FSET: Fast Structure Embedding Technique for Self-reconfigurable Modular Robotic Systems

Aliah Majed, Hassan Harb, Abbass Nasser, Benoit Clement

FSET: Fast Structure Embedding Technique for Self-Reconfigurable Modular Robotic Systems

Aliah Majed, Hassan Harb, Abbass Nasser and Benoit Clement

Abstract The rapid growth in communication technologies has lead to a new generation of robotics called as Modular Robotic System (MRS). The most crucial process in MRS is self-reconfiguration, which is regarded as the major challenge for such technology. Indeed, creating new morphology and behaviors manually is a time-consuming and costly process, especially when dealing with complex structures. In this paper, we have proposed a fast self-reconfiguration technique called FSET, i.e. Fast SET, dedicated to MRSs. Our proposed technique consists mainly in two stages: root selection and morphology formation. The final goal of these stages is to enhance the time cost to get new morphology of the traditional SET algorithm thus, ensure fast self-reconfiguration. The root selection stage selects a small number of modules in order to find the best tree roots that effects the topological conditions that leads to successful of the embedding process or not. The morphology formation stage uses the traditional SET algorithm to calculate the embedding truth table where the initial roots used are taken from the first stage. Finally, we show the efficiency of our mechanism through simulations on real scenario using M-TRAN, in terms providing a fast reconfiguration process in MRS and reducing the energy consumption of modules thus, increasing its lifetime.

A. Majed and A. Nasser
ICCS-Lab, Computer Science Department, American University of Culture and Education (AUCE), Beirut, Lebanon, e-mail: {firstnamelastname}@auce.edu.lb.

H. Harb
College of Engineering and Technology, American University of the Middle East, Kuwait, e-mail: hassan.harb@aum.edu.kw.

B. Clement
Lab-STICC, UMR CNRS 6585, Ensta-Bretagne University, France e-mail: benoit.clement@ensta-bretagne.fr.

1 Introduction

Since the early 1980's, the robotics industry has seen an increase in the production and manufacture of millions of robots of different types and missions. These robots are altering our everyday lives and assisting us in making our jobs more efficient and successful. Furthermore, the rapid advancement of technology in the new century has ushered in a new robotic era: Modular Robotic System (MRS).

The most important attribute of modular robots, regardless of their design, is their ability to reconfigure their morphology, a mechanism known as self-reconfiguration (See Fig. 1), which is regarded as the major challenge for such technology [1, 2]. Self-reconfiguration is the mechanism by which a modular robot's initial arrangement of modules (and thus initial form, also known as configuration) is changed into a target configuration. Indeed, MRS use self-reconfiguration to get new morphology and new behavior to performs specified tasks. However, creating new morphology and behaviors manually is a time-consuming and costly process, especially when dealing with complex structures. As a result, designing an algorithm that creates new morphology and new behavior from already existing modular robotic structures, takes a great attention from researchers and communities and became an active research field nowadays.

Therefore, to avoid the above-mentioned challenge, Structure Embedding Technique (SET) for the self-reconfigurable modular robotic system has been introduced. SET decides if a given modular robot structure can be embedded into another structure in order to form new morphology. In this paper, we present a fast SET, abbreviated FSET, a technique for modular robotic systems to minimize the delay to get new morphology. The proposed technique consists of a two-stage algorithm and can highly outperform the traditional SET in terms of the time cost to get new morphology and energy consumption of modules. The first stage of our technique, called root selection, has an objective to find the best roots of the initial trees, by selecting a small number of modules instead of the whole sets. The second stage, which is called morphology formation, uses the first stage's root of trees, to calculate the embedding truth table between modules in order to check the embeddability of the two modular robotic designs, resulting in the formation of new morphologies. Consequently, the calculation time cost of our FSET will highly minimize that of traditional SET due to the small number of the training modules used in the first stage and the low number of iteration loops needed in the second stage.

The rest of the paper is organized as follows. In section 2, we present related works in self-reconfiguration techniques used in MRS. Section 3 detail the SET mechanism. The system demonstration and the results are presented in Section 4. Finally, Section 5 concludes our paper and gives some perspectives.

