17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID 344

Dynamic Onboard Routing Algorithm for LEO Satellite Constellations

Abdelrahman Metwally[a]

[a]Department of Engineering Systems, Skolkovo Institute of Science and Technology, Bol'shoy Bul'var, 30, 1, Moscow, 121205, Abdelrahman.Metwally@skoltech.ru

Abstract

Satellite constellations to provide flight segment for Internet of Things connectivity are coming very fast and there are many prototypes in orbit already. One solution is to install many receiving ground stations. In this work, we consider a dynamic algorithm that uses radio communications to route messages dynamically to the closest satellite to the ground station. This approach will enable delivery of a signal with minimum delay, while keeping costs of mission operations at a minimum, since we're routing employs existing capabilities of a small satellite. The key challenge of this work is the development of efficient routing algorithms which take into account relative motion between satellites, available memory and computing power constraints, specifically in a decentralized and dynamic topology network. Here we describe an onboard dynamic routing algorithm for low earth-orbit satellite constellations, which can be used to build a simple computer network in space. We developed a routing algorithm that depends on time and trajectory parameters. The algorithm dynamically updates the routing table using the SGP4 orbit propagator to calculate satellite routes and geographic positions. Each satellite acts as a node in the network. Each has a fixed number (similar to IP address). The network map shows the available nodes (e.g. Satellite or Ground station) corresponding to the current geographical position. The routing table contains ephemeride elements (two line elements, which are updated regularly) for all satellites in a constellation. Each node is a transparent re-transmitter to relay traffic received from terminals and earth gateway stations, and return the traffic to the ground. Also, each node with an Intersatellite Link capability is a network switch for being able to communicate with neighboring satellites. In our approach, terminals within a node visible area can specify the ending destination for a data packet and the algorithm will decide and calculate the fastest and shortest path at any moment in time and use it for transmission to the destination. The purpose of this algorithm is to allow distributed systems to operate autonomously, provide real-time data transmission, and extend the World Wide Web into space by allowing ground stations to act as gateways between satellites, and can also work with any COTS. OBCs (ISIS Computer or Beaglebone Black) are also present in all Linux distributions. We implemented the algorithm on Kubos Linux on Beaglebone black OBC. Linux-based operating systems allow algorithms to run as background applications without interfering with key mission applications.

1 Introduction

Satellite constellations are the key feature for the next era of new space technology, which can provide global coverage and real-time communication also can be used in many applications such as weather monitoring, navigation, communication and many more. Satellite constellations are becoming trendy nowadays (Example: IRIDIUM, LeoSat, SpaceX Star-link, Telesat), many

organizations and research centers are making substantial investments in the satellite constellations to carry out fundamental scientific research, advancing knowledge for humanity.. In order for these satellite constellations to achieve its mission objective in the harsh space environment, specially constellations with Inter-satellite links, these constellations mainly depend on sharing data between each other and users; therefore, an optimal networking topology is required for these cases. Satellite networks are quite different from the traditional ground network with significant transmission delay, transmission loss, and high dynamic change of the network topology. To use satellite networks, routing issues must be solved first. it has been the focus of research in networking topologies, routing topology is still a challenging problem because the satellite has the following obstacles: limited on-board processing and storage capabilities, highly dynamic changes in geolocation and high bit error. Several strategies have been proposed.

1.1 Routing based on virtual topology

The idea behind the virtual topology[1] strategy is to divide the system period into n time slices as [t0,t1][t1,tn] where t is the period; each time slice can be considered as one snap-shot. The connection and disconnection of the links present only at a discrete-time. The dynamic topology of the network can be modeled as fixed topology. However, this method has many problems, such as that significant storage is required to store routing information because the system is divided into a massive number of time slices.

1.2 Finite State Automata Algorithm

FSA[2] routing algorithm is dividing the network into a finite number of time slices, and it is a connection-oriented routing algorithm, the FSA algorithm takes the network topology of the slice as a state, and the dynamic topology of the satellite network is modeled as a finite state machine. However, the FSA has an advantage is the optimization of network resources. Also has a disadvantage in the shortest path is not the optimal path, dynamic link allocation technology is complicated for LEO constellations, and computational complexity is too high.

