Abstract—Advances in space exploration have led to an explosion of tasks. Conventionally, these tasks are offloaded to ground servers for enhanced computing capability, or to adjacent low-earth-orbit satellites for reduced transmission delay. However, the overall delay is determined by both computation and transmission costs. The existing offloading schemes, while being highly-optimized for either costs, can be abysmal for the overall performance. The computation-transmission cost dilemma is yet to be solved.

In this paper, we propose an adaptive offloading scheme to reduce the overall delay. The core idea is to jointly model and optimize the transmission-computation process over the entire network. Specifically, to represent the computation state migrations, we generalize graph nodes with multiple states. In this way, the joint optimization problem is transformed into a shortest path problem over the state graph. We further provide an extended Dijkstra’s algorithm for efficient path finding. Simulation results show that the proposed scheme outperforms the ground and one-hop offloading schemes by up to 37.56% and 39.35%, respectively on SpaceCube v2.0.

Index Terms—LEO satellite network, task offloading, adaptive scheduling, state graph, delay optimization.

I. INTRODUCTION

TASK offloading is an important issue for space applications such as remote sensing (RS) [1]. Advances in space technology have accelerated the exploration and exploitation of space resources. Meanwhile, massive tasks with different data volumes and computational requirements make delay reduction more challenging. Conventionally, these tasks are routed via spacecraft, such as low-earth-orbit (LEO) satellites, to ground servers near the destination for computation [2], [3]. With the development of onboard computing, these tasks are also offloaded to LEO satellites within one hop for computation, the results of which are then routed to the destination [4].

However, these conventional offloading schemes are susceptible to a transmission-computation cost dilemma, because their extreme offloading targets prevent them from coping with task diversity. Specifically, the ground offloading scheme reduces computation delay by the powerful computing capability of ground servers. However, for tasks with large data volumes, the transmission delay can be high, as large amounts of raw data need to be transmitted to distant destinations. The one-hop offloading scheme reduces transmission delay by performing onboard computing. However, for tasks with high computational requirements, the computation delay can be high because computing resources within one-hop range are limited. Briefly, conventional offloading schemes are devoted to reducing transmission or computation delay, but may lead to excessive delay for the other; thus creating a transmission-computation cost dilemma.

An intuitive solution to escape this dilemma is extending the offloading targets. LEO satellites located between the source and destination could be a promising solution. This is because offloading tasks to these satellites not only achieves lower transmission delay than the ground offloading scheme by performing onboard computing, but also provides more computing resources than the one-hop offloading scheme. Thus, LEO satellites beyond one-hop range hold the promise of avoiding both high transmission delays and computation delays existing in conventional schemes.

Novel targets bring novel challenges. While offloading tasks to LEO satellites beyond one hop may capture more computing resources to reduce the computation delay, multi-hop transmission of raw data increases the transmission delay. It is difficult to balance the transmission and computation costs to minimize the overall delay. To address this challenge, the transmission and computation processes should be optimized jointly.

In this paper, we propose a state-graph-based task offloading scheme that adaptively selects the optimal offloading target according to the characteristics of tasks. By integrating all potential offloading targets, and representing both computation and transmission processes in the state graph, the proposed scheme jointly optimizes computation and transmission delays. In this way, the optimal offloading target and path with the minimal overall delay of any task can be obtained. The main contributions of this paper are summarized as follows.

• We propose a state graph as a mathematical optimization tool. It can represent migrations between different nodes and between different states of the same node. Additionally, we propose a low-complexity extended Dijkstra’s algorithm that finds the shortest path in the state graph.
• We enrich the offloading targets with LEO satellites beyond one-hop range to resolve the transmission-computation cost dilemma. To this end, we construct the network as a state graph with two states (uncomputed v.s. computed) and transform the task offloading problem into a shortest path problem in the state graph.
• We propose an adaptive task offloading scheme which applies the extended Dijkstra’s algorithm to the constructed state graph. On SpaceCube v2.0, it outperforms the ground offloading scheme and the one-hop offloading scheme by up to 37.56% and 39.35%, respectively.

