

Assessing Standard and Inverted Skip Graphs Using Multi-Dimensional Range Queries and Mobile Nodes

Gregory J. Brault¹, Christopher J. Augeri², Barry E. Mullins², Christopher B. Mayer², Rusty O. Baldwin²

¹Air Force Information Operations Center, Lackland AFB, TX USA

²Air Force Institute of Technology, Wright-Patterson AFB, OH USA

gregory.brault@lackland.af.mil

{christopher.augeri, barry.mullins, christopher.mayer, rusty.baldwin}@afit.edu

Abstract—The skip graph, an application-layer data structure for routing and indexing, may be used in a sensor network to facilitate queries of the distributed k -dimensional data collected by the nodes, such as their geographic positions. Nodes in a standard skip graph sort keys in layered groups, where group membership is determined by random membership vectors.

We propose a skip graph extension that inverts the key and membership vector roles, wherein group membership is based on deterministic z -ordering of k -dimensional data and sorting within groups is based on random keys. This extension retains the structure of a standard skip graph, and can be modified for use in 3-D environments such as unmanned aerial vehicle swarms.

Our results indicate this extension reduces the number of nodes contacted relative to the query precision, the volume of the multi-dimensional space, and the skip graph's height. We discuss our extension's performance relative to metrics at the application and network layers for static and mobile 2-D networks.

Keywords - skip graph; k -dimensional range query; mobility

I. INTRODUCTION

Considerable research efforts are devoted to distributed sensor network (DSN) technologies. DSNs consist of devices with limited power and processing resources that communicate when executing user requests. It is often not feasible to conduct research on physical implementations of a large DSN, e.g., an unmanned aerial vehicle (UAV) swarm of 10,000 nodes. Thus, DSN research often relies on analysis and simulation.

Our research focuses on a *skip graph*, an application-layer overlay useful for storing distributed k -dimensional (k -D) data. A skip graph [1], independently proposed as a SkipNet [7], and herein called a *standard* skip graph, is a distributed extension to the *skip list* [10]. A standard skip graph's nodes contain uni-dimensional (1-D) *keys*, e.g., temperature readings. Keys are hierarchically grouped on random group *membership vectors*. Most skip graph variants for indexing k -D data, e.g., node geographic positions, typically map the k -D data to 1-D keys versus changing group membership vectors [4][6].

This work is partially supported by the Air Force Communications Agency. The views expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Air Force, Department of Defense, or the U.S. Government.

We propose a skip graph extension that increases query performance of distributed k -D data under certain conditions. Our primary objective is to minimize query response time but not necessarily the energy required to execute the query. We assume the data is time-critical, e.g., the positions of nodes in a UAV swarm. Herein, we discuss modifications applied to a standard skip graph in order to create an *inverted* skip graph capable of indexing k -D data. We then present results obtained by simulating k -D queries in a static sensor network of fixed nodes and maintenance costs in a dynamic sensor network containing mobile nodes.

II. USING K -DIMENSIONAL DATA WITH SKIP GRAPHS

A. Linearization: Mapping k -D Data to One Dimension

Although the raw data may be k -dimensional, skip graphs store all data as 1-D keys. Therefore, k -D data must first be mapped (linearized also reduced) to one dimension. Herein, we use a node's geographic position as a k -D data source. Several dimension reduction techniques exist to map a node's (x, y) coordinates to a 1-D scalar value. A frequently used method is the z -order space-filling curve (SFC) [2], as shown in Figure 1.

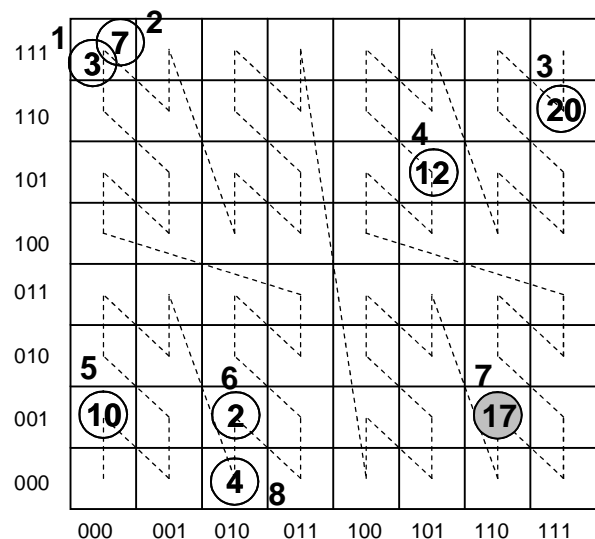


Figure 1. 2-D Area with 8 Nodes and a Z-order SFC

Figure 1 shows a 2-D area populated with eight nodes and overlaid with a 2-D z-order curve. The number inside a node is its sensor reading to be used in query examples; a number adjacent to a node denotes its global unique identifier (GUID). The (x, y) axes are given in binary to aid illustrating the z-order mapping. The z-order curve was first defined by Lebesgue [11] and named after the shape it forms as it snakes through a k -D space. It is also known as bit interleaving and Morton order [8]. We are not using more “optimal” linearization methods, such as multi-dimensional scaling (MDS) [3] or PageRank [9], given our application’s distributed nature.

