Spanning Tree Partitioning Approach for Configuration Generation in Modular Robots

Ayan Dutta[1], **Prithviraj Dasgupta**[1], **José Baca**[1] **and Carl Nelson**[2]
[1] Computer Science Department, University of Nebraska at Omaha
[2] Mechanical and Materials Engineering Department, University of Nebraska-Lincoln.
Email: {adutta, pdasgupta, jbacagarcia}@unomaha.edu, cnelson5@unl.edu

Abstract

We consider the problem of configuration generation in modular self-reconfigurable robots, where a set of modules that are in a certain configuration are required to form a new configuration while remaining within the size, battery and communication constraints. This problem is computationally non-trivial as the set of possible configurations grows exponentially with the number of modules. We propose a novel, anytime and distributed algorithm that uses a branch-and-bound pruning technique to reduce its search space, by constructing a minimum spanning tree and finally determines the highest utility configuration among the set of modules. Experimental results show that our technique can quickly identify modules to form new configurations for different configuration sizes.[1]

Introduction

Modular self-reconfigurable robots (MSRs) (Yim et al. 2007) are autonomous robots composed of individual modules, which can change their connections with each other to form different shapes. MSRs are particularly attractive for maneuvering tasks in initially unknown and unstructured terrain due to their ability to adapt their shapes to the current environment features, so that they can perform their assigned task efficiently. A central problem in the reconfiguration of MSRs is the configuration generation problem - how to identify a suitable set of modules to form a new configuration when it cannot continue to perform its operations in its current configuration. The MSR configuration generation problem is non-trivial as the set of possible configurations increases exponentially as the number of modules increases. Also, generally the formulation of the problem relies on centralized algorithms and modules are required to report their operational parameters to a central location to perform the computations (Dutta et al. 2014). Because of these issues, it becomes difficult to realize the centralized techniques within the time and space constraints available for generation of configurations.

[1]This research has been supported by NASA grant no. NE-NNX11AM14A as part of the ModRED project.

To address these challenges, we propose a novel technique that attempts to reduce the search space by first constructing a minimal reachability graph between modules that are within communication range of each other, in the form of a minimum spanning tree (MST). Then, the possible combinations of only those modules that are connected in the MST, upto a specific size, n_{max}, which is determined from the maneuverability constraints of modules, are explored using an integer partitioning based technique to find the combination or configuration of modules that gives the highest expected utility. The computations are performed in a distributed manner by the modules.We have performed simulated experiments and shown that our proposed technique is able to rapidly identify the highest utility configuration for different number of modules and performs significantly better in terms of time and space complexity than previously existing techniques for MSR configuration generation and coalition structure search algorithms.

Related Work

MSRs are a type of self-reconfigurable robots that are composed of several small modules (Yim et al. 2007). Self-reconfiguration of MSR involves the problem of finding the best configuration for current environment. This problem has been solved using graph search-based techniques (Hou and Shen 2008) and control-based techniques (Chirikjian, Pamecha, and Ebert-Uphoff 1996). In (Kurokawa et al. 2008), the authors proposed a task-based reconfiguration technique where the goal configuration is determined dynamically as the configuration that can perform the robot's current task efficiently. Our work in this paper is complementary to these techniques as it enables modules from different disjoint configurations to identify the best set of modules to form goal configurations that give the highest suitability, denoted in terms of the configurations' utility for performing the MSR's current task such as navigation. Any of the above reconfiguration techniques could be used to effect the mobility and position selection by modules to reach the identified goal configurations.

Coalition game theory gives a set of techniques that can be used by a group of agents to form teams or coalitions with each other. Similar to our proposed approach of limiting team size up to a size of n_{max}, Shehory and Kraus (Shehory and Kraus 1998) proposed a search-based heuristic to

find coalitions upto a pre-determined maximum coalition size. In terms of MSRs, a coalition represents a set of MSR-modules that are connected together. Within coalition game theory, the coalition structure generation problem deals with partitioning the agents into a partition that gives the highest value. This problem is NP-complete, and researchers have proposed a coalition structure graph based algorithm (Sandholm et al. 1999) to find the optimal coalition structure. In (Michalak et al. 2010), authors have proposed an algorithm, which distributes the search space among agents, but also requires significant communication among them. For physical robots, because wireless communication consumes energy and might be unreliable, our proposed algorithm attempts to minimize the communication overhead. However, in all these approaches the possible module configurations are generated and inspected in a centralized manner, which increases the computation by the modules. In contrast, this paper proposes a distributed solution to this problem.