In rest of this paper, the system model is introduced in Section II. Section III propose a state graph and an extended Dijkstra’s algorithm. An adaptive task offloading scheme is proposed in Section IV. Performance evaluations are given in Section V. Conclusions are drawn in Section VI.
II. SYSTEM MODEL

Due to the superiority in latency, cost, development cycle, etc., LEO satellite networks are deemed as the most prospective satellite communication system [5]. Existing onboard computing systems can provide up to thousands of Giga floating-point operations per second (GFLOPS) of computing capability [6], making them a promising solution for space computation offloading.

In this paper, we investigate the space task offloading problem in the LEO-satellite-based multi-tier network shown in Fig. 1. Without loss of generality, we take a Walker Star constellation [7] with polar orbits as an example of LEO satellite networks. All circular orbits are evenly distributed over 180° range, traveling north on one side of the Earth, and south on the other side. They cross each other only over the North and South poles. LEO satellites are uniformly distributed over each orbit. Each LEO satellite has four inter-satellite links (ISLs) with its neighbors where two are intra-plane and two are inter-plane. The ISLs in cross-seam and Polar Regions are switched off due to high dynamic motions.

In the above definition, \( V = \{V_{k,s} \mid k \in K, s \in V\} \) is a set of nodes related to a set of ordered states, where \( V_{k,s} \) represents the node \( s \) in state \( k \). \( K \) is the set of state.

\( E = \{E_k \mid k \in K\} \) is a set of functions related to multiple states, where \( E_k(s,s')(s,s' \in V) \) represents the function from node pair (i.e., edge) \((V_{k,s}, V_{k,s'})\) in state \( k \) to \( \mathbb{R} \) (i.e., \( E_k : V \times V \to \mathbb{R} \)).

\( T = \{T_k \mid k \in K\} \) is a set of functions related to multiple states, where \( T_k(s)(s \in V) \) represents the function from node pair \((V_{k,s}, V_{k+1,s})\) to \( \mathbb{R} \) (i.e., \( T_k : V \to \mathbb{R} \)).

Each node in \( G_S \) has \( |K| \) states. Thus, there are \( |K| - 1 \) functions for calculating the edge weights between nodes in the same state and \( |K| \) functions for calculating the edge weights between adjacent states of the same node.

B. Finding the Shortest Path

Since state graphs have some differences compared with simple graphs, we first give a definition of paths in state graphs, and then propose an extended Dijkstra’s algorithm which finds the shortest path in state graphs.

Definition 2 (Path): A path \( P_S \) in the state graph \( G_S = (V, E, T) \) is a finite sequence of distinct edges \((e_1, e_2, \ldots, e_l)\) \( e_l = (V_{k_l,s_l}, V_{k_{l+1},s_{l+1}}), l \in \mathbb{Z}^+ \) for which there is a sequence of nodes \((V_{k_1,s_1}, V_{k_2,s_2}, \ldots, V_{k_{l+1},s_{l+1}})\) subjecting to the following constraints:

- \( \forall l \in \mathbb{Z}^+ (k_{l+1} \equiv k_l); \)
- \( \forall i, j \in \{1, 2, \ldots, l + 1\} (i \neq j \land k_i = k_j \Rightarrow s_i \neq s_j). \)

In the above definition, \( V_{k_1,s_1} \) and \( V_{k_{l+1},s_{l+1}} \) are the source and destination nodes of path \( P_S(V_{k_1,s_1}, V_{k_{l+1},s_{l+1}}) \). \( V_{k_1,s_1} \) \( k_{l+1}, s_{l+1} \) are the head and tail of edge \( e_l = (V_{k_1,s_1}, V_{k_{l+1},s_{l+1}}) \). The first constraint indicates that for any edge in \( P_S \), the state index of its head is not greater than that of its tail. The second constraint indicates that any two nodes of \( P_S \) must be different from each other. In other words, two nodes of the same path can be two different nodes belonging to \( V \) in the same state or can be the same node belonging to \( V \) in different states.