1.3 Compact Explicit Multi-path Routing Algorithm

CEMR[3] routing algorithm is the first algorithm to study the multi-path routing problem of a mobile satellite network. The idea behind the CEMR algorithm is to encode the path using a compact path identifier, such as PathID is a global representation. meanwhile, the CEMR algorithm can reduce delay, increase the throughput, and balance the load because the traffic is divided towards feasible adjacent paths.

1.4 Priority-based Adaptive Routing Algorithm

PAR[4] Takes into account the link utilization and historical information in order to achieve a uniform load distribution. PAR sets the destination using a priority mechanism that relies on the historical utilization and cache information of the ISLs. PAR algorithm is designed to set a destination with a distributed minimum hop path.

1.5 Dynamic Detection Routing Algorithm

DDRA[5] is a typical routing algorithm based on a virtual topology strategy; the idea is to determine whether a link is normal by the ACK (Acknowledge) and periodic detection of the number of data packets in a link's unexpected situation and make appropriate adjustments in

time. The advantage of this algorithm is a low time delay, mainly when an unexpected condition occurs. However, it has the same disadvantage as virtual topology, the large number of time slices may be generated, and the computation of the routing algorithm will increase the complexity.

1.6 Datagram Routing Algorithm

the DRA[6][7] (Datagram Routing Algorithm) is a typical virtual node routing algorithm. It is assumed that the whole earth's surface is covered by logical locations of satellites. These logical locations of a satellite S [p,s] are not moving, always filled with the nearest satellite. Where p is the orbit number and s is the satellite number, which will be taken over by the successor satellite. The routing algorithm does not consider the satellite movement and considers these logical locations as hops. DRA routing table is derived from the precalculation of the ground station and sent to the constellation. However, when a satellite node or link fails, the performance and efficiency of the algorithm dramatically reduce, and network robustness will decrease because satellite nodes cant update the routing table on-board.

1.7 IP Routing in satellite constellation networks

The idea behind IP routing[8] topology is to extend (TCP/IP) to satellite networks. The first experiment was done with SATNET in the 1970s, and it worked well over satellite links. However, this topology faced some problems like Routing table management: table size and complexity is an obstacle to preform IP routing on-board. Performance: space-qualified computing hardware usually has the low processing power, and this kind of algorithms can produce a lag in the system.
Satellite movement: Satellite network is quite different from the ground network; continuous satellite movement requires continuously routing table updates.
Satellite constellations global coverage significantly extended space communication and has excellent potential for the development and applications of next-generation space technology. However, in the process of establishing a satellite network the main problem is routing technology although there is no effective dynamic satellite routing system available nowadays.

This paper presents a dynamic onboard routing algorithm. The idea behind onboard routing is to make network configurations, routing table updates and path calculation fully autonomous and invulnerable to node and inter-satellite communication link failure. As a result, the algorithm can serve more than 400 satellites.

2 Material and methods

In the proposed dynamic routing algorithm, each satellite in the network must store all satellite TLE datasets (a two-line element list of orbital elements for Earth-orbiting objects at a given point in time) in a file. Create a network map and find the shortest path based on the least number of satellites. TLE entries can be preprogrammed prior to start-up. However, they are updated in orbit in the same way that computer network changes are shared between routers on the ground: using special service messages. Based on TLE entries, each satellite knows its neighbor list.
Our simulations were conducted using data from the NORAD TLE database in order to test this approach. Each satellite in the network is a Node class object consisting of two parameters: the first is a TLE object with all orbit parameters, and the second is a list of neighbors.

The algorithm was programmed in C/C++ and python programming languages and calculates satellite position and separation distance with the following sequence:

1. SGP4 [9] The orbit propagator is employed to calculate the satellite's Cartesian position and velocity in the ECI coordinate system, which is centered on Earth and is associated with Universal Time Coordinated (UTC).
2. A transformation from ECI to true geographical coordinates.
3. A transformation from geodetic coordinates to ECEF (Earth-Centered, Earth-Fixed) coordinates [10].
4. Separation distance between satellites is calculated using two ECEF coordinates.