Dynamic Configuration Generation in MSRs

In this paper, we focus primarily on this problem of determining a new configuration, while assuming that the basic operations for maneuvering and moving in a fixed configuration (Baca et al. 2014) are already available. Let A be the set of modules or agents that have been deployed in the environment. Let $\Pi(A)$ be the set of all partitions of A and let $CS(A) = \{A_1, A_2, ..., A_k\} \in \Pi(A)$ denote a specific partition of A. We call $CS(A)$ a configuration or a coalition structure, and A_i as a coalition.[2] Though we have used this proposed configuration generation technique for chain-type MSRs, this technique can be used for other types of MSRs such as lattice and hybrid also.We denote B^{max} as the maximum battery power on a module and $\mu_B^{A_i}$ and $\sigma_B^{A_i}$ as the mean and standard deviation of the battery power available on MSR A_i. Let $Val : A \rightarrow \mathbb{R}$ denote the value function given below:

$$Val(A_i) = \begin{cases} \mid A_i \mid^2 (\frac{\mu_B^{A_i} - \sigma_B^{A_i}}{B^{max}}), & \text{if } \mid A_i \mid \leq n_{max} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The value function assigns to each coalition A_i a real number corresponding to a virtual worth of the coalition. We have kept the notion of a task abstract; it could correspond to lifting non-operational modules or crossing a raised surface. As a single module does not have much physical as well as computational power, thus most of the time it is better to form larger coalitions. Again, if the size of the coalition is huge, then it becomes intractable to manage them, because of maintenance cost such as communication overhead and mobility inefficiency of modules etc. (Shehory and Kraus 1998). Our value function ensures that larger coalitions are able to perform their task better and obtain higher rewards, up to a size of n_{max}, subject to their available battery power.

The term $(\frac{\mu_B^{A_i} - \sigma_B^{A_i}}{B^{max}})$ considers the available battery in A_i; it ensures that for two MSRs of same size, higher preference is given to the MSR that has a higher mean and lower variance in battery power. Lower variance will ensure similar battery power having modules in a particular coalition. n_{max} denotes the maximum allowable size of a coalition; it is given as input to our algorithm and it does not change the operation of the algorithm. n_{max} can be determined from the physical characteristics of the modules (e.g., battery, maximum weight of other modules that can be lifted by the modules), the features in the environment such as the amount of free space, the task that the robot has to perform etc. As the focus of this paper is on determining the best module configurations, the problem of determining n_{max} is not considered further in this paper. The value of configuration $CS(A)$ is given by the summation of the values of coalitions comprising it, i.e., $Val(CS) = \sum_{A_i \in CS(A)} Val(A_i)$.

The configuration formation cost incurred by modules includes the energy expended by them to move to each others' vicinity, align their docking faces appropriately and finally connect with each other using a docking mechanism. We denote the cost of forming a coalition between A_i and A_j, following (Dutta et al. 2014), as: $cost(A_i, A_j) = cost_{loc} + cost_{dock}$. The cost of a coalition structure $CS(A)$ can be written as: $Cost(CS) = \sum_{A_i, A_j \in CS(A)} cost(A_i, A_j)$. The utility of a coalition structure $CS(A)$ is then given by $U(CS) = Val(CS) - Cost(CS)$. The objective function of the MSR configuration formation problem is to find CS^*, where $CS^* = \arg \max_{CS \in \Pi(A)} U(CS)$. [3]