A z-order curve has several useful properties with respect to querying geographical data. First, it is logically equivalent in 2-D to a quad-tree, where 2-D space is recursively divided into four equal-sized quadrants [5][12][13][8]. Figure 2 shows how a unit grid can be divided into four quadrants and each of those quadrants further divided into four quadrants, and so on. A node’s z-order value is the hierarchical concatenation of each quadrant that the node is located. For example, node ‘3’ is located in the top-left quadrant in the highest level, and then the top-left quadrant within that quadrant, and then the top-left quadrant within that quadrant, giving its z-order value of 101010₂. Likewise, node ‘12’ is located in the top-right quadrant, then the bottom-left quadrant of that, then the top-right quadrant within that, giving its z-order value of 110011₂.

A z-order curve can also be efficiently and locally computed since the bits of k coordinates are simply interleaved, or “zipped” together. The resulting value can then be used as a 1-D key in a skip graph. Figure 3 shows the z-order mapping process based on the coordinates of node ‘7’ in Figure 1 having a sensor reading of ‘17’ and whose geographic coordinates are (6, 1) or (110₂, 001₂). The z-order value is given by interleaving successive bits of each coordinate, e.g., 101001₂.

Query *precision* refers to the number of hierarchical levels of the equivalent quad-tree referenced in a query. In Figure 2, a query with a precision of one level returns all nodes in the top-most level of the grid. For example, the query |10X₂| should return nodes within the ‘10₂’ quadrant of the highest level, where the ‘X’ denotes “don’t care” - i.e., all nodes matching the prefix ‘10₂’. Thus, in Figure 2, this query returns node ‘17’.

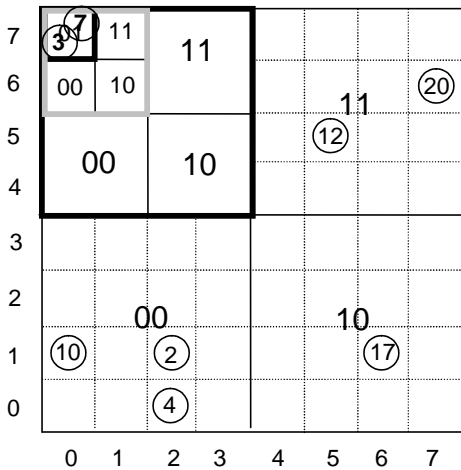


Figure 2. Hierarchical Quadrant Numbering

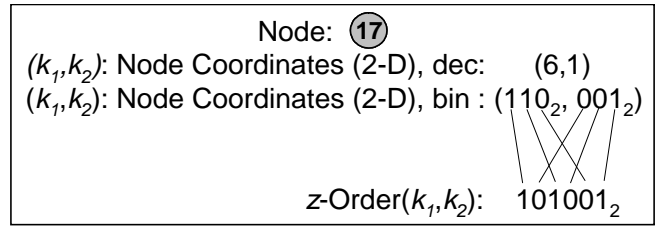


Figure 3. Linearizing 2-D Coordinates Using Z-Ordering

B. Multi-Dimensional Standard Skip Graphs

To index k -D data, e.g., node positions, in a standard skip graph, 1-D keys are locally computed using some linearization process, e.g., z-ordering; a node then generates a random group membership vector. Figure 4 depicts a k -D standard skip graph based on the nodes shown in Figure 1. Keys are first computed deterministically from a k -D source, i.e., node (x, y) positions: [(2, 1), (0, 7), (2, 0), (0, 7), (0, 1), (5, 5), (6, 1), (7, 6)]. We then compute z-order values, i.e., keys: [001001₂, 010101₂, 001000₂, 010101₂, 000001₂, 110011₂, 101001₂, 111110₂]. The respective random group membership vectors used are [100110₂, 000100₂, 011011₂, 110001₂, 000101₂, 110100₂, 110101₂, 010011₂].

Range queries of k -D data are especially useful for locating nodes within a geographic boundary, e.g., identifying nodes in a certain area of the network illustrated in Figure 1. We shall use the range defined by a rectangle bounded on the lower-left by coordinate grid (2, 0) and on the upper-right by coordinate grid (3, 1). We can see from the figure this range query should return nodes ‘6’ and ‘8’. Since key values in a k -D standard skip graph are z-ordered linearizations of the node’s geographic coordinates, the range query must be converted to z-order as well. Using the process outlined in the Section IIA, the z-order of the first coordinate, (2, 0), is (001000₂) and the z-order of the second coordinate, (3, 1), is (001011₂). Thus, the query range of |(2, 0) – (3, 1)| becomes |001000₂ – 001011₂|.

Figure 4 depicts a standard skip graph and the range query of |001000₂ – 001011₂|. The query is injected in the skip graph at node ‘62’. The query transfers to adjacent nodes within the same level until it is determined that it should not propagate in that level anymore. This is determined when an adjacent node’s key lies beyond the opposite end of the query range in the direction that the query is propagating. For instance, if the query is propagating to the left and searching for a key within the key range of |8 – 11|, and currently holds a reference to a node with a key of ‘4’, the query can determine it must drop to a lower level to explore more nodes that might be relevant to the query. The query must also drop to a lower level if the node where the query is presently located is at the end of a list.