The power of transmission, frequency, and baud rate all affect the communication's range. Since it is a variable parameter that is simple to change, our simulation range was predicted to be 3000 KM. The depth-first search (DFS) method is used by the searching algorithm to delve deep into the network until it reaches the target or returns "Not Found" if the destination is outside of the local network, according to the network topology that is generated based on separation distance and satellite geolocation.

2.1 Network configuration

Network construction: Following the determination of the location and separation distance between each network node. Each node updates the neighbors list by adding any additional nodes that are within the communication range. After completing this phase, the algorithm has complete information of the network architecture.

2.2 Data structure

Nodes within the communication range are listed in the neighbors list. In Python, lists were used to keep objects, and each object is a node containing information about its orbital parameters and a list of its neighbors. Due to the existence of many copies of the same items in various lists, this technique resulted in a large memory use. Using pointers instead of objects to overcome this problem in C/C++ resulted in a significant decrease in memory utilization, which also had an impact on computing performance.

3 Theory and calculation

In order to build network topology and determine pathways, dynamic routing uses a connection-oriented routing algorithm based on the satellite's geolocation. It takes less time to identify and construct network topology using this technique because it is not susceptible to node failure and that links failure is independent of announcements and acknowledgements. Additionally, unlike announcements or acknowledgments techniques, it can spot any change in the network (auto detection). Multiple search techniques were explored when the network architecture was built aboard.

3.1 Searching algorithms

3.1.1 Breadth-First Search

An algorithm used in an uninformed search technique is called breadth-first search. In the tree, do this to locate the appropriate node. The algorithm operates by traversing all nodes in great detail. It operates by extending descendent nodes at that level and searching from the root node. By increasing its nearby breadth, it then transfers to a different node. The procedure needs a lot of memory and time to run when the target nodes are at the end of the tree or graph. In order to implement the BFS algorithm, a first-in-first-out queue must be used. BFS techniques function without domain expertise.

An approach for traversing a network that starts at the root node and checks all neighboring nodes is called breadth-first search. Then it picks the nearest node and investigates every undiscovered node. Every node in the graph may be viewed as the root node when using BFS to traverse it.

3.1.2 Depth-First Search

Depth-first search is a recursive algorithm that uses the concept of backtracking. It involves an exhaustive search of all nodes, continuing if possible, otherwise backtracking. The word backtracking here means once you have moved forward and there are no more nodes on the current path, move backwards on the equivalent path to find the node to traverse. Visit all nodes on the current path until all unvisited nodes have been traversed, then select the subsequent path.

The main strategy of depth-first search is to dig into the graph as deeply as possible. Depth-first search examines the edges resulting from the last found vertex. Only edges leading to unexplored vertices are checked. When all edges have been explored, the search loops back until it reaches an unexplored neighbor. This process continues until all vertices reachable from the original source vertex are found. If there are unvisited vertices, depth-first search chooses the closest one to the goal as a new source and repeats the search starting from that vertex. The algorithm repeats the whole process until it finds every vertex. The algorithm takes care not to duplicate vertices, so each vertex is checked once. DFS uses a stack data structure to keep track of vertices.

3.1.3 Depth-First Search with sorted neighbors list

One way to optimize depth-first search is to use an adjacency list instead of an adjacency matrix. This can be done by storing a list of nodes that are adjacent to each other in an array. The advantage of this approach is that it reduces the memory requirements of the algorithm, as well as the time required to initialize the data structures. Additionally, using an adjacency list allows for more efficient search as the graph traversal becomes more direct since any given node is only considered once.