Spanning Tree Based Representation of Coalitions

Algorithm 1: Stochastic MST Formation

```
formMST()
Each module a_i will do the following:
for a_k in comm. range do
    w_{i,k} ← est. range of a_k from a_i
    w^min_{i,k}, w^max_{i,k} ← (w_{i,k} − δ, w_{i,k} + δ)
    send w_{i,k} = (w^min_{i,k}, w^max_{i,k}) to all modules in comm. range
wait till recd. w_{i,k} from all modules in comm. range
for each e_{j,l} ∈ E do
    w^min_{j,l} ← min(w^min_{j,l}, w^min_{l,j})
    w^max_{j,l} ← max(w^max_{j,l}, w^max_{l,j})
    μ_{j,l}, σ_{j,l} ← mean(w^min_{j,l}, w^max_{j,l}), stdev(w^min_{j,l}, w^max_{j,l})
    w*_{j,l} ← μ_{j,l} − σ_{j,l}
G* ← Prim(V, E, W*)
```

The first step in our configuration generation process is to build a communication graph representing the connectivity between the modules. We assume a scenario where modules can form a connected graph using their communication range, i.e., any module's emitted message can be delivered to every module in the environment. To build the communication graph each module broadcasts a message

[2] A coalition is a set of modules connected together and a coalition structure or a configuration is set of coalitions.

[3] Modules exchange their positions and amount of battery available as a pre-processing step.

with its unique identifier. Each module receiving this broadcast message sends back an acknowledgement message with its identifier to the modules from whom it has received broadcast messages. The communication graph is denoted as $G = (V, E, W)$, where V is the set of nodes (modules), E is set of edges between modules and W is the set of edge weights that denotes the cost of the connecting or docking the modules at the extremities of the edge. Initially, modules are uncertain about the edge weights. So, they use a decentralized protocol to arrive at a consensus about the expected weight of each edge as shown in Algorithm 1.

Each module a_i estimates weight for edge $e_{i,k}$ between itself and a_k as weight $w_{i,k} \pm \delta$, where δ represents uncertainty that might arise in the edge weight (docking cost) due to modules' motion, localization errors and initially unknown environment features (lines $4-5$). As every module can have a different realization of $\mathbf{w}_{i,k}$, each module broadcasts its calculated weights $\mathbf{w}_{i,k}$ (line 6). When a_i receives edge weights for edge $e_{j,l}, \forall j, l \in V$ from all the modules, it calculates the minimum and maximum values from the weight estimates $\mathbf{w}_{j,l}$ and $\mathbf{w}_{l,j}$, sent by a_j and a_l respectively and sets the weight of $e_{j,l}$, $w^*_{j,l}$ as the difference between the mean and standard deviation of the minimum and maximum weight values (lines $9-11$). The main idea behind calculating the final edge weights in such as way is to give preference to edges which have lower standard deviation, i.e., lower uncertainty of connecting the end modules. Each module then generates the MST $G^* = (V^*, E^*, W^*)$ from the graph $G = (V, E, W^*)$ using, for instance Prim's algorithm (Prim 1957); G^* gives a connected set of all the modules, where $\sum_k w^*_k$ is minimum.

Distributed Tree Partitioning (DTP) Algorithm

After the MST is generated, the next step in our proposed MSR configuration-generation technique is to distribute the search for the best configuration among the modules. As shown in Algorithm 2, we first partition the number of modules into its integer partitions in a restricted manner. As our value function gives preference only to the coalitions with sizes up to n_{max}, so we prefer searching through the integer partitions which has no member greater than n_{max}. Let $IP(n)$ denote a function that returns an ordered set of integer partitions of a positive integer n and let $IP^*(n, n_{max}) \subset IP(n)$, where $n_{max} < n$ denote the integer partitions of n which have members not greater than n_{max}. For example, with $\mid A \mid = 4$ and $n_{max} = 2, IP(\mid A \mid) = IP(4) = \{(4), (3,1), (2,2), (2,1,1), (1,1,1,1)\}$ while $IP^*(4,2) = \{(2,2), (2,1,1), (1,1,1,1)\}$.