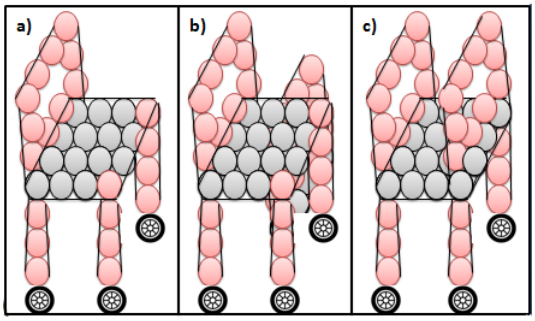

Fig. 1 Sample self-reconfiguration of about 52 3D modules from a chair into a stroller. (a) Chair initial configuration; (b) An intermediate configuration from the self-reconfiguration process; (c) Stroller goal configuration..

2 Related Work

Various methods for self-reconfiguration introduced for MRS can be found in the literature. Initially, researchers in the field of modular self-reconfigurable robotics based their focus on the hardware issue of creating metamorphic robots; then, research interest eventually arose in the generalized management of categories of these systems and various software frameworks were proposed; More recently, in the field of Self-Organizing Particle Systems, researchers have begun to show interest in what may be called a theoretical kind of metamorphic system.

In fact, three categories can be extracted from these three methods. We would then refer to self-reconfiguration algorithm, which Bottom-Up, Top-Down, and Theoretical, as well as[3]. Such methods vary by how they relate to their target execution platform, and therefore, by the nature of the constraints that make up the algorithm model used. We'll go through each of the three self-reconfiguration methods mentioned above, as well as the possible solutions.

From one hand, the authors of [4, 5, 6, 7, 8, 9] have proposed Bottom-Up methods to self-reconfiguration in MRS. The Bottom-Up method involves a focus on modular robotic hardware at first. They've come up with a range of module models, ranging from UCMs like Telecube [4] and Crystalline [5] to hybrids like M-TRAN [6] and Roombot [7]. bi-partite models like the Robotic Molecule [9] and I-Cubes [10, 11], as well as self-reconfigurable structures like Fracta [8]. There are several other models in the literature, but these are the ones that are used in the algorithms under consideration.

Due to the complexities of the geometry of hardware modules or their motion capacities, this approach credibly provides a very complicated self-reconfiguration preparation. Non-holonomic motion constraints are typical in these systems, complicating the reconfiguration method. Motion constraints can either be local: caused by module geometry and blocking constraints; or they can be global: like the connectivity constraint which specifies that the whole system's graph must stay connected at all times. Several strategies for achieving holonomy at the expense of

granularity have been devised, including using higher holonomy module aggregates (meta-modules) [10] or arranging the device into a porous structure [9] from which modules can flow unconstrainedly (a scaffold). Although the kinematics in the Bottom-Up method are typically more complicated, modules are more likely to expect a greater understanding of their surroundings. Sensor data about their orientation, location in the system, neighborhood, and other factors are used to produce these environmental information.

The authors in [9] have suggested a centralized solution for their bipartite Molecule robot, depending on a three-level hierarchical planner. The highest level of preparation is task-level planning, which chooses a configuration that is suitable for the task at hand. It then admits on a motion plan for Molecules to turn the initial configuration into the target configuration using configuration planning. The configuration planner uses trajectory planning to shift individual modules to their target positions at a lower stage. They also implemented the aforementioned scaffolding principle to ensure that Molecules reached into the target configuration, but this increased the granularity of their device dramatically since a single scaffold tile consisted of 54 modules. While these early works using centralized planners laid the foundations for most of the field and implemented useful problem simplification techniques, they lack the robustness, scalability, and autonomy that self-reconfiguration needs. As a result, researchers transformed to the decentralized self-reconfiguration process, as we'll see below.