The sorting algorithm used is a hybrid stable sorting algorithm known as Timsort, which derived from merge sort and insertion sort. This algorithm is used to sort an array of numbers in ascending order. The time complexity of this algorithm is O(nlog(n)), which means it will take nlog(n) time to sort an array of n numbers. The main advantage of this algorithm is its stability. In addition, it is well suited for partially sorted arrays due to its fast performance on already sorted elements. Furthermore, Timsort utilizes adaptive algorithm design which helps it run faster on large datasets. Due to its stability and adaptive design, this algorithm is highly suitable for sorting applications where large data volumes are involved.

3.1.4 Depth-First Search with unsorted neighbors list

When we talk about pathfinding algorithms, one of the most commonly used is the A* algorithm. A* is used in a variety of applications, such as video games, to find the shortest path between two points. The way it works is by looking at a given node and its neighbors and then selecting the node that is closest to the goal.

However, there is a newer algorithm that is starting to gain traction in the pathfinding community. This algorithm is known as the Brezenham algorithm. Unlike A*, Brezenham does not sort the neighbors list. Instead, it selects the closest node to the destination in each iteration. This means that the algorithm is constantly getting closer to the goal, which makes it more efficient.

3.2 Network topologies

In its simplest form, a network topology is the arrangement of the elements of a communication network, whether these elements are nodes or connections. The topology of a network can be represented by a diagram, which can then be used to illustrate the flow of data between the various elements of the network. In this section, two different network topologies are proposed.

3.2.1 All neighboring nodes topology

This method allows the searching algorithms to acquire both long and short pathways since they take into account the entire neighbors list without any alterations. The algorithm may still select the route that is most appropriate for transmitting data even while it has access to both many long pathways and multiple short paths.

3.2.2 Possible neighboring nodes topology

In this method, the neighbors list is modified by the searching algorithms. On each step, it only takes into account the nearby nodes that are closer to the target than the present node. This enables the search engines to obtain only one kind of path—the short path. Even when the algorithm is given a number of short pathways, it may still decide which one is best for transmitting data.

4 Results

4.1 Breadth-First Search

With the PyOrbital module [11], the breadth-first approach was built in Python. It demonstrated the feasibility of creating network topologies onboard and the functionality of working with off-the-shelf onboard computers like the Beagle-bone Black (1 GHz single-core processor and 512 MB RAM). Additionally, it demonstrated that the algorithm could function as a background program on a Linux-based operating system, such as Kubos Linux, without interfering with the primary mission application. Iridium, Iridium NEXT, Global-star, Planet, and Spire are just a few examples of the Celestrak Norad two-line element data sets that were used in the simulation.

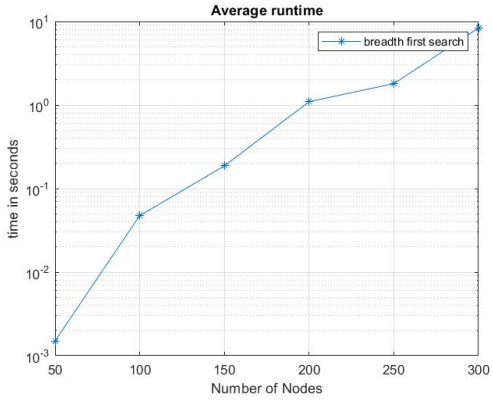

Figure 1: Runtime semi-log

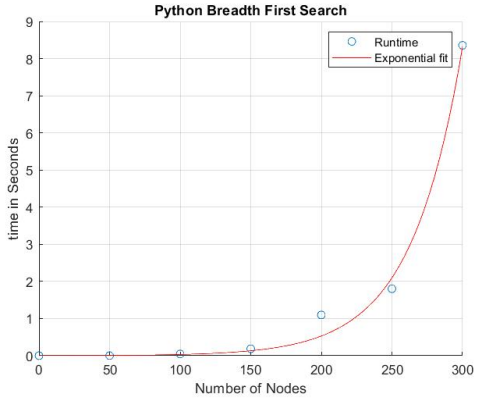

Figure 2: Exponential fitting

Number of Nodes	Average Runtime in seconds
50	0.0015
100	0.0474
150	0.1871
200	1.0960
250	1.8028
300	8.3585

Table 1: Average Runtime

The runtime of the breadth-first algorithm exponentially grows as the number of satellites in the network increases, as illustrated in Figs 1 and 2. as a result of the algorithm's high level of complexity and the use of a Python interpreter. The algorithm managed to construct a network of 300 satellites despite having a very long runtime, compared to big constellations. Although the runtime indicates that 200 satellites are still the maximum for this technique.