Distribution of partitions: An integer partition and a coalition structure are complementary representations of each other - the integer partition denotes the number of modules in the different coalitions forming a coalition structure, while the coalition structure denotes the identity of the modules that comprise the coalitions in a partition. For example, for integer partition (4) the corresponding coalition structure is $\{1,2,3,4\}$. Similarly, the corresponding coalition structures of integer partition $(3,1)$ are given by all coalition structures which have 2 coalitions in them with sizes of 3 and 1 respectively, such as $\{\{1,2,3\},\{4\}\}$, $\{\{1,2,4\},\{3\}\}$ and so on. First, we rank the integer partitions from 1 to $|IP^*(|A|, n_{max})|$ based on the upper bound of the value of the coalition structures corresponding to the integer partitions. Recall from Equation 1 that the value of a coalition is a function of its size. Therefore, the coalition structure's value can be calculated from the coalition sizes given by its corresponding integer partition. Simultaneously, the modules are ranked based on decreasing amount of energy (e.g. battery power) available on them. This ranking is done by every module before distribution of partitions at every round, from the battery information available from other modules at the beginning. The ranked integer partitions are then allocated to the ranked modules; the integer partition with rank 1 is allocated to the module ranked 1 and so on.

The main reason behind this rank-based allocation is to use more 'eligible' modules that have higher available battery to look for higher valued coalition structures, so that they have a higher chance to finish the computations for the search process with their available energy, as higher ranked partitions are more likely to contain the best coalition structure. Each module searches through the coalition structures corresponding to its assigned integer partition. The search procedure is described in the next subsection. At the end of each search phase, the modules exchange the best coalition structure found, CS^*, and the highest utility found, U^*, among themselves. Then they prune the necessary partitions from the remaining partitions, $(\mid IP^*(|A|, n_{max}) \mid - \mid A \mid)$, using the *checkFitness* rule.

checkFitness Rule: The insight behind this rule is that if the maximum possible value of a coalition structure, that can be calculated from its corresponding integer partition ($Val(CS) \leq \sum_j (ip_{i,j})^2$), is already less than the highest utility (U^*) seen thus far, then there is no need to further search the coalition structures that can be generated from that integer partition. For example, if $ip_{i,j}$ is $(2,2)$, then highest value of any coalition structure corresponding to $(2,2)$ will be $max(Val(2) + Val(2)) = 2^2 + 2^2 = 8$ and if U^* is, say 9, then we can say that no coalition structure in

Algorithm 2: Distributed Partition Search

Input: MST G^* and n_{max}
Each module a_i will do the following:
while *All partitions in* $IP^*(|A|, n_{max})$ *has not been searched* **do**
  for *each phase of search* **do**
    $IP_{rank} \leftarrow$Ranked set of partitions ip_k, $\forall ip_k \in IP^*(|A|, n_{max})$ and ip_k is not searched before.
    $A_{rank} \leftarrow$Ranked set of modules $a_j, \forall a_j \in A$.
    $ip_{curr} \leftarrow$ Allocated partition to module a_i. (Described in text)
    $searchCS(ip_{curr}, ip_{curr,1}) \rightarrow$Generate and search coalition structures. (Algorithm 3)
    if *all coalition structures in* ip_{curr} *have been searched* **then**
      Communicate with $a_k, \forall k \neq i$ and a_k in communication range of a_i, to update U^*, CS^*.

partition $(2, 2)$ will have higher utility than U^*. If ip_i is the partition to be searched next, then the $checkFitness$ rule for ip_i is given by:

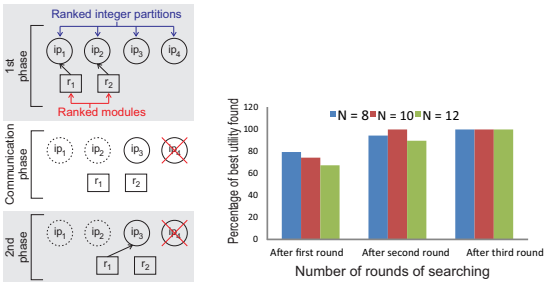
$$checkFitness(ip_i) = \begin{cases} Prune, & \text{if } \sum_{j \in ip_i} (ip_{i,j})^2 \leq U^* \\ Search, & \text{otherwise} \end{cases} \quad (2)$$

where $ip_{i,j}$ is the j-th member of ip_i.