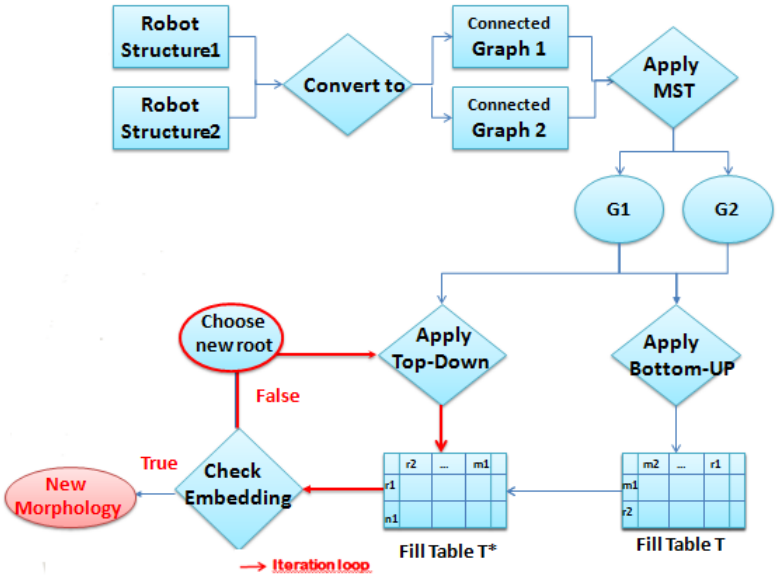

Fig. 2 SET Algorithm Flowchart.

On the other hand, a lot of Top-Down methods have proposed to self-reconfiguration in MRS [12, 13, 14, 15, 16, 17]. The Top-Down method plays a critical role in constructing shape formulation techniques that aren't related to a particular hardware application and can be applicable to a variety of MSR in a generalized way.

The authors in [12] developed the Pixel meta-module framework for lattice-based modular robots, which could greatly simplify reconfiguration planning in large modular robots. The key idea is to split the reconfiguration problem into two tasks: planning and resource allocation. The former's job is to figure out the meta-module positions in the target configuration need to be filled next, while the latter's job is to figure out where the meta-modules that will fill that position should come from.

In [17] the authors presents a two-level hierarchical approach to completely generic algorithms (for any architecture), in which the planning problem is formulated as a distributed Markov Decision Process (MDP). An MDP is defined by the four-tuple S, A, T, R, where S is the set of states, represented by open positions to be filled by modules, and is equal to the number of faces of the modules; A is the set of actions, represented by the disconnection of a connector from a neighbor module and the reconnection to another, potentially using a different connector; T is a deterministic or stochastic transition function that determines the next action to take; R is the estimated reward, which is set to 1 to reduce the number of moves.

The authors overcome this MDP by introducing dynamic programming in a distributed environment using message passing. The MDP works at the planner's higher levels, deciding for each moving module which other modules and connectors should link to during the next time phase. The low-level planner then calculates the sequence of individual module movements that the mobile module can take to detach from its current neighbor and reconnect at its new anchor point. Modules look at the structure to make sure they aren't an articulation point in the system's graph, then determine if they are mobile or not and lock a portion of it during their motion if they are. Since several modules can lock the same part of the structure, they can shift in parallel, speeding up the reconfiguration phase. Designing an effective kinematic planner to serve as the transition function T is the most difficult part of this scenario.

The authors in [18] developed their PacMan self-reconfiguration algorithm for two-dimensional unit-compressible modules to three-dimensional structures. As compared to surface moving modules, one advantage of UCM is that they can migrate through the volume of the system, theoretically benefiting from a higher number of parallel motions and a shorter distance to their destination. The authors use a technique known as virtual relocation to transfer modules from one end of the configuration to the other, switching their identities as they compress and decompress along the path to their target point.

PacMan depends on a two-stage distributed planning algorithm in which: (1) modules locally calculate the difference between the current and target shapes to determine which modules should move; and (2) A suitable search (depth-first search) for a mobile module is carried out from the desired locations, with pellets dropped along the way to mark the route that the selected module should follow. Our proposed method consists of two phases, where it applies one of the bottom-up algo-

rithms in its first phase, while in the second phase it applies one of the top-down algorithms. Recognizing if two complete configurations are the same [19], detecting graph automorphisms[20], and recognizing similar substructures for efficient reconfiguration are all examples of existing work in graph representations of modular robots. Our work stands out by incorporating task implications on configurations and specifying criteria for replicating a design's capabilities by replicating its design.