Accordingly, the path length of \( P_S(V_{k_1,s_1}, V_{k_{l+1},s_{l+1}}) \) can be calculated as \( L(P_S) = \sum_{e_l \in P_S} \omega_l \), where \( \omega_l \) is the weight of \( e_l \) and can be calculated as:

\[
\omega_l = \begin{cases} 
E_k(s_l, s_{l+1}), k_l = k_{l+1}, \\
T_k(s_l), k_l \neq k_{l+1},
\end{cases}
\] (1)
Algorithm 1: The Extended Dijkstra’s Algorithm

Data: The state graph $G_S = (\mathbb{V}, \mathbb{E}, T)$ and the state set $K$.
Data: The source node $u$ and the destination node $v$.
Result: The shortest path $\pi$ from $V_{1,u}$ to $V_{|K|,v}$ and its length $L_\pi$.

1. Initialize $Q = \emptyset$ and $\pi \leftarrow \emptyset$;
2. foreach $V_{k,s} \in \mathbb{V}$ do
   3. $D[k,s] \leftarrow \infty$;
   4. $P[k,s] \leftarrow$ undefined;
   5. $Q \leftarrow Q \cup \{V_{k,s}\}$;
3. end
4. $D[1,u] \leftarrow 0$;
5. while $Q \neq \emptyset$ do
   6. $\theta \leftarrow \infty$;
   7. for $V_{k,s} \in Q$ do
      8. if $D[k,s] < \theta$ then
         9. $\theta \leftarrow D[k,s]$;
         10. $k^* \leftarrow k$;
         11. $s^* \leftarrow s$;
      12. end
   13. end
   14. $Q \leftarrow Q \setminus \{V_{k^*,s^*}\}$;
15. for $V_{k,s} \in Q$ do
   16. if $D[k,s] < \theta$ then
      17. $\gamma \leftarrow D[k^*,s^*] + E_{k^*,s^*}(s^n, n)$;
      18. if $\gamma < D[k^n, n]$ then
         19. $D[k^n, n] \leftarrow \gamma$;
         20. $P[k^n, n] \leftarrow V_{k^*,s^*}$;
      21. end
   22. end
   23. if $k^* + 1 \leq |K|$ then
      24. $\gamma \leftarrow D[k^*,s^*] + E_{k^*,s^*}(s^n, n)$;
      25. if $\gamma < D[k^n, n]$ then
         26. $D[k^n, n] \leftarrow \gamma$;
         27. $P[k^n, n] \leftarrow V_{k^*,s^*}$;
      28. end
   29. end
30. $L_x \leftarrow D[|K|, v]$;
31. $V_{k,x} \leftarrow P[|K|, v]$;
32. while $V_{k,x} \neq V_{1,u}$ do
   33. $\pi \leftarrow \pi \cup \{V_{k,x}\}$;
   34. $V_{k,x} \leftarrow P[k,x]$;
35. end

As Algorithm 1 shows, the state graph $G_S = (\mathbb{V}, \mathbb{E}, T)$, the corresponding state set $K$, the source node $u$, and the destination node $v$ are the input data. Algorithm 1 outputs the shortest path $\pi$ from $V_{1,u}$ to $V_{|K|,v}$ and its length $L_\pi$.

Algorithm 1 contains three parts. The first part is the initialization (line 1 to line 7). $Q$ is the unvisited node set. $D$ is the set recording the tentative distance value, i.e., the length of the shortest path discovered so far) between $V_{1,u}$ and each node in $\mathbb{V}$. The tentative distance value is set to zero for our initial node and to infinity for all other nodes. In addition, $P$ is the set of parent nodes of each node in $\mathbb{V}$.