While allocating integer partitions to modules, if the number of modules is greater than the number of partitions to be searched, then some of the lower ranked modules are not allocated any partition to search. An example is shown in Figure 1.(a) where two modules r_1 and r_2 need to search 4 integer partitions; modules and integer partitions are ranked as shown. First, ip_1 and ip_2 are searched by r_1 and r_2 respectively; modules r_1 and r_2 exchange the best utilities they have found for a coalition structure following their searches and update U^* (communication phase). After that, using $checkFitness$ rule, unfit integer partition ip_4 will be pruned by both modules and the remaining partition ip_3 will be searched by r_1 (2nd phase).

Search for the best coalition structure: After the partitions are distributed among the modules, each module searches for the best coalition structure within G^*, corresponding to the partition which is allocated to it. Each connected component in G^* corresponds to a coalition. Thus the spanning tree itself is a *grand* coalition, where all the modules are connected together. To find the best coalition structure, modules need to break or partition this grand coalition and search through the promising coalition structures. For example, with $|A| = 4$, if G^* is $1 - 2 - 3 - 4$ ($a - b$ denotes an edge between nodes a and b.), then the only coalition structure corresponding to integer partition $(2, 2)$ is $\{\{1, 2\}, \{3, 4\}\}$. But $\{\{1, 3\}, \{2, 4\}\}$ and $\{\{1, 4\}, \{2, 3\}\}$ will not be a promising coalition structure to search because there is no edge between 1 and 3 or 1 and 4 in G^*; because weights of these edges are already high, i.e., costs of forming these coalitions will be higher than others. So, the modules will only search those coalitions which have a corresponding edge in G^*. Thus the search is already directed towards the coalitions which have lower costs. A large amount of unpromising coalitions can be pruned by this search technique using the minimum spanning tree, and, thus, the time complexity of the search can be reduced drastically.

How the search is done within the minimum spanning tree corresponding to each partition, is shown in Algorithm 3. First, we calculate how many edge deletions are needed to transform the spanning tree (i.e., the grand coalition) to a coalition structure corresponding to the current integer partition. For example, if the integer partition is $(1, 3)$, then we need to partition the MST $1 - 2 - 3 - 4$, into 2 coalitions, one with size 1 and the other with size 3. All possible coalitions corresponding to $ip_{curr,i} \in ip_{curr}$ are generated only from possible connected components in G^*. If deletion of a certain edge does not generate a coalition corresponding to $ip_{curr,i}$, then we discard information about that coalition from the current search (line 3). Once one coalition is generated, corresponding connected component is removed from G^*, to avoid overlap in coalitions (line 7). Coalition structures are formed by generating member coalitions and adding them to the coalition structure. Once we add a coalition from $ip_{curr,i}$ to the current coalition structure, the next coalition is generated from $ip_{curr,i+1}$; and this continues until we find all the member coalitions for a coalition structure (lines $9 - 10$) and then finally we update the best utility value (lines 13). While generating the member coalitions of a certain coalition structure, we continuously check whether the current coalition structure being searched, can be the best coalition structure or not. This is done using *checkCS* rule.



(a) (b)

Figure 1: (a) Partition distribution processes among modules, (b) Empirical data on anytimeness property of DTP algorithm

checkCS Rule: As a coalition structure consists of coalitions and we search through the coalitions, so before adding a coalition to one coalition structure, we check whether that coalition can help the current coalition structure to be the best (in terms of utility) or not. Let CS_α be the current coalition structure that is being formed and $C = \{C_1, C_2, .., C_{|C|}\}, C_i \subset CS_\alpha$ be the set of non-overlapping coalitions. $checkCS$ Rule says that if $\sum_j U(C_j) + \max \sum_i U(C_i) \leq U^*, \forall j \in \{1, curr - 1\}, \forall i \in \{curr, |C|\}$, then CS_α need not to be searched anymore. C_j is the set of coalitions that is already been generated and added to the current coalition structure CS_α. C_{curr} is the current coalition that is being generated. If the maximum utility of the future coalitions, i.e. $\max \sum_i U(C_i) = \sum_i |C_i|^2$, added with the sum of the utilities of previously added coalitions can not provide higher utility than the current best utility U^*, then there is no need to form CS_α anymore. Similar pruning technique has been used in (Rahwan et al. 2009).