4.2 Depth-First Search

Employing the SGP4 orbit propagator [9], the depth-first search was developed in C/C++. It makes use of the same data sources and methodology as breadth-first search to build the network topology onboard. The depth-first search demonstrated how to use a recursive search function on a dynamic network architecture. This method was evaluated on an off-the-shelf onboard computer called the NUCLEO-L496ZG (80 MHz single-core processor and 320 KB RAM).

As previously indicated, there are several pairings of searching and sorting algorithms with network topologies.

4.2.1 Depth First Search in sorted & unsorted all neighboring nodes topology

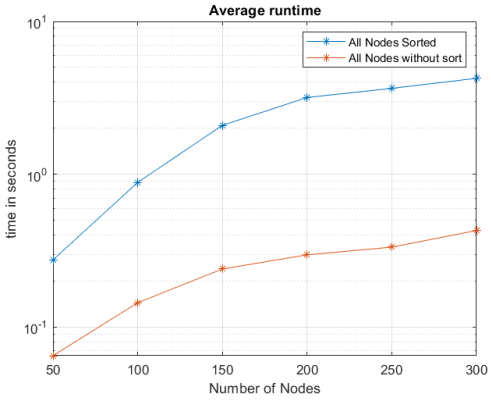

Figure 3: Runtime semi-log

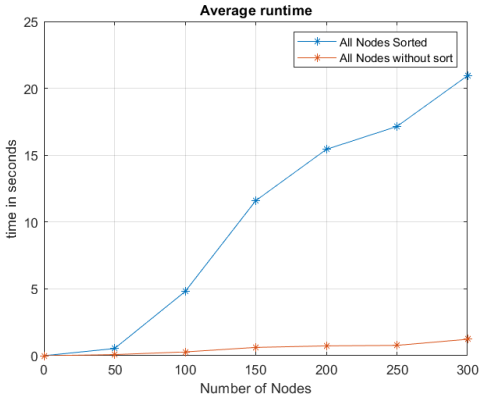

Figure 4: Average Runtime

Number of Nodes	Average Runtime
50	0.5375
100	4.8011
150	11.5958
200	15.4318
250	17.1480
300	20.9434

Table 2: All Neighboring Nodes Sorted Average Runtime

Number of Nodes	Average Runtime
50	0.2753
100	0.8848
150	2.0904
200	3.1804
250	3.6492
300	4.2472

Table 3: All Neighboring Nodes Unsorted Average Runtime

Both methods were evaluated using the identical hardware and networks, as seen in Figs. 3 and 4. The unsorted adjacent list technique, however, was able to provide the same outcomes with reduced runtime, computational complexity, and memory use. Nevertheless, due to the hardware limitations of the method, the maximum number of satellites is still 300.

