Robot Workload Distribution in Active Sensor Networks

Weihua Sheng
School of ECE
Oklahoma State University
Stillwater, OK, 74078
email: weihua.sheng@okstate.edu

Girma S. Tewolde
ECE Department
Kettering University/Formerly GMI
Flint, MI 48504
email: gtewolde@kettering.edu

Abstract—This paper discusses the workload distribution problem in an Active Sensor Network, which integrates multiple sensor network-friendly mobile robots into a traditional sensor network and introduces adaptivity, self-healing, responsiveness and longer lifetime to the network. In order to distribute the workload among the service robots, first, a Service Set Partition (SSP) algorithm is developed to disperse the robots so that each robot can attend one subset of the sensors. Second, to service multiple sensors simultaneously, a Contract Network Protocol (CNP) is adopted to assign tasks to multiple service robots. Through CNP, certain performances such as energy efficiency and service time can be optimized. The proposed algorithms are verified through simulations.

I. INTRODUCTION

Wireless sensor networks can be used in many real world applications, such as environmental monitoring, search and rescue, military surveillance, and intelligent transportation [1], [2], [3], etc. In many situations, people do not have the luxury to carefully deploy the sensors. Instead, the sensors are deployed in large quantities very quickly. For example, in environmental monitoring and military surveillance, sensors can be dropped from airplanes. At disaster sites, search and rescue sensor networks are manually deployed by rescue workers in a quick fashion. The inherited problems with such traditional wireless sensor networks are: (1) It is very difficult to control the sensor density, coverage and connectivity of the sensor networks. (2) Once deployed, the sensor network can not adapt to a changing environment. (3) The overall life time of the sensor network is limited by the capacity of the batteries carried by the sensors and the network will not be able to carry out the mission if a critical number of sensors deplete their batteries. (4) There lack efficient methods to quickly determine the geographic locations of a large number of sensors while the location information is very important to the applications.

In our previous work [4], we developed a new sensor network architecture called active sensor network (ASN), which employs multiple sensor network-friendly micro service robots (µServBots) to implement an actuation mechanism and thus closes the loop. This active sensor network is contrary to traditional “open loop”, passive sensor network concept. Based on a set of core functions, the micro service robots can provide both logistic and network services. Examples of the logistic services include (1) sensing coverage control; (2) sensor power supply and (3) sensor calibration, etc. Examples of the network services include (1) network connectivity, or topology management; (2) hierarchical routing and (3) time synchronization, etc. Active sensor networks have many desirable merits, such as adaptivity, self healing, responsiveness and longer lifetime. The active sensor network architecture is illustrated in Figure 1, where logistic or network service requests are either initiated by the control center or by the sensors. Here the control center, which can be a soldier, a fire fighter, a rescue worker, or simply a computer, is not just an information sink [1] as in many traditional “open loop” sensor networks. Instead, it can actively generate commands to control and manage the underlying sensor network.

Embedding mobile robots in sensor networks has received some attention recently. Larmaca et al. [5] proposed a sensor network that uses a robot to carry out the following functions: deploy and calibrate sensors, detect and react to sensor failures. Corke et al. [6] used a UAV (an autonomous helicopter) to quickly deploy sensors for large-scale environmental monitoring purpose. They then used the UAV to discover the topology of the deployed sensor network and repair the network to achieve certain connectivity. Bychkovskiy et al. proposed a sensor network to investigate the control and actuation in data-centric wireless sensor networks [7]. Different from these existing work, our work aims to develop sensor-network-oriented robots that can be smoothly integrated into the underlying sensor network, as well as systematic models, approaches and methodologies to control and manage multiple mobile robots in the context of a sensor network.

In order to provide the logistic and network services, we identify the following four core functions that the µServBots should implement: (1) sensor localization; (2) sensor network-assisted inter-robot communication; (3) ser-
vice set partition and (4) task allocation. Sensor localization is the process of determining the geographic location of each sensor. Sensor-network-assisted inter-robot communication utilizes the underlying sensor network to provide a backup communication channel when two robots can not directly talk to each other. Service set partition aims to divide the sensors into multiple subsets so that each \( \mu \)ServBot is mainly responsible for one subset. This provides a scalable solution to the maintenance of a large network. Task allocation addresses how to distribute the given task among multiple \( \mu \)ServBots while maximizing the energy and time efficiency. Each high level service relies on one or more of the core functions provided by the \( \mu \)ServBots.

This paper focuses on the workload distribution among service robots. We assume that the sensors are already localized through the distributed localization algorithm developed in our previous paper [4]. The paper is organized as follows: Section II introduces the coordinated service set partition algorithm and proves its convergence. In Section III, the Contract Net Protocol is developed to assign tasks among multiple robots. Simulation results are provided in Section IV and Section V concludes the paper.

II. THE COORDINATED SERVICE SET PARTITION ALGORITHM

Due to the large number of sensors, it is hard for any single \( \mu \)ServBot to maintain the knowledge of the whole sensor network and service it in a global way. Therefore, it is necessary to partition the sensors into subsets and let each \( \mu \)ServBot play as the “housekeeper” for one subset, or in some situations, for the neighboring subsets as well. This local service model leads to the desired scalability. The service set partition can be carried out when needed. For example, when the sensor distribution is changed through sensor transportation, the service set partition will generate a new subset for each \( \mu \)ServBot.

The major criterion considered in the partition is load balance. That is, each \( \mu \)ServBot should have a roughly equal share of the service load, which can be defined as a function of the number of sensors, the distances to the sensors, etc. We propose a coordinated service set partition (SSP) algorithm. The basic idea of the coordinated SSP algorithm is that each \( \mu \)ServBot, starting from its initial position, is driven by certain virtual forces until a equilibrium state is reached. The virtual forces are determined by the service load differences between the current subset and the adjacent subsets. As illustrated in Figure 2, the SSP algorithm runs as follows.

SSP algorithm for \( \mu \)ServBot \( R_i \)

1. Starting from the initial Voronoi Diagram and the corresponding initial sensor subsets \( S_{s1}, S_{s2}, ..., S_{sn} \), where \( n \) is the number of sensor subsets *.
2. Calculate the service load, \( L_{si} \), of the corresponding subset \( S_{si} \) for all \( i = 1, 2, ..., n \).
3. If \( f_{ij} \geq f_{th} \) then change the velocity of \( R_i \) according to the following equation:

\[
v_i = v_i + (f_i - \lambda v_i)/m \cdot \Delta T
\]

(2)

4. Exchange new location information with neighbors and calculate the new Voronoi Diagram cell and update the sensor subset \( S_{si} \).

6. Go to (1).

In step (4), \( f_{th} \) is a small threshold value. In Equation (1), \( k \) is a scale factor and \( \vec{n}_{ij} \) is the unit vector from \( R_i \) to \( R_j \). In Equation (2), the term \( \lambda v_i \) introduces certain viscous force so that the deployment can be quickly stabilized. Considering the physical constraints of the real robots, we also limit the maximum acceleration rate and velocity to \( a_{max} \) and \( v_{max} \) respectively.

One concern is the convergence of the SSP algorithm. Here we prove the following lemma.

**Lemma 1**: The SSP algorithm leads to equal partition of the sensor nodes, or in other words, the partition process converges.

**Proof**:

The convergence of SSP algorithm can be proved using the Consensus theory [8]. First, we prove the convergence of an alternative SSP algorithm: area-SSP algorithm, which partitions the whole environment into equal-area Voronoi cells among all the \( \mu \)ServBots. The