As Figure 4 illustrates, once a node is found that is within the query range, the query drops down to the base list L_0 and advances in both the left and right directions until the query finds nodes with key values that are outside of the query range. This is indicated in Figure 4 by the two ‘X’s. As this figure shows, the two node GUIDs found are {6, 8}, which as noted earlier, are the two nodes that are located within the geographic query range |(2, 0) – (3, 1)|. We explicitly note that we are executing geographic range queries, not node GUID queries.

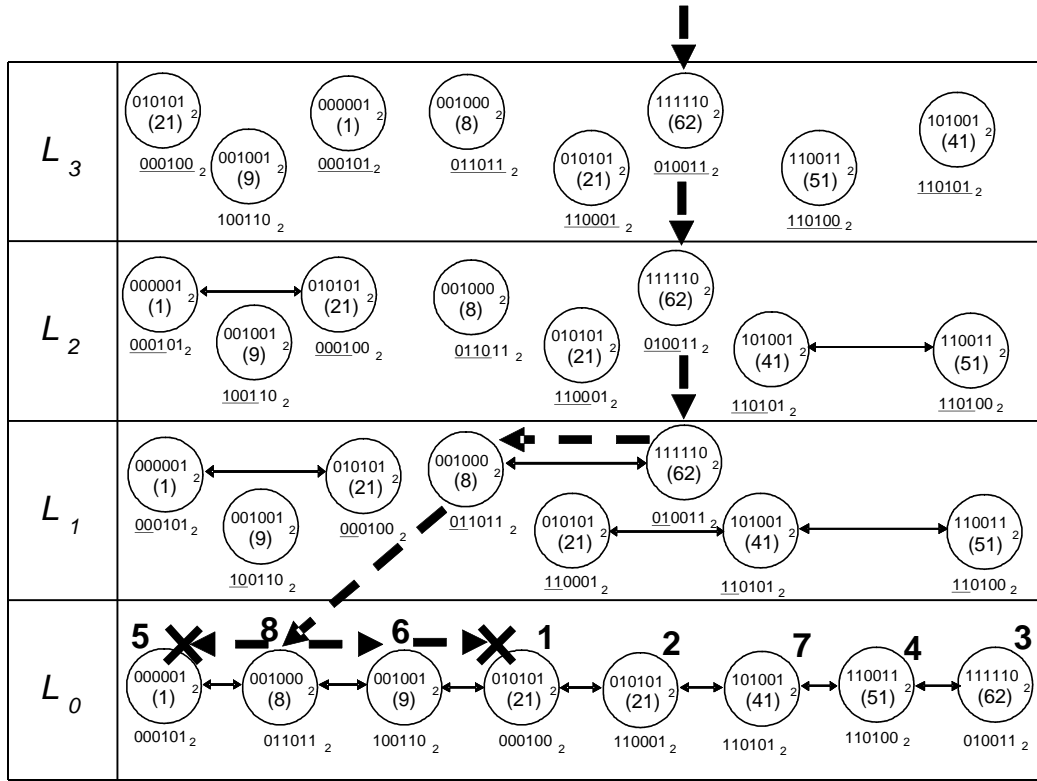


Figure 4. K -Dimensional Standard Skip Graph Executing a Linearized Geographic Query ‘ $8 \leq key \leq 11$ ’

C. Multi-Dimensional Inverted Skip Graphs

Our proposed extension to the standard skip graph *inverts* the deterministic key and random membership vector roles of a standard skip graph. In a k -D inverted skip graph, the node key values are generated randomly at each node in the network, much as how membership vectors are randomly generated in a k -D standard skip graph. Furthermore, the node membership vectors are deterministically computed from the k -D data being queried, much like the key value linearization in standard skip graphs. We still use z -ordering to linearize the k -D data being stored, however, the z -order bit sequence now deterministically defines the node membership vectors, not the node keys.

Figure 5 depicts a k -D inverted skip graph holding the same nodes stored in the k -D standard skip graph shown in Figure 4. The key difference is the node keys are now randomly selected: [5, 13, 29, 37, 40, 63, 70, 89]. Similarly, membership vectors are now computed deterministically by z -ordering each node’s geographic coordinates. These values are identical to the node keys in the standard skip graph: [001001₂, 010101₂, 101001₂, 010101₂, 000001₂, 110011₂, 001000₂, 111110₂].

To perform a range query in a k -D inverted skip graph, the query is injected at a node as it is in the standard skip graph. However, instead of the query starting at the topmost level, the query begins at the base level, L_0 , and traverses in both directions. The query traverses the base list until it finds a node having an L_1 membership vector prefix matching the query’s prefix for L_1 . When the query finds that node, it advances to L_1 and begins traversing the list associated with the matching membership vector. It follows a similar process as it did in the

base list, but now the query is looking for a longer prefix to match. The query continues this process of longer prefix matching as it advances up the skip graph levels and terminates when one of three conditions is met.

The first condition arises when the query’s prefix fully matches the prefix of a membership vector on some level of the skip graph. This indicates that all nodes within that list should be notified of the query. The query progresses in both directions until reaching both ends of the list. For example, Figure 5 illustrates the execution of the same range query as found in Section IIB, a query range of [001000₂ – 001011₂]. This query returns all nodes that have a membership vector prefix of ‘0010₂’, the longest common prefix of the range.

The second condition for terminating a query occurs when a query passes to the highest level in the skip graph. At this point, the query prefix is being compared with the longest possible membership vector prefix. The query must traverse this entire list, and at each node compares the entire query prefix with the appropriate prefix length of the membership vector, and either includes the node in the query results or not.