4.2.2 Depth First Search in sorted & unsorted possible nodes topology

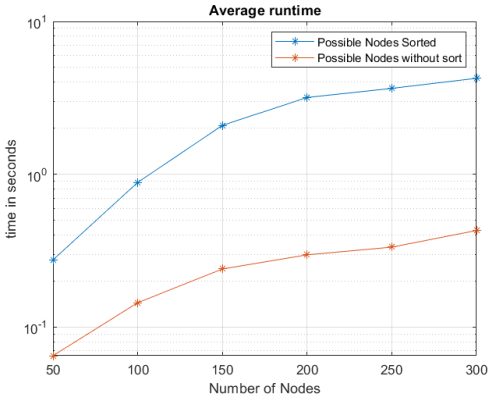

Figure 5: Runtime semi-log

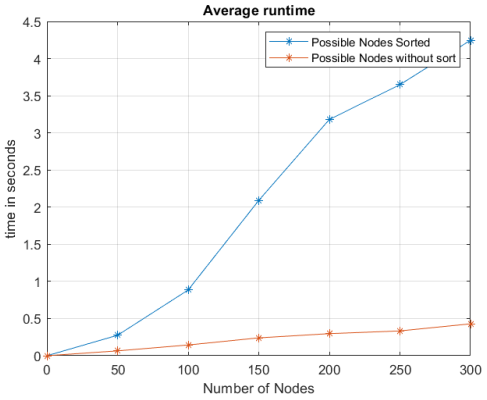

Figure 6: Average Runtime

Number of Nodes	Average Runtime
50	0.2753
100	0.8848
150	2.0904
200	3.1804
250	3.6492
300	4.2472

Table 4: Possible Neighboring Nodes Sorted Average Runtime

Number of Nodes	Average Runtime
50	0.0650
100	0.1444
150	0.2401
200	0.2970
250	0.3336
300	0.4290

Table 5: Possible Neighboring Nodes Unsorted Average Runtime

Both methods were evaluated using the identical hardware and networks.
Regardless of the network architecture, the unsorted technique can nevertheless achieve the same result as the sorted one while using less runtime, computational complexity, and memory. as seen in Figs. 5 and 6.

5 Discussion

The findings show that the network structure is influenced by the quantity of nodes, their spacing, and the orbital inclination. The network needs more than 150 satellites in Low Earth Orbit in order to create a single network where all nodes may transit data amongst one another.
The algorithm's runtime was impacted by utilizing the Beagle-bone Black and Python Interpreter, therefore the complexity and runtime were reduced by using the C compiler and NUCLEO-L496ZG.

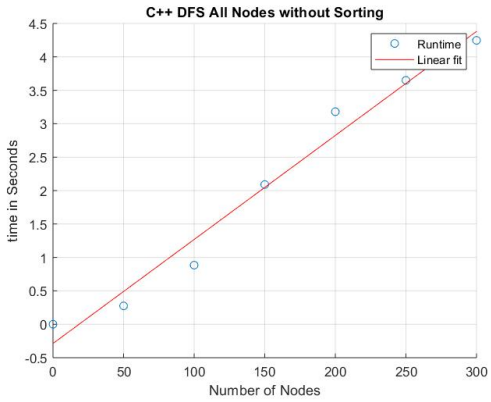

Figure 7: Linear fitting

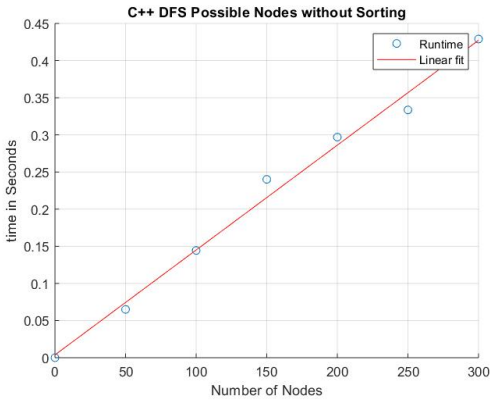

Figure 8: Linear fitting

However, by switching the search criterion from depth-first to breadth-first and from searching into a certain number of levels to recursion function. The runtime growth changed from exponential $O(c^n)$ Fig 2 to linear $O(n)$ Fig 7 and Fig 8.

The unsorted algorithms can provide the same outcomes as the sorted methods, as demonstrated in Figures 3 and 5. The fact that it is 100 times quicker was also shown. The sorted method won't be used in a real project and will be substituted with the unsorted ones based on the outcomes of prior experiments and tests.