3 FSET Technique

In the literature, one can find a huge number of self reconfiguration algorithms like SET, PacMan, scaffold-based etc. However, SET is one the most popular algorithms used in self reconfiguration. Unfortunately, traditional SET suffer from its huge calculation time cost needed to obtain the new morphology. In order to overcome this problem, we propose a new version of SET called FSET, Fast SET, which highly enhances the time cost of traditional SET. Our FSET consists of two stages, root selection and morphology formation stages, and calculate the embedding truth table according to the topological embedding condition. In the next sections, we first recall the traditional SET and its topological embedding conditions then we detail the two stages of our technique.

3.1 Recall of SET Algorithm

SET is one of self reconfiguration algorithm that decides if a given modular robot structure can be embedded into another structure to form new morphology (Fig. 2.). The process of SET starts by taking the two robotic structures and convert them into two connected graph. Then, it applies MST to them and randomly selects the initial roots for the trees.

We consider that the robotic system is modeled as a connected graph G (M, L): M denotes the set of nodes representing the modules, and L $\subseteq$ M $\times$ M denotes the set of links that connecting such modules.

SET maintain a $|M_1| \times |M_2|$ truth table T, where T[m_1, m_2] is true, under a specified rooting $r_1 \in M_1$, $r_2 \in M_2$. At the end of the algorithm, T[r_1, r_2] answers whether structure (S_1) embeds in stucture (S_2) under r_1 and r_2; if the answer is negative, it repeat the process for a new rooting until it either get a positive answer or we exhaust all possible rootings, in which case we conclude that S_1 does not embeds in S_2.

SET is a two pass algorithm. At first, all entries of the truth table are false. After that, it start proceed bottom-up, starting from the leaves of S_1, and keep going gradually towards r_1. As a starting point, it consider a leaf $m_1 \in M_1$ and check whether it embeds in the leaves of S_2, to calculate the truth table that give us the new morphol-

ogy according to the topological condition (see [21] for details). Basically, it fill the truth table by traversing G_1 in reverse pre-order, where at each step of the traversal process the nodes of G_2 in reverse pre-order.

After first pass, the second pass is lunched, where it involves a top-down message passing. It iterate top-down, starting from the roots of S_1, and progressively down to v_1. Then it compute a new table called T^*, based on the topological condition in top-down and the preceding truth table in bottom-up pass. However, $T^*[r_1, n] = \text{true}$ iff $T_n[r_1, n] = \text{true}$, $\forall n \in M_2$. It is then not hard to see that S_1 embeds in S_2 iff at least one entry of the $r_1 - th$ row of T^* is true. if the answer is negative, This process is repeated until we either get a positive answer or we exhaust all possible rootings.

Indeed, it is proved that the loop process generated in the algorithm will always end. Subsequently, SETis highly dependent on the randomly initial tree roots. SET algorithm is one of the simple method in the self reconfiguration approach that has been used in wide range of domains.

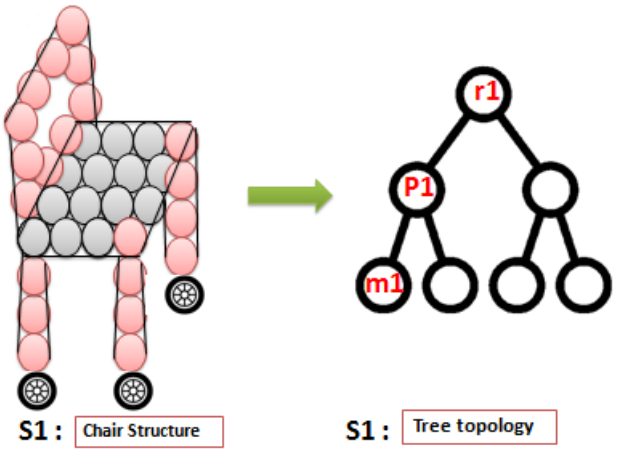

Fig. 3 Convert MRS to Connected Graph.

3.2 FSET: Fast SET Algorithm

3.2.1 Root Selection Stage

Mostly, the efficiency and performance of the SET algorithm are greatly affected by initial tree roots as different initial tree roots often lead to different embedding or failure in the embedding and thus, failure to obtain the new morphology. Therefore, the calculation time cost for the embedding truth table between the modules of the

two robotic structures to form new morphology will be high. Hence, the selection of the initial root trees is becoming a challenge for the SET algorithm.