The third condition for terminating a query occurs when a query traverses a list and does not find any nodes having a membership vector prefix that matches the query prefix for the next higher level, L_{i+1} . Continuing our example with Figure 5, if the query was [0011X₂], upon reaching L_1 the query would traverse through the entire list of node ‘70’, node ‘40’, and node ‘5’, and not find any L_2 membership vector prefixes of ‘0011₂’. Therefore, the query determines that no nodes exist with a membership vector prefix of ‘0011₂’.

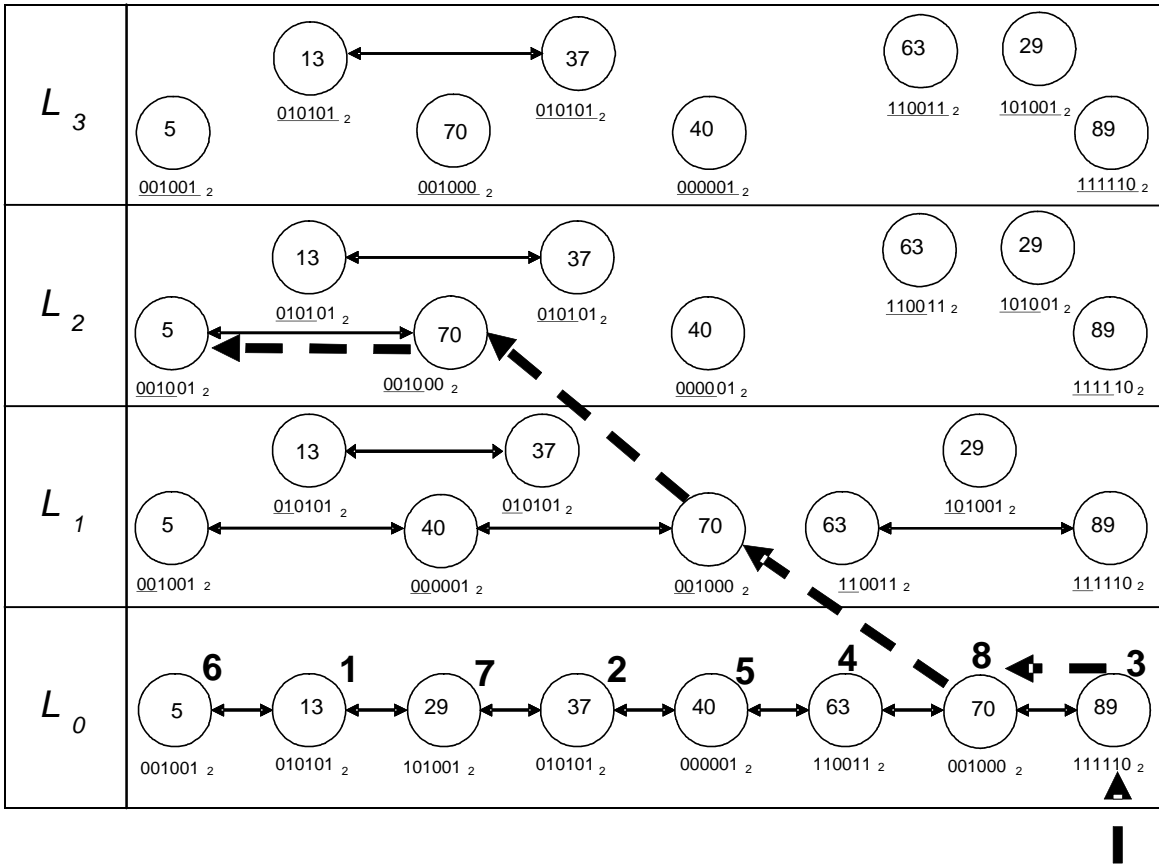


Figure 5. K -Dimensional Inverted Skip Graph Executing a Query '0010X'

For example, consider a query $[0010X_2]$ injected into the skip graph at node '89' and progresses to the left in the base list L_0 . The query is transferred to node '70', where it is determined that the L_1 membership vector prefix '00₂' of node '70' matches that of the query $[0010X_2]$ at that level. At this point, the query remains within node '70', but advances to L_1 . It is determined that the L_2 membership vector prefix '0010₂' of node '70' also matches that of the query $[0010X_2]$ at that level. The query can now traverse the entire list at this level in which node '70' resides, because the query prefix of '0010₂' exactly matches. Figure 5 shows that the two nodes returned in the query, nodes '5' and '70', which are the two nodes within the specified query range of $[001000_2 - 001011_2]$, i.e., $[0010X_2]$.

III. MEASURING PERFORMANCE

The metric we use to compare the performances of standard and inverted skip graphs is the number of skip graph messages sent to execute some task. Two types of tasks were examined in this research: executing k -D range queries, and maintaining a skip graph by updating broken logical links among nodes due to mobility. The skip graph applications we have in mind, e.g., UAV swarms, require short query execution times, and so we wish to minimize query response time, possibly at the expense of energy. Thus, analyzing the number of skip graph messages required to execute a query or to update the skip graph due to node mobility is a reasonable metric to use for comparing skip graph performance. An increase in the number of messages required to execute a query equates to a longer response time.