The second part is the shortest path searching (line 8 to line 34). First, we find the node $V_{k^*,s^*}$ whose tentative distance value is the minimum among all unvisited nodes (line 9 to line 16). Then, delete the above selected node from the unvisited nodes set $Q$ (line 17). Next, we consider all the unvisited neighbors in state $k^*$ of $V_{k^*,s^*}$ and calculate their tentative distances through the selected node (line 18 to line 26). By comparing the newly calculated tentative distance to the one currently assigned to the neighbor, the smaller one is assigned to it. If the tentative distance of a neighbor is updated, its parent node needs to be changed to $V_{k^*,s^*}$ accordingly. Similarly, in line 27 to line 33, we consider the corresponding node of $V_{k^*,s^*}$ in state $k^* + 1$ and calculate the tentative distances of $V_{k^*+1,s^*}$ through $V_{k^*,s^*}$. The smaller of the newly calculated and the currently assigned tentative distance is assigned to $D[k^* + 1, s^*]$. If the tentative distance is updated, its parent node of $V_{k^*+1,s^*}$ needs to be changed to $V_{k^*,s^*}$ accordingly.

The third part is the output generation part (line 35 to line 40). Line 35 outputs the length of the shortest path between $V_{1,u}$ and $V_{|K|,v}$. From line 36 to line 40, by searching the parent node set iteratively, the shortest path between $V_{1,u}$ and $V_{|K|,v}$ is output.

C. Algorithm Correctness

In the following, we prove that the extended Dijkstra’s algorithm is correct by induction. Let $D[k,s]$ be the distance label found by the algorithm and let $\delta[k,s]$ be the shortest path distance from $V_{1,u}$ to $V_{|K|,v}$. We want to show that $D[k,s] = \delta[k,s]$ for every node $s$ in any state $k$ at the end of the algorithm, showing that the algorithm correctly computes the distances. We prove this by induction on $|R| (R = \mathbb{V} \setminus Q)$ via the following lemma.

**Lemma 1:** For each $V_{k,s} \in R$, $D[k,s] = \delta[k,s]$.

**Proof:** Base case ($|R| = 1$): Since $R$ only grows in size, the only time $|R| = 1$ is when $R = \{V_{1,u}\}$ and $D[1,u] = 0 = \delta[1,u]$, which is correct.

**Inductive hypothesis:** Let $V_{r,w}$ be the next node added to $R$. Let $R' = R \cup \{V_{r,w}\}$. Our I.H. is: for each $V_{k,s} \in R'$, $D[k,s] = \delta[k,s]$.

**Using the I.H.:** By the inductive hypothesis, for every node in $R'$ that isn’t $V_{r,w}$, the distance label is correct. We need only show that $D[e,w] = \delta[e,w]$ to complete the proof.

Suppose a contradiction that the shortest path from $V_{1,u}$ to $V_{r,w}$ is $O$ and has length $L(O) < L(e,w)$. $O$ starts in $R'$ and at some leaves $R'$ (to get to $V_{r,w}$ which is not in $R'$).

**Case 1:** Let $e_{x,y} = (V_{k,x,s}, V_{k,y,s})$ (for $k_y < k_x \in \{0, 1\}$) be the first edge along $O$ that leaves $R'$. Let $O_x$ be the sub-path between $V_{1,u}$ and $V_{k_x,s_x}$ of $O$. Clearly: $L(O_x) + L(e_{x,y}) \leq L(O)$. (If $k_x = k_y$, $L(e_{x,y}) = E_{k_x}(s_x, s_y)$; otherwise, $L(e_{x,y}) = T_{k_x}(s_y)$.) Since $D[k_x, s_x]$ is the length of the shortest path from $V_{1,u}$ to $V_{k_x,s_x}$ by the I.H., $D[k_x, s_x] \leq L(O_x)$, giving us $D[k_x, s_x] + L(e_{x,y}) \leq L(O_x) + L(e_{x,y})$. Since $V_{k_x,s_x}$ is adjacent to $V_{k_y,s_y}$, $D[k_y, s_y]$ must have been updated by the algorithm, so $D[k_y, s_y] \leq L(k_x, s_x) + L(e_{x,y})$. Finally, since $V_{r,w}$ was picked to be the next node added to $R'$ by the algorithm, $V_{r,w}$ must have the smallest distance label: $D[e,w] \leq D[k_y, s_y]$. Combining these inequalities in reverse order, a contradiction $D[k_x, s_x] < D[k_y, s_y]$ is formed. Thus, no such shorter path $O$ must exist. So $D[k,s] = \delta[k,s]$.

