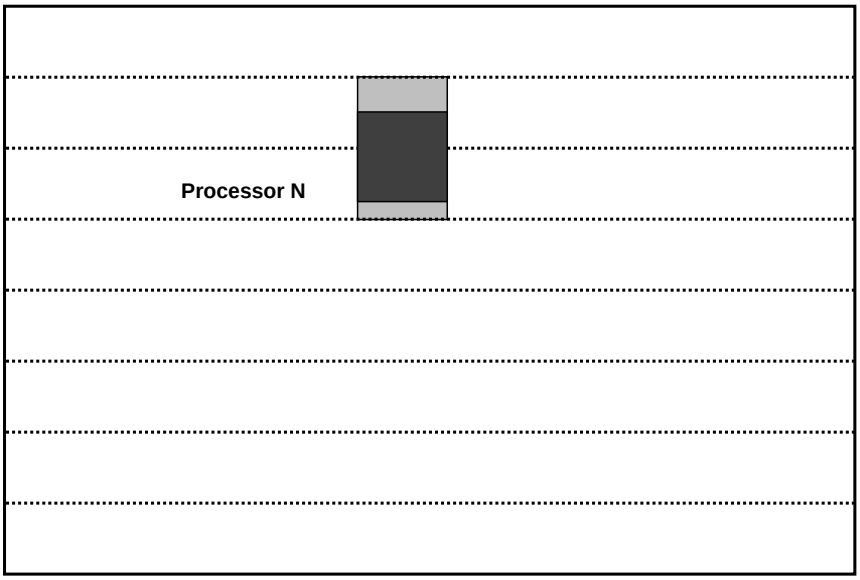

Figure 14: Horizontal division for parallel implemenataion

effects of these drawbacks. If processors are assigned in strips they can be placed so that they break up, or hash, the non indexed dimension into regions. This would reduce the size of the retrieved set P. Figure 14 shows a map area that has been divided into strips. This division requires communication with at most one other processor. This approach is also less sensitive, though not immune, to unbalanced densities and workloads. A potential disadvantage is the scalability of this scheme. The number of processors cannot be increased past a point where the width of a processor's region is smaller than the estimated region size.

Each of these parallel approaches relies on the guarantee that active messages do not interfere with each other. When there is interference by messages we can either block the processor until the interfering estimated region is released or the message can be delayed and another non interfering message processed. The first approach would sacrifice parallelism but is equivalent to the serial approach and would generate an identical representation. The second approach would permute the order of the messages in the stream in order to maximize parallelism. It is not clear to what degree the final representation would be affected by a permutation of the message stream.

7 Summary

This work was undertaken to show that the time required to process messages for correlation and fusion algorithms could be drastically reduced by using region estimation and direct retrieval techniques. We have shown that inexpensive region estimation can be used to reduce the number of expensive correlation tests and that range search techniques can result in near constant costs. The range estimation improvement alone cut running time to less than 40% of the original time. The reduction in time would be even more dramatic if more sophisticated correlation measures were used. The estimated regions also allowed the use of range search techniques. The range search method resulted in a near constant time of approximately 100 seconds over large operational ranges and a large message set size. These simulations had previously taken between 714 and 15,040 seconds. This is a significant improvement over previous approaches and allows the testing of much larger simulations and scenarios.

References

[BF79] Jon Louis Bentley and Jerome H. Friedman. Data structures for range searching. *Computing Surveys*, 11(4), December 1979.

[CLR90] Thomas H. Cormen, Charles E. Leirserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[Com79] Douglas Comer. The ubiquitous b-tree. *Computing Surveys*, 11(2), 1979.

[Fre93] James C. French. Estimating retrieval regions for correlation and fusion algorithms. Technical Report IPC-93-06, Institute for Parallel Computation, University of Virginia, October 1993.

[SJB89] Andrew R. Spillane, Clarence L. Pittard Jr., and Donald E. Brown. A method for the evaluation of correlation algorithms. Technical Report IPC-89-04, Institute for Parallel Computation and Department of Systems Engineering, University of Virginia, April 1989.