Using the standard OSI network model as a template, skip graphs operate in the application layer. Nodes in a skip graph have logical references to other nodes in every level of the skip graph. When skip graph messages are sent from one node to another, the messages are sent using the network layer's routing protocol for that particular network. Thus, a single skip graph message may be transmitted through several nodes in the physical network.

The simulations record the number of application-layer skip graph messages. However, a full network-layer simulation is not implemented. Instead, the geographical distance between two nodes logically connected in the skip graph is used as an estimated metric as to the network layer cost associated with sending a message between those two nodes. Figure 6 shows how this is calculated using our example network.

A logical link between node '4' and node '12' may exist in a particular level in a skip graph as shown in Figure 6. However, the physical route a message takes between the two nodes is most often not a direct path between them. Figure 6 illustrates that instead of following a direct path from node '4' to '12' a message may be forwarded to node '12' through node '17', shown by the dotted line. This physical route cost is estimated in the simulations by calculating the distance between the two nodes and using that as the cost metric. The farther that two nodes are separated, the more hops in the network layer will (generally) be incurred to send an application-layer skip graph message between them.

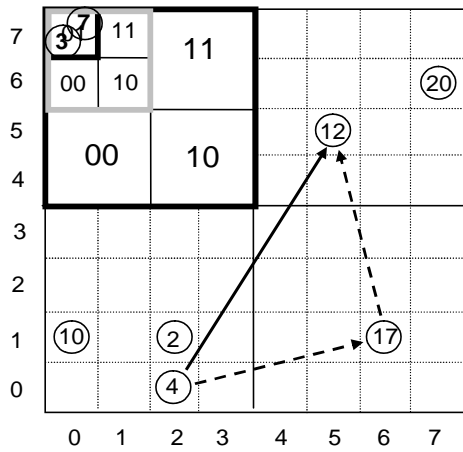


Figure 6. Estimating Network Layer Cost

IV. RESULTS

Both skip graph types and the querying algorithms were implemented in Java using the Eclipse integrated development environment (IDE) [14]. As mentioned above, two tasks are independently simulated in this research: executing k -D range queries, and updating broken logical links between nodes in the skip graph due to node mobility.

A. Multi-Dimensional Range Querying Results

The k -D querying task analyzed six unique query scenarios each having different query precision levels. Each data point in the following figures represents an average of one hundred iterations in that particular query scenario. In each of the iterations, a new skip graph for each type is created and populated with a new random distribution of varying node counts. The x -axis in each graph indicates the number of nodes contained in the skip graph for that particular simulation. The query is executed, and the number of messages required to perform the query is recorded along with the network layer cost estimate. We highlight two of the six query precision levels we assessed in this section: Figure 7 provides results for a scenario in which the inverted skip graph outperforms the standard skip graph, and Figure 8 summarizes results for a scenario in which a standard skip graph outperforms an inverted skip graph.

Figure 7 shows the results of a $|100110X_2|$ query execution simulation. Standard skip graphs having 4 levels perform this query poorly compared to 4-level inverted skip graphs as node count *increases*. In this particular query scenario, inverted skip graphs outperform standard skip graphs in networks with more than one thousand nodes.

The main reason for the poor performance in the 4-level standard skip graph is as the node count increases, so does the length of the linked lists in each level of the skip graph. The two different query methods for each type of skip graph impacts query performance, especially as node counts increase. In a standard skip graph, which starts the query at the topmost level, it is possible to traverse several nodes before dropping down to a lower level. A list is traversed as long as the query is approaching the specified key range in that level without traversing beyond the key range.

If a query determines it cannot proceed at a given level, it drops to a lower level at the node the query is currently located. As the node count increases in a standard skip graph with a fixed number of levels, the lists at the topmost level can become quite long, forcing the query to traverse extraneous nodes before dropping to a lower level.

In an inverted skip graph with a fixed number of levels, extraneous nodes are still contacted, but the process is quite different. Recall that an inverted skip graph query begins at the base level, L_0 . The query traverses in both directions, looking for a matching prefix in the membership vector of each node. The node keys are random, and so the probability a matching prefix will be found is a random variable. As Figure 7 shows, inverted skip graph querying outperforms the standard skip graph's querying mechanism for the specified query scenario.

Figure 8 shows the results from executing the geographic query $|100110001101_2|$. As the figure indicates, there are now scenarios in which standard skip graph outperforms the inverted skip graph. This behavior is primarily related to the precision of the query being executed.

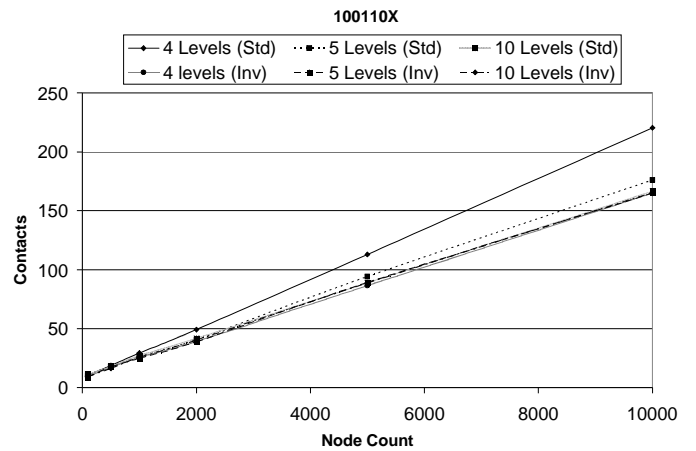


Figure 7. Query Performance Analysis: $|100110X_2|$ (Application Layer Messages, Low-Precision Query)