The first stage of our adapted SET is called root selection and aims to solve the above problem. We propose to select a subset/training from the two sets of modules G_1 and G_2 of the connected graph that represents the two modular robotic structures, in order to find the approximate final tree roots $R = r_i, r_j$ that generate the final morphology. Our intuition is to reduce the number of iterations needed in the traditional SET to obtain the final morphology as fast as possible, thus enhancing the processing time of the SET.

Obviously, the efficiency of the selection root stage is highly related to the percentage, represented by Ts (i.e. training size), of a training set of modules. Subsequently, increasing the value of Ts leads to an increase in the calculation time of FSET so no profit will be noticed compared to traditional SET. On the other hand, the lowest the value of Ts is the better the processing time, but the error in the final obtained morphology will increase. Therefore, selecting the appropriate value of Ts is very essential in the first stage of our technique. Indeed, we believe that Ts should be determined by the decision-makers or experts depending on the application requirements.

3.2.2 Morphology Formation Stage

After having the approximate initial root of trees R, the second stage is lunched which aims to reduce the number of iteration loops in SET. So, it take the obtained roots in the first stage and the whole sets of modules G , and then it apply SET over G in order to get the final morphology.

Algorithm 1 describes the procedure of the second stage of our technique. First, we determine the number of modules needed to find the tree roots in the first stage of our technique (line 2). Based on this number, we randomly select the training sets among the whole sets of modules G1 and G2 (lines 3-6). The modules in the training set represent now the approximate roots of the trees. Then, we calculate the T* embedding truth table. This process is repeated until we either get a positive answer or we exhaust all possible rootings (lines 14-20). At this moment, the first stage is accomplished and the initial roots are determined (line 21). After that, the second stage is running where the process starts by considering the roots obtained in the first stage as the initial roots of the trees. Then, we calculate the T* embedding truth table based on the obtained roots from the first stage. Again, the loop is repeated until we either get a positive answer or we exhaust all possible rootings that give us the embedding sequence that form the new morphology. (lines 21-30).

Algorithm 1 FSET Algorithm.

Require: Two sets of modules: $G_1 = \{m_1, m_2, \ldots, m_k\}$; $G_2 = \{n_1, n_2, \ldots, n_q\}$; Percentage of training set: T_s.

Ensure: Embedding truth table T* that generate the new morphology.

1: $G_s \leftarrow \emptyset$

$N_s \leftarrow [(T_s \times k)/100]$
for $i \leftarrow 1$ to N_s **do**
  // randomly selects the training set of modules among G_1 and G_2
  $G_s \leftarrow G_s U G_i$
end for
for $i \leftarrow 1$ to 2 **do**
  $T_i \leftarrow \emptyset$
  // randomly choose roots r_i among G_s belongs T_i
end for
// Initially, all entries are false
(i.e $T[m_i, n_j]$=false)
trace the two tree in a bottom-up to calculate the embedding truth table T between modules
repeat
  for $i \leftarrow 1$ to s **do**
    for $j \leftarrow 1$ to s **do**
      // involves a top-down message passing to calculate embedding truth table $T^*[m_i, n_j]$ between modules
    end for
  end for
until $r_t h$ row of T^* contain at least one true value or exhaust all possible rootings
extract the roots r_i and r_j from the previous T*
//use the whole set of modules G_1 and G_2
repeat
  for $i \leftarrow 1$ to k **do**
    for $j \leftarrow 1$ to q **do**
      // involves a top-down message passing to calculate embedding truth table $T^*[r_i, r_j]$ between modules
    end for
  end for
until $r_t h$ row of T^* contain at least one true value or exhaust all possible rootings
return T^* that generate the new morphology

4 Simulation and Results

In order to evaluate its efficiency, we tested our mechanism on one of the most used MRS proposed in recent years, e.g. M-TRAN [22]. Indeed, M-TRAN is a three-dimensional modular robotic system, with characteristics of both lattice and chain (linear) types of modular robot. Each M-TRAN module is made up of two semi-cylindrical pieces that can rotate 180 degrees around their axis and have an

independent battery, two degrees of freedom motion, six surface connections, and intelligence with inter-module communication. The M-TRAN system can perform flexible and adaptive locomotion in various configurations using coordination control based on a CPG [23]. Figure 4 presents the components of a M-TRAN module.