In our proposed model, if an integer partition can produce a coalition structure which has more number of coalitions of size n_{max}, then that partition is ranked higher and searched first. Therefore, the first coalition structure generated will have the highest number, $\lfloor \frac{|A|}{n_{max}} \rfloor$, of coalitions of size n_{max} in any coalition structure with $\mid A \mid$ modules. This guarantees that after the first generated coalition structure, the utility of any coalition structure that is admitted by the algorithm will be within a certain bound from the optimal coalition structure, (*if not the optimal itself*). Therefore, we can establish a worst case bound from the very first coalition structure inspected by the algorithm, which makes *DTP*

Algorithm 3: Search Algorithm for Finding Best Coalition Structure

$searchCS(ip_i, ip_{i,j})$
For $ip_{i,curr} \in ip_i$ do the following:
$\mathbf{v_{ip_{i,j}}} \leftarrow$ generate all combinations of $ip_{i,curr}$ nodes that are connected in G^*
for *each* $v_{ip_{i,j}} \in \mathbf{v_{ip_{i,j}}}$ **do**
  // $v_{ip_{i,j}}$ is a set of $ip_{i,curr}$ nodes, connected in G^*
  Apply checkCS rule.
  $G^* \leftarrow G^* \setminus v_{ip_{i,j}}$
  if $G^* \neq \{\emptyset\}$ **then**
    $curr$++;
    searchCS($ip_i, ip_{i,curr}$);
  else
    // found connected components of all $ip_{i,curr}$ nodes
    Update U^* and CS^*.
return

algorithm anytime.

Lemma 1 checkFitness Rule *does not remove an optimal coalition structure.*

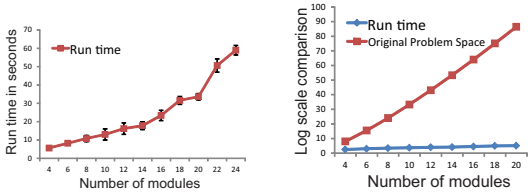
Proof: Suppose that $checkFitness(ip_i)$ returns *Prune* and removes the optimal coalition structure $CS^*(A)$ and let $\mathbb{Z} = \sum_{j=1}^{|ip_i|}(ip_{i,j})^2$. From the definition of $checkFitness(ip_i)$ in Eqn. 2, it follows that $\mathbb{Z} \leq U^*$. Again, from the definition of optimal coalition structure, $U(CS^*(A)) > U^*$. But from the value function, we can see that $\mathbb{Z}$ is the upper bound of the utility that any coalition structure in ip_i can have, thus $U(CS^*(A)) \leq \mathbb{Z} \leq U^*$. So, $U(CS^*(A)) < U^*$ and thus $CS^*(A)$ is not the optimal coalition structure. Hence proved.

Lemma 2 checkCS Rule *does not remove an optimal coalition structure.*

Proof: Suppose that $checkCS(CS)$ prunes the optimal coalition structure CS^*, and thus $U(CS^*) > U^*$, where U^* is the current best utility. From the definition of $checkCS$ rule, if $\sum U(C_j) + \sum |C_i|^2 \leq U^*$, where $C_j, C_i \in CS_{curr}, \forall j \in \{1, curr-1\}, \forall i \in \{curr, |C|\}$. Now if CS^* has been pruned by this rule, then $U(CS^*) \leq U^*$. So, we can say, $U(CS^*) = \sum U(C_j) + \sum |C_i|^2 \leq U^*$. Now as, $\sum |C_i|^2$ is the upper bound on the value of future coalitions which need to be added to the current coalition structure and $\sum U(C_j)$ is the total utility for previously generated coalitions, so the upper bound of the current coalition structure is $U(CS^*)$, where $U(CS^*) \leq U^*$. Thus the optimal coalition structure can not be pruned by the $checkCS$ rule. Hence proved.