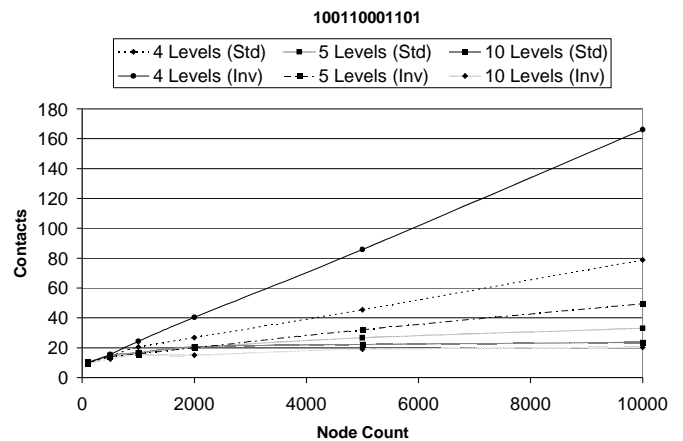


Figure 8. Query Performance Analysis: $|100110001101|$ (Application Layer Messages, High-Precision Query)

Each hierarchical quadrant of the query is a level in the skip graph, as shown in Figure 6. A query of the entire area returns all nodes, i.e. the base level, L_0 , of the skip graph. A query of $|10X_2|$ returns a sub-list in L_1 , the one containing node '17' (cf. Figure 5). This corresponds to the geographical layout in Figure 6 where node '17' is the only node in the top-level quadrant '10₂'. A query of $|100110001101_2|$ is a perfectly legal query to execute, however performance is hindered severely for inverted skip graphs of four levels or less.

The precision of the query is such that instead of returning an entire linked list in the skip graph, the query must traverse the entire list, returning only those nodes whose membership vectors match the entire prefix of the query. Those extra node contacts reduce the performance of the query execution. This explanation is reinforced by analyzing scenarios using more than five skip graph levels. In these instances, an inverted skip graph performs comparably to a standard skip graph.

Figure 7 and Figure 8 provide interesting insights into the relationship between query precision and the skip graph level count. The figures illustrate that if the query precision increases and the number of skip graph levels is the same, an increase in extraneous node contacts occur. These extraneous node contacts thus degrade query performance due to the increased response time needed for all of the extra messages to be sent.

The effects suffered from additional query precision are not as prominent in the standard skip graph query results, as shown in Figure 8. A standard skip graph query drops to the base level when it finds a node with a key value within range. Since the nodes are sorted in the base list, the query traverses in both directions until finding nodes with key values outside of the query range. The query does not waste time traversing through nodes that are ultimately not included in the final query results.

These results indicate that for certain querying conditions, standard skip graphs outperform inverted skip graphs. However, these simulations were performed in a static network in which nodes are not mobile. The following section shows that networks consisting of mobile nodes are able to maintain an inverted skip graph structure with fewer application-layer messages than a standard skip graph structure.

Another conclusion that can be drawn from the two figures is that increasing the skip graph level count leads to similar query performances for both types of skip graphs. That is, if the number of skip graph levels is above the required number for efficient query performance for a particular precision, increasing the number of skip graph levels while keeping total node count fixed causes the query performances of both skip graphs to become comparable. In that situation, the total node count needs to increase to see any gains in query performance.

Although inverted skip graphs perform better at the application layer, Figure 9 shows the relationship between total node count in a network and the average total distance traveled by all messages. In low node count networks, messages must traverse large portions of the network, while a denser network allows for a more efficient query path. Recall this total distance metric is an estimate of the network layer cost, and although the messages in a standard skip graph appear to travel a shorter distance, both skip graph types show a similar response.

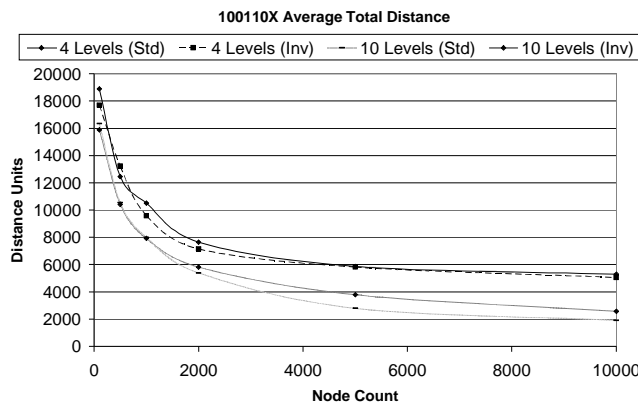


Figure 9. Query Performance Analysis: 100110X₂ (Network Layer Cost Estimates)

Figure 10 shows the estimated network layer cost for a $|100110001101_2|$ query simulation. This figure is similar to Figure 9 in that the query executing in the standard skip graph traverses a shorter distance, on average, than a query executing in the inverted skip graph. Keep in mind, these queries are executing in static networks, and may change when mobility is considered. Networks with lower node counts still traverse a higher distance than networks with larger node counts for the same reasons as specified above.