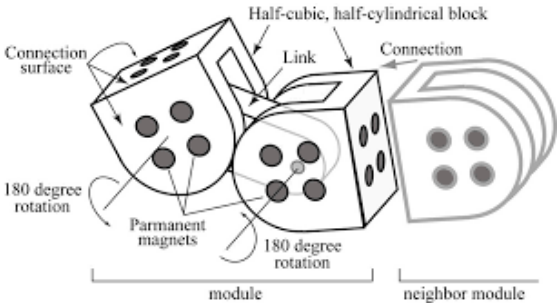

Fig. 4 M-TRAN components.

In our simulations, we used two robotic that have chair and wall design and are made out of M-TRAN modules. We obtain a new design with stroller morphology quickly after using the FSET method (Fig. 5).

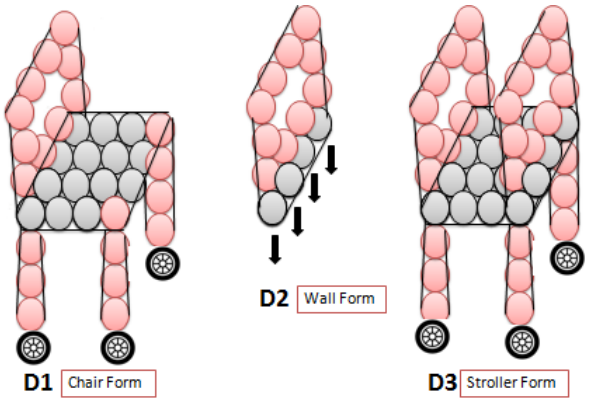

Fig. 5 Morphologies adapted in our simulation.

The objective of our simulations was to confirm that our technique can successfully achieve intended results for reducing the delay to obtain new morphology and reducing the energy consumption in modules that leads to extend the MRS lifetime. In order to evaluate the performance, we compare our results to the traditional SET. In our simulations, we evaluated the performance using the following parameters:

- The Number of Modules for substructure n_b, takes the following values: 100, 200, 300, 1000 and 2000.

- the Number of Modules for superstructure n_p, takes the following values: 500, 950, 2000 and 4000.
- the percentage of module chosen, T_s, takes the following values: 5, 10, 15 and 20.

4.1 Execution Time

Sometimes, getting new morphology fast time as possible to the end-user is a crucial operation especially in e-health and military applications. Fig. 6, shows the execution time for both FSET and SET when varying the number of modules (for both the substructure and the superstructure respectively). The results show that FSET can optimize the execution time, comparing always to the SET, from 10% (while varying number of module from (100,500) to (300,2k) module) to 37% (while varying number of module from (500, 2k) to (2k,4k) module).

Obviously, the execution time of FSET will be highly affected by the selection of the tree roots as well as the number of iteration loops to obtain the final morphology. Therefore, FSET outperforms the SET where the processing time to get new morphology is twice accelerated when using FSET, compared to SET algorithm

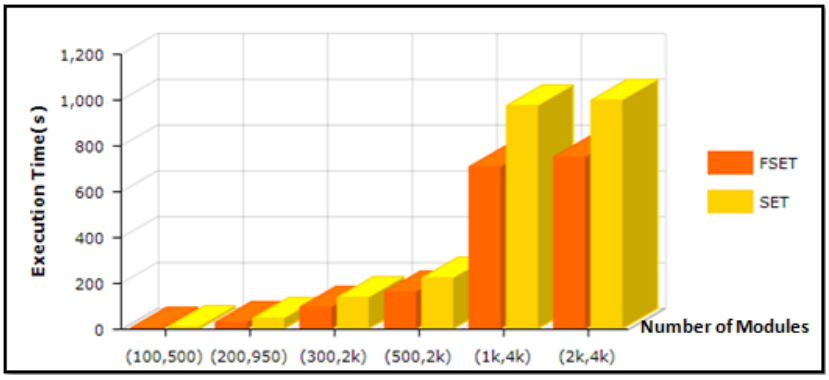

Fig. 6 Processing time for FSET and SET..