Experimental Evaluation

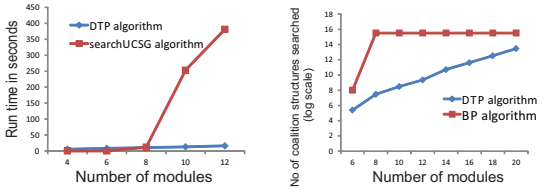
We have implemented the DTP algorithm in Webots simulator. We consider different scenarios where the simulated MSR consists of $N = |A| = 4$ through 24 modules and it needs to generate the best configuration. The initial positions of modules are drawn from $\mathcal{U}[(0,0),(10,10)]$; the



(a) (b)

Figure 2: (a) Run time of DTP algorithm in seconds, (b) Comparison of growth of original problem space and run time of DTP algo. with number of modules

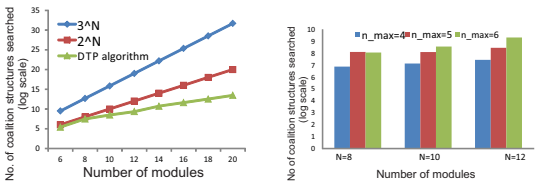
orientations are drawn from $\mathcal{U}[0,\pi]$.The battery left in each module is also drawn from $\mathcal{U}[1,100]$ and B^{max} is set to 100. n_{max} is set to 2 for $N = 4$; set to 4 for $N = 6$ and is set to 6 for $N \geq 8$. For all of the cases, all the modules were within each others communication range, which is a complete communication graph, which demonstrates the worst case scenarios for time complexity. Also, we did not notice significant effect in run time even if we have reduced the initial communication graph size by $10 - 20\%$. In the first set of experiments, we have shown the run time of DTP algorithm for different number of modules. Each test is run 5 times for different number of modules. For 4 modules, DTP algorithm takes 5.6 seconds in average and for 24 modules it takes 59 seconds. Figure 2.(a) shows the run time changes with the number of modules. This result proves that DTP algorithm scales better than existing algorithms like the BP algorithm (Dutta et al. 2013) that runs out of memory after 12 agents and the coalition search algorithm in (Rahwan et al. 2009) that takes around 90 mins for 27 agents. In the next set of experiments, we have compared the size of the original problem space, which is N^N, and corresponding run time to search that space. As can be seen in Figure 2.(b), even if the problem space is increasing exponentially, the run time is still within an affordable range as it is (almost) constant in log scale (polynomial in linear scale).



(a) (b)

Figure 3: (a) Run time comparison with searchUCSG algorithm, (b) Comparison of no. of coalition structures generated with BP algorithm

We have also compared the run time of the DTP algorithm against a previous centralized algorithm for reconfiguration planning, namely searchUCSG algorithm (Dutta et al. 2014). The result is shown in Figure 3.(a). As can be seen, the DTP algorithm outperforms the searchUCSG algorithm for higher number of modules. For example, for 12 mod-



(a) (b)

Figure 4: (a) Comparison of no. of coalition structures generated with existing algorithms, (b) Effect of changing n_{max}

ules, DTP algorithm takes 16.2 seconds on average, whereas searchUCSG takes 395 seconds. We have also compared DTP algorithm against the BP algorithm (Dutta et al. 2013), which is also a size constrained reconfiguration planning algorithm. The comparison with the BP algorithm in Figure 3.(b) shows that our algorithm performs better than the BP algorithm in terms of number of coalition structures generated. The comparison is shown in log_2 scale. By making the search procedure distributed and parallel in DTP algorithm, as opposed to BP, we are reducing time and space complexity largely. In both searchUCSG and BP algorithm, changing n_{max} from 4 to 6 caused the jumps between $N = 6$ and 8 in the graphs. Note that, even with the changes in n_{max} values, variance in DTP algorithm's performance is nominal.

The complexities of two existing coalition structure search algorithms, proposed in (Rahwan et al. 2009; Sandholm et al. 1999), are 2^N and 3^N respectively. Figure 4.(a) illustrates that using our algorithm, each module searches considerably fewer coalition structures than these algorithms and is able to prune unpromising search spaces intelligently. Notice that as the number of modules increases, the performance of DTP algorithm gets better than the others.