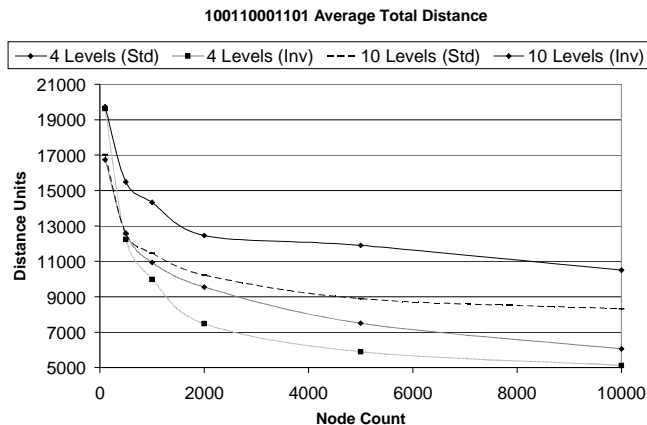


Figure 10. Query Performance Analysis: $|100110001101_2|$ (Network Layer Cost Estimates)

Table 1 summarizes all the of query simulations performed. The grid indicates which skip graph performs better as node count increases at the specified number of skip graph levels and precision of query, where a '-' denotes a tie, and 'inv' ('std') indicates an inverted (standard) skip graph performed better.

Table 1. Summary of Query Performance

Query Precision	Number of Levels				
	4	5	6	8	10
10X	inv	-	-	-	-
1001X	inv	inv	-	-	-
100110X	inv	inv	-	-	-
10011000X	std	inv	inv	-	-
1001100011X	std	std	-	inv	inv
100110001101	std	std	-	inv	inv

B. Node Mobility Results

The node mobility task considered the number of messages required to repair the skip graph when a single node changes its geographical location. Similar to the query simulations, each data point in the figure represents an average of one hundred iterations. In each of the iterations, a new skip graph is created and populated with a new node distribution. A single node is then chosen to have its geographic position randomly altered, i.e., it is “moved” to a new location. The number of update messages needed to repair the skip graph is then recorded.

Figure 11 reflects four of eight collected data sets; the two data sets that increase linearly are from the standard skip graph simulations, i.e., as the network node count increases, more update messages are required to maintain a standard skip graph. The inverted skip graphs, however, demonstrate a relatively flat response, i.e., the same number of update messages are needed to repair the inverted skip graph, independent of the node count

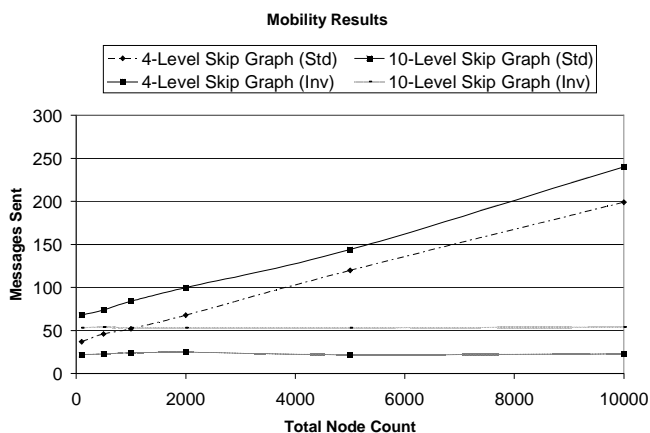


Figure 11. Message Traffic Induced by Randomly Moving a Single Node

The primary reason for the two different responses is that a node inserted into a standard skip graph must perform a query within the network to determine its position in the base list, L_0 . As network size increases, that query requires a greater number of messages to execute. An increase in messages sent implies an increase in the time it takes to insert the node.

Likewise, a node inserted into an inverted skip graph does not need to perform a query within the network. The node’s key does not change, so its position in the base list remains the same. The message counts result from the node’s re-assignment to new linked lists in each level of the skip graph due to the change in the node’s membership vector.

These re-assignment message counts are much smaller than the number of messages incurred by executing a query within the entire network. The small increase in the number of messages sent as the number of skip graph levels is increased in an inverted skip graph is a result of slightly more messages being sent as the node iterates through the levels at insertion.

Figure 11 shows that inverted skip graphs are well suited to mobile networks as compared with a standard skip graph. The number of messages required to repair a skip graph is directly

proportional to the number of node movements generated. The generally flat response of inverted skip graphs is very attractive as compared with the linearly increasing response of standard skip graphs. A flat response indicates that the number of update messages sent in order to maintain an inverted skip graph is not dependent on the number of nodes in the skip graph. However, the performance of standard skip graphs is negatively impacted as the network node count increases.

V. CONCLUSIONS

A. Results

The results show the inverted skip graph that we propose executes range queries better under certain conditions. First, the number of levels in the skip graph must be greater than the query precision. In our simulations, the standard skip graph outperforms the inverted skip graph by 54.5% using a query precision of six levels ($|100110001101_2|$) executed on a 4-level skip graphs, with a node count of 10,000. When the number of skip graph levels exactly accommodates query precision, the inverted skip graph outperforms the standard skip graph by 8.7%. The scenario we first observe this behavior is when using a query precision of three levels ($|100110X_2|$) on 4-level skip graphs and a node count of 10,000. These results are obtained in static networks.