4.2 Iteration Loop

One of the factor that can delay the obtain of new morphology is the number of iterations. In Fig. 7, we show how many iterations are generated by the the two robotic structures to find the final morphology for both FSET and the SET. It is important to know that a high number of iterations can increase the complexity of

the proposed algorithm. The obtained results show that, The number of iterations is reduced by at least 30% as shown in these figure when applying FSET on the SuperBot modules. Therefore, FSET minimize the morphology delay by reducing the number of iterations.

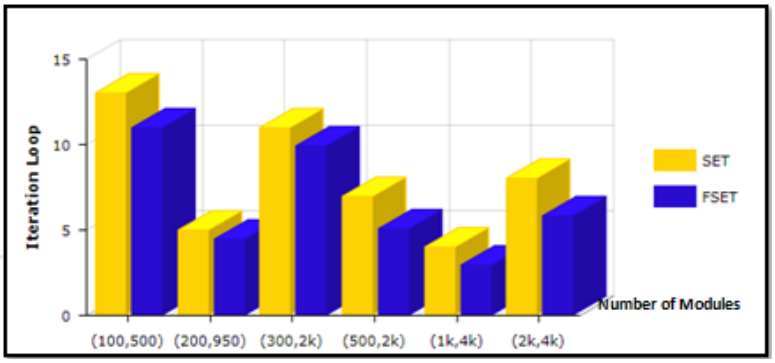

Fig. 7 Iteration loop number for FSET and SET.

4.3 Energy Consumption

Energy consumption is a crucial parameter to assess in MRS since it impacts the overall system's functionality. Indeed, the data transmission or activity done during the transition consumes the majority of a module's limited energy. In our simulation, we implemented the same energy model that used in [16] to calculate the energy consumption in SuperBot modules, assuming that each module has fixed energy units, based on the MRS size, then we considered that each message transmission consumes 0:2 unit and each successful embedding modules consumes 0:8 unit.

Fig. 8, shows the energy consumed in the SuperBot robot depending on modules number. The obtained results show that the energy consumption increases with the increasing of the modules number while it is optimized, using FSET, up to 68% compared to the SET approach. Therefore, our proposed technique can be considered very efficiently in terms of reducing the energy consumption of the modular robotic system, thus, increasing its lifetime.

5 Conclusion and Future Work

In this paper, we have proposed a fast self reconfiguration technique called FSET, i.e. Fast SET, dedicated to MRSs. Our proposed technique consists mainly in two

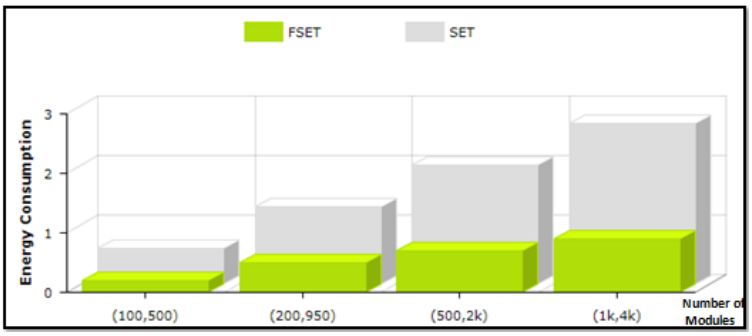

Fig. 8 Energy consumption for FSET and SET.

stages: root selection and morphology formation. The final goal of these stages is to enhance the time cost of creating new morphology of traditional SET algorithm thus, ensure fast self reconfiguration. The root selection stage selects a small number of modules in order to find the best tree roots that effects the topological conditions that leads to successful of the embedding process or not. The morphology formation stage uses the traditional SET algorithm to calculate the embedding truth table where the initial roots used are taken from the first stage. Finally, we demonstrated FSET's efficiency in terms of complexity, optimality (lowest number of steps), and time efficiency by simulating its performance on a real robot called SuperBot.