The number of coalition structures searched (in log_2 scale) for different N and n_{max} values is shown in Figure 4.(b). Here N is varied between $8, 10$ and 12 and n_{max} is varied through $4, 5$ and 6. As can be seen, higher the value of n_{max}, more coalition structures need to be searched by modules. So, the number of coalition structures generated is clearly a function of n_{max}. For example, with $n_{max} = 4$, number of coalition structures searched for $8, 10$ and 12 modules was $119, 270$ and 280 respectively. But with $n_{max} = 6$, these numbers went up to $176, 356$ and 650 respectively. So, we can see even with increasing n_{max} values, our proposed DTP algorithm is able to bound the search space within an affordable and realistic limit.

Figure 1.(b) provides empirical proof of the anytime nature of DTP algorithm. This test is done while varying N between $8, 10$ and 12. The result shows that in all the cases, after the third phase, the modules found the coalition structure with the highest utility. But right after the first round of searching, modules are able to achieve a good result in terms of percentage of best utility by finding a solution that is within $79.52\%, 74.38\%$ and 67.41% of the best utility for $N = 8, 10$ and 12 respectively. This result shows that our proposed DTP algorithm is anytime - it can first find a valid solution quickly and then improve that solution further.

Conclusions and Future Work

In this paper, we have introduced a novel distributed configuration generation algorithm. Our algorithm takes into account different constraints of the real world scenario such as communication, battery power and size of a configuration and finds the optimal configuration for the current situation. To the best of our knowledge, this work is the first approach to solve the constrained configuration generation problem in a distributed manner. We are currently trying to develop an algorithm to find the optimal value of n_{max} from different factors like terrain, current task etc. Finally we are planning to implement this algorithm on real MSRs.

References

Baca, J.; Hossain, S.; Dasgupta, P.; Nelson, C. A.; and Dutta, A. 2014. Modred: Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration. *Robotics and Autonomous Systems* 62(7):1002–1015.

Chirikjian, G.; Pamecha, A.; and Ebert-Uphoff, I. 1996. Evaluating efficiency of self-reconfiguration in a class of modular robots. *Journal of robotic systems* 13(5):317–338.

Dutta, A.; Dasgupta, P.; Baca, J.; and Nelson, C. 2013. A block partitioning algorithm for modular robot reconfiguration under uncertainty. In *European Conf. on Mobile Robots*, 255–260. IEEE.

Dutta, A.; Dasgupta, P.; Baca, J.; and Nelson, C. 2014. searchucsg: a fast coalition structure search algorithm for modular robot reconfiguration under uncertainty. *Robotica* 32(2):225–244.

Hou, F., and Shen, W.-M. 2008. Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 3135–3140. IEEE.

Kurokawa, H.; Tomita, K.; Kamimura, A.; Kokaji, S.; Hasuo, T.; and Murata, S. 2008. Distributed self-reconfiguration of m-tran iii modular robotic system. *The International Journal of Robotics Research* 27(3-4):373–386.

Michalak, T.; Sroka, J.; Rahwan, T.; Wooldridge, M.; McBurney, P.; and Jennings, N. R. 2010. A distributed algorithm for anytime coalition structure generation. In *Proc. 9th Intl. Conf. on Autonomous Agents and Multiagent Systems*, 1007–1014.

Prim, R. C. 1957. Shortest connection networks and some generalizations. *Bell system technical journal* 36(6):1389–1401.

Rahwan, T.; Ramchurn, S.; Jennings, N.; and Giovannucci, A. 2009. An anytime algorithm for optimal coalition structure generation. *J. Artif. Intell. Res. (JAIR)* 34:521–567.

Sandholm, T.; Larson, K.; Andersson, M.; Shehory, O.; and Tohme, F. 1999. Coalition structure generation with worst case guarantees. *Artificial Intelligence* 111(1-2):209–238.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence* 101(1):165–200.

Yim, M.; Shen, W.-M.; Salemi, B.; Rus, D.; Moll, M.; Lipson, H.; Klavins, E.; and Chirikjian, G. S. 2007. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE* 14(1):43–52.