This lemma shows the algorithm is correct by applying the lemma for $R = \emptyset$.

D. Algorithm Complexity

The computational complexity of the proposed extended Dijkstra’s algorithm on state graph $G_S = (\mathbb{V}, \mathbb{E}, T)$ mainly resides in lines 8–34. Line 8 indicates that lines 9–33 will be executed $|Q|$ times. Since there are $\Theta(|\mathbb{V}|)$ nodes in set $Q$, the complexity of executing lines 10–16 $|Q|$ times is $\Theta(|\mathbb{V}|^2)$. Lines 18–19 restrict $E_{k,v}(V_{k,s}, V_{k,s'})$ to the edges connecting $V_{k,s}$ and its adjacent nodes in the same
state; thus, the complexity of executing lines 18–26 \(|Q|\) times is \(\Theta(|K| \cdot |E|)\). Since there’s no loop in lines 27–33, the complexity of executing lines 27–33 \(|Q|\) times is \(\Theta(|V|)\). According to the definition of \(G_S\), \(|V| = |V| \cdot |K|\) and \(|E| = |V|^2\). Therefore, the computation complexity of the proposed algorithm is \(\Theta(|K|^2 \cdot |V|^2)\).

IV. ADAPTIVE OFFLOADING WITH STATE GRAPH

In this section, we utilize the mathematical tools proposed in Section III to solve the space task offloading problem. As the optimal offloading targets can be different for tasks with diverse transmission and computational requirements, we propose an adaptive offloading scheme based on state graph.

Definition 3 (Task): Any task \(\tau\) has five attributes, whose set is \(\Lambda_\tau = (u,v,t,C,N,N')\). \(u\) and \(v\) are the source and destination, respectively. \(t\) is the instance when \(\tau\) is generated. \(C\) is the computational requirement of \(\tau\). \(N\) and \(N'\) are data volumes of \(\tau\) before and after computation, respectively. Tasks are the smallest unit of transmission and computation.

As stated in Section I, computational states need to be distinguished to select the optimal offloading target. Therefore, we construct the network shown in Fig. 1 as a state graph with two states. By defining the two states as the uncomputed state and computed state, the task computation process can be expressed as a migration between the above two states. Next, we introduce the state graph \(G_S = (\bar{V}, \bar{E}, T)\) constructed based on the network shown in Fig 1 in detail.

\[
\bar{V} = \{V_{k,s} | k \in \{1, 2\}, s \in V \cup u \cup v\},
\]

\(1\) and \(2\) represent the uncomputed and computed states, respectively. \(V\) is the set of LEO satellites, \(u, v\) are the source and destination of task \(\tau\), respectively.

\[
\bar{E} = \{E_k | k \in K\},
\]

\(E_k(s,s')(s, s' \in V \cup u \cup v)\) satisfies

\[
\begin{align*}
\bar{N} &= \int_{t_s}^{t_s + \bar{E}_k} R(s, s', t) dt, \quad k = 1, \\
\bar{N'} &= \int_{t_s}^{t_s + \bar{E}_k} R(s, s', t) dt, \quad k = 2,
\end{align*}
\]

where \(t_s\) is the instant that task \(\tau\) arrives at node \(s\). \(R(s, s', t)\) is the available data transmission rate of edge \(e_{s,s'}\) at instant \(t\), which is independent from states. Depending on the node types of \(s\) and \(s'\), \(e(s, s')\) can be an ISL or satellite-to-ground link (SGL). If \(s\) and \(s'\) are not communicable\(^2\) (i.e., \(s \not\leftrightarrow s'\)), \(R(s, s', t) = 0\); otherwise, \(R(s, s', t) \in [0, R_{\max}]\), where \(R_{\max}\) is the maximum data transmission rate of \(e_{s,s'}\).

\[
\bar{T} = \{T_k | k \in K\}, \quad T_k(s)\] satisfies

\[
\bar{C} = \int_{t_s}^{t_s + \bar{T}_k} C(s, t) dt,
\]

where \(C(s, t)\) is the available computing capability of node \(s\) at instant \(t\). Since the source node does not perform computation, when \(s = u\), \(C(s, t) = 0\). Additionally, ground servers (such as supercomputers) usually have large amounts of computing resources so that they can complete the computation in a very short time; therefore, when \(s = v\), \(C(s, t) = \infty\). In other cases, \(C(s, t) \in [0, C_{\max}]\), where \(C_{\max}\) is the maximum computing capability of each satellite.