As a future work, we will study how to handle designs with a small number of cycles and how to reduce the kinematic checking runtime. We will go from detecting embeddability to design synthesis in the long run. We consider our embedding method is a good place to start for this line of research, and preliminary results are promising.

References

1. P. Thalamy, B. Piranda, and J. Bourgeois, "Distributed self-reconfiguration using a deterministic autonomous scaffolding structure," Ph.D. dissertation, UBFC, 2019.
2. R. Thakker, A. Kamat, S. Bharambe, S. Chiddarwar, and K. Bhurchandi, "Rebis-reconfigurable bipedal snake robot. 2014 ieee," in *RSJ International Conference on Intelligent Robots and Systems, Chicago-USA*, pp. 309–314.
3. P. Thalamy, B. Piranda, and J. Bourgeois, "A survey of autonomous self-reconfiguration methods for robot-based programmable matter," *Robotics and Autonomous Systems*, vol. 120, p. 103242, 2019.
4. S. Vassilvitskii, M. Yim, and J. Suh, "A complete, local and parallel reconfiguration algorithm for cube style modular robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 1. IEEE, 2002, pp. 117–122.
5. S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji, "Hardware design of modular robotic system," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, vol. 3. IEEE, 2000, pp. 2210–2217.

6. S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME transactions on mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.
7. A. Sproewitz, P. Laprade, S. Bonardi, M. Mayer, R. Moeckel, P.-A. Mudry, and A. J. Ijspeert, "Roombots—towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1126–1132.
8. E. Yoshida, S. Murata, H. Kurokawa, K. Tomita, and S. Kokaji, "A distributed method for reconfiguration of a three-dimensional homogeneous structure," *Advanced Robotics*, vol. 13, no. 4, pp. 363–379, 1998.
9. K. D. Kotay and D. L. Rus, "Algorithms for self-reconfiguring molecule motion planning," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, vol. 3. IEEE, 2000, pp. 2184–2193.
10. C. Unsal and P. K. Khosla, "A multi-layered planner for self-reconfiguration of a uniform group of i-cube modules," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, vol. 1. IEEE, 2001, pp. 598–605.
11. C. Ünsal, H. Kiliççöte, and P. K. Khosla, "A modular self-reconfigurable bipartite robotic system: Implementation and motion planning," *Autonomous Robots*, vol. 10, no. 1, pp. 23–40, 2001.
12. D. J. Dewey, M. P. Ashley-Rollman, M. De Rosa, S. C. Goldstein, T. C. Mowry, S. S. Srinivasa, P. Pillai, and J. Campbell, "Generalizing metamodules to simplify planning in modular robotic systems," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 1338–1345.
13. M. Yim, Y. Zhang, J. Lamping, and E. Mao, "Distributed control for 3d metamorphosis," *Autonomous Robots*, vol. 10, no. 1, pp. 41–56, 2001.
14. R. Fitch, Z. Butler, and D. Rus, "Reconfiguration planning for heterogeneous self-reconfiguring robots," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 3. IEEE, 2003, pp. 2460–2467.
15. ——, "In-place distributed heterogeneous reconfiguration planning," in *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pp. 159–168.
16. P. Thalamy, B. Piranda, and J. Bourgeois, "A survey of autonomous self-reconfiguration methods for robot-based programmable matter," *Robotics and Autonomous Systems*, vol. 120, p. 103242, 2019.
17. R. Fitch and R. McAllister, "Hierarchical planning for self-reconfiguring robots using module kinematics," in *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 477–490.
18. Z. Butler and D. Rus, "Distributed planning and control for modular robots with unit-compressible modules," *The International Journal of Robotics Research*, vol. 22, no. 9, pp. 699–715, 2003.
19. M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 403–421, 2008.
20. B. McKay, "Nauty user's guide (v2. 4)," *Computer Science Dept., Australian Nat. Univ*, 2007.
21. Y. Mantzouratos, T. Tosun, S. Khanna, and M. Yim, "On embeddability of modular robot designs," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1911–1918.
22. S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME transactions on mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.
23. H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata, "Self-reconfigurable modular robot m-tran: distributed control and communication," in *Proceedings of the 1st international conference on Robot communication and coordination*, 2007, pp. 1–7.