Second, if the number of skip graph levels accommodates query precision, inverted skip graphs outperform standard skip graphs as node counts increase. This is shown in all of the figures in which the query precision is equal to or less than the number of levels in the skip graph. As a corollary, as the number of skip graph levels are increased for each of the skip graph types, a higher node count is required to notice performance differences. In networks having only a few nodes, there is no significant impact on range query performance using either standard or inverted skip graphs. Essentially, the query propagates through the network so quickly that both skip graph types exhibit similar performance.

Lastly, if the number of skip graph levels accommodates query precision, as the range of the query increases, the query performance of both skip graph types become comparable. The ratio of queried nodes to total nodes is so large that both skip graph types are able to process the queries comparably.

Inverted skip graphs are capable of processing mobile node updates within the skip graph with fewer skip graph messages. In a 10,000 node network, inverted skip graphs process a mobile node’s position update using a fourth of the messages (on average) a standard skip graph requires for the same task. When a node changes geographical locations in the context of a standard skip graph, a query is required to re-assign the node in the proper position in the base list in L_0 .

This update query is the root cause of poor performance in mobile networks observed when using a standard skip graph. Since an inverted skip graph does not need to perform this query, it has an almost flat response as node count increases. Alternatively stated, a flat response indicates maintenance of an inverted skip graph is not as negatively impacted by an increase in node count as a standard skip graph, especially in mobile scenarios, such as a UAV swarm.

B. Future Work

This research has generated several possible paths of future work. First, the z-ordering process is easily extended to a 3-D, environment. Specifically, we are interested in examining how a 3-D inverted skip graph performs, e.g., in a UAV swarm.

We used physical node distance to estimate network layer performance of both skip graph types. A more robust method is to use a network simulator, e.g., OPNET [15], to obtain precise performance results for query and mobility simulations. Since our skip graph implementation is in Java, one option is to use an OPNET co-simulation library, such as JOCosim [2].

Although standard skip graphs have better performance for many queries in static networks, mobility performance results appear to be quite promising for inverted skip graphs. The query simulations performed in this research were executed in static sensor networks. Further research should be conducted to study query execution, and not just skip graph repair behavior, within mobile networks. This larger set of simulations would also benefit from the use of an analysis of variance (ANOVA) in order to identify significant factors influencing the optimum skip graph to use in a given network.

These factors can then be used to foster developing a hybrid skip graph, i.e., a combination of the standard and inverted skip graphs. This research indicate there are scenarios in which the inverted skip graph outperforms the standard skip graph, but performance suffers when the specified query conditions are not met. This work would also provide a more robust means of determining when to switch between the two skip graph types.

ACKNOWLEDGMENT

We thank Gauri Shah for providing an implementation of the standard skip graph [1]. We also thank Hanan Samet for supplying us an advance copy of [12].

REFERENCES

- [1] Aspnes, J. and Shah, G. "Skip Graphs". In *Proceedings of the 14th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 384-393.
- [2] Augeri, C., Morris, K., and Mullins, B. "HARVEST: A Framework and Co-Simulation Environment for Analyzing Unmanned Aerial Vehicle Swarms", in *Proc. of the 25th Military Communications Conference (MILCOM)*, Washington D.C., 23-25 October 2006, IEEE.
- [3] Borg, I., and Groenen, P. *Modern Multidimensional Scaling*. Springer-Verlag, 1997.
- [4] Eppstein, D., Goodrich, M. T., and Sun, J. Z. "The skip quadtree: a simple dynamic data structure for multidimensional data". In *Proc. of the 21st Symposium on Computational Geometry (SGC)* ACM Press, 2005, pp. 296-305.
- [5] Finkel, R. A. and Bentley, J.L. "Quad trees: a data structure for retrieval on composite keys". *Acta Informatica*, 4(1):1-9, 1974.
- [6] Ganesan, P., Yang, B., and Garcia-Molina, H. "One torus to rule them all: multi-dimensional queries in P2P systems". In *Proceedings of the 7th Int'l Workshop on the Web and Databases (WebDB)*, ACM Press, 2004, pp. 19-24.
- [7] Harvey, N. J. A., Jones, M. B., Saroiu, S., Theimer, and M., Wolman. "SkipNet: a scalable overlay network with practical locality properties". In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.
- [8] Morton, G. M. *A Computer Oriented Geodetic Data Bases; and a New Technique in File Sequencing*. IBM (Ontario Research Center), Technical Report, March 1, 1966.
- [9] Page, L., Brin, S., Motwani, R., and Winograd, T. *The PageRank Citation Ranking: Bringing Order to the Web*. TR 1999-66, Stanford University, Stanford, MA, 1998.
- [10] Pugh, W. "Skip Lists: a probabilistic alternative to balanced trees". *Communications of the ACM (CACM)*, 33(6):668-676, 1990.
- [11] Sagan, H. *Space-Filling Curves*. Springer-Verlag, New York, NY, 1994.
- [12] Samet, H. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006
- [13] Warnock, J. E. *A Hidden Surface Algorithm for Computer Generated Halftone Pictures*. Doctoral Thesis, University of Utah, 1969.
- [14] Eclipse IDE. URL <http://www.eclipse.org>
- [15] OPNET. URL <http://www.opnet.com>