In other words, \(E_k(s, s')\) represents the edge weight between different nodes in the same state, whose value equals the delay of transmitting task \(\tau\) from node \(s\) to \(s'\); whereas, \(T_k(s)\) represents the weight of edge from a node in the uncomputed state to itself in the computed state, whose value equals the delay of computing task \(\tau\) on node \(s\). In this way, both the transmission and computation processes can be represented in the state graph. It should be noted that, we fit the continuous integration to our discrete model with discretization, where minor error may be introduced.

In the state graph above, the task offloading problem is transformed into a shortest path problem. By performing the extended Dijkstra’s algorithm on the constructed stated graph, the transmission and computation delays are jointly optimized. In this way, the shortest path, which contains the optimal offloading target, is obtained. Since the constructed state graph contains both conventional and extended offloading targets, the optimal offloading decision can be made adaptively according to the characteristics of tasks.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed adaptive offloading scheme and compare it with the ground and one-hop offloading scheme.

In this section, the Walker constellation has 8 orbits, each with 16 satellites, located at an altitude of 500 km. Based on the state-of-the-art techniques, \(R_{\max}^{\text{SL}}\) and \(R_{\max}^{\text{SGL}}\) are set to 5 Gbps [8] and 1 Gbps [9], respectively. Unless otherwise stated, the onboard computing platform is SpaceCube v2.0 [10] whose maximum computing capability is 200 GFLOPS. The task arrival of each area in space is a Poisson stochastic process whose arrival rate is estimated based on the network connection data from Internet Census [11]. The sum of arrival rates in space is set to 1000. The destination locations of tasks are randomly generated. \(N'\) is set to 16 bits.

Fig. 2: (a) The overall delay versus the data volume \(N\) and computational requirement \(C\) of tasks. (b) The offloading scheme which achieves the lowest overall delay.
scheme decreases the overall delay by 37.56% and 39.35% compared with the ground offloading scheme and the one-hop offloading scheme, respectively.

From the dimension of data volume, the proposed scheme degrades to the ground offloading scheme when $N$ is small and to the one-hop offloading scheme when $N$ is large. When $N$ is small, the advantage of computing with powerful ground servers outweighs the disadvantage of transmitting raw data across the network. When $N$ is large, the cost of transmitting raw data increases dramatically, the advantage of reducing transmission delay by offloading tasks to satellites within one-hop range outweighs the disadvantage that these satellites can only provide limited computing capability. From the dimension of computational requirement, the proposed scheme degrades to the one-hop offloading scheme when $C$ is small. In this situation, satellites within one-hop range are already able to provide sufficient computing resources to ensure that the saved transmission delay outweighs the increased computation delay. Except for these extreme cases, the proposed scheme outperforms the benchmark schemes, whose reasons will be introduced in the analyses of Fig. 3.

Fig. 2 (b) shows the offloading scheme which achieves the lowest overall delay for different tasks. It can be concluded that the proposed adaptive offloading scheme outperforms the benchmark schemes over a wide range of parameters (i.e., the red area). In the blue and green area, the proposed degrades to the ground offloading scheme and the one-hop offloading scheme, respectively.

![Fig. 3: Breakdown of the overall delay.](image)

Fig. 3 shows the breakdown of the overall delay of the proposed and benchmark offloading schemes. Here the transmission/computation delay includes the waiting delay between the arrival instant of the corresponding node and the start of transmission/computation, respectively.