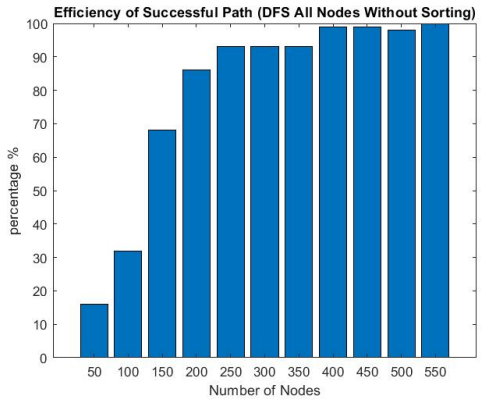

Figure 9: Algorithm Efficiency

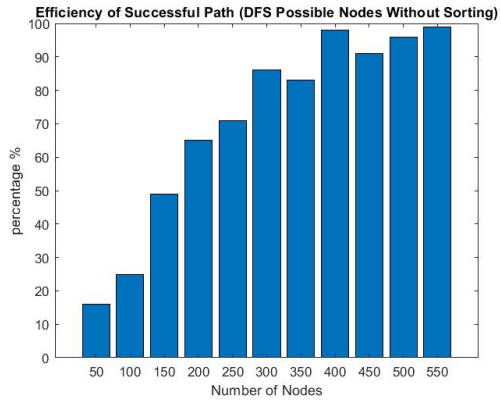

Figure 10: Algorithm Efficiency

The ability of algorithms to effectively acquire pathways between two satellites while using the same network architecture, number of satellites in the network, sources, and destinations serves as a measure of the algorithms' effectiveness. The technique that takes into account the whole network topology, as illustrated in Figs. 9 and 10, is more efficient but requires a longer runtime than the approach that just takes into account the nodes that are closest to the destination, which demonstrated that it is less efficient in small satellite constellations.

6 Conclusion

The depth-first search in all surrounding nodes performs better with small constellations (50–350 satellites), but the depth-first search in potential nodes only performs better with big constellations (greater than 400 satellites). The network application and the constellation's number of satellites are therefore taken into account while selecting the right algorithm.
The algorithm can, moreover, operate more than 400 nodes with great productivity, a short running time, little memory usage, and little processing power. As a result, the processing resources may be more effectively used and the usefulness of satellites is increased. The algorithm can be run on a separate thread from the primary mission program.

References

1. Werner, M. A dynamic routing concept for ATM-based satellite personal communication networks. *IEEE journal on selected areas in communications* **15,** 1636–1648 (1997).
2. Chang, H. S. *et al.* FSA-based link assignment and routing in low-earth orbit satellite networks. *IEEE transactions on vehicular technology* **47,** 1037–1048 (1998).
3. Jianjun, B., Xicheng, L., Zexin, L. & Wei, P. *Compact explicit multi-path routing for LEO satellite networks* in *HPSR. 2005 Workshop on High Performance Switching and Routing, 2005.* (2005), 386–390.
4. Korcak, O. & Alagoz, F. *Analysis of priority-based adaptive routing in satellite networks* in *2005 2nd International Symposium on Wireless Communication Systems* (2005), 629–633.
5. Tan, H. & Zhu, L. *A novel routing algorithm based on virtual topology snapshot in LEO satellite networks* in *2014 IEEE 17th International Conference on Computational Science and Engineering* (2014), 357–361.
6. Ekici, E., Akyildiz, I. F. & Bender, M. D. *Datagram routing algorithm for LEO satellite networks* in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)* **2** (2000), 500–508.
7. Ekici, E., Akyildiz, I. F. & Bender, M. D. A distributed routing algorithm for datagram traffic in LEO satellite networks. *IEEE/ACM Transactions on networking* **9,** 137–147 (2001).
8. Wood, L., Clerget, A., Andrikopoulos, I., Pavlou, G. & Dabbous, W. IP routing issues in satellite constellation networks. *International Journal of Satellite Communications* **19,** 69–92 (2001).
9. Lee, B.-S. NORAD TLE conversion from osculating orbital element. *Journal of Astronomy and Space Sciences* **19,** 395–402 (2002).
10. Hofmann, B., Lichtenegger, H. & Collins, J. GPS theory and practice. *Springer Wien NewYork* (2001).
11. Pytroll. *Pyorbital is a python package to compute orbital parameters for satellites from TLE files as well as astronomical parameters of interest for satellite remote sensing.* `https://pyorbital.readthedocs.io/en/latest/`.