The proposed scheme outperforms the ground offloading scheme because the data volume is greatly reduced after the onboard computation. As shown in Fig. 3, the transmission delays on ISL and SGL of the proposed scheme is much lower than the ground offloading scheme. In the meanwhile, the increase in computation delay is less than the decrease in transmission delay. The proposed scheme outperforms the one-hop offloading scheme because the proposed scheme can utilize computing resources in a larger area to relieve local overloads. As shown in Fig. 3, the computation delay of the adaptive offloading scheme is much lower than that of the one-hop offloading scheme. In the meanwhile, the increase in transmission delay on ISL is less than the decrease in computation delay.

| Onboard Computing Platform | Computing Capability (GFLOPS) | Improvements |
|---------------------------|------------------------------|--------------|
| Unibap iXS-100            | 127                          | +21% +159%   |
| SpaceCube v2.0             | 200                          | +42% +93%    |
| SpaceCube v3.0             | 590                          | +139% +15%   |
| MOOG V-Series Ryzen        | 1000                         | +256% +6%    |

Table I evaluates the performance of the adaptive offloading scheme over the benchmark schemes on several existing onboard computing platforms. (+0% means a tie, whereas +100% means saving half the time.) As the computing capability increases, the performance improvement over the ground offloading scheme increases and the improvement over the one-hop offloading scheme decreases. This is because the computation delay of LEO satellites decreases when their computing capability increases, making the disadvantage of limited onboard computing resources less severe. In the meanwhile, the increase in computing capability leads to a reduction in local overloads; thus, fewer tasks need to be offloaded to satellites beyond one-hop range for computation.

### VI. CONCLUSION

We have proposed an adaptive offloading scheme based on state graphs for space missions. The proposed scheme outperforms the benchmark schemes for being able to escape the transmission-computation cost dilemma by extending offloading targets with LEO satellites beyond one-hop range and implementing joint optimization of transmission and computation. Its low computational complexity offers the possibility of application in real networks.

### REFERENCES

[1] A. J. Plaza and C.-I. Chang, "High performance computing in remote sensing," CRC Press, 2007.
[2] Z. Lai, Q. Wu, H. Li, M. Lv, and J. Wu, “Orbitcast: Exploiting mega-constellations for low-latency earth observation,” in Proc. IEEE ICNP, 2021, pp. 1–12.
[3] X. Jia, T. Lv, F. He, and H. Huang, “Collaborative data downloading by using inter-satellite links in leo satellite networks,” IEEE Transactions on Wireless Communications, vol. 16, no. 3, pp. 1523–1532, 2017.
[4] Y. Jin, H. Yiao, and T. Mai, “Double auction game-based computing resource allocation in leo satellite system,” in 2020 International Wireless Communications and Mobile Computing (IWCMC), 2020, pp. 274–279.
[5] M. D. Staff, “Small spacecraft technology state of the art,” NASA, Ames Research Center, Mission Design Division Rept. NASA/TP-2015-216648/REVI, Dec. 2015.
[6] “Integrated avionics unit - datasheet,” https://www.moog.com/content/dam/moog/literature/Space_Defense/spaceliterature/avionics/moog-integrated-avionics-unit-datasheet.pdf, 2020.
[7] J. G. Walker, “Satellite constellations,” Journal of the British Interplanetary Society, vol. 37, p. 559, 1984.
[8] I. Del Portillo, B. G. Cameron, and E. F. Crawley, “A technical comparison of three low earth orbit satellite constellation systems to provide global broadband,” Acta Astronautica, vol. 159, pp. 123–135, 2019.
[9] B. Yost, S. Weston, G. Benavides, F. Krage, J. Hines, S. Mauro, S. Etchey, K. O’Neill, and B. Braun, “State-of-the-art small spacecraft technology,” 2021.
[10] “Spacecube v2.0 hybrid data processing system,” https://spacecube.nasa.gov/SpaceCube_v2_BriefPDF.pdf.
[11] C. Botnet, Internet Census 2012: Port scanning 0 using insecure embedded devices, 2012. [Online]. Available: http://census2012.sourceforge.net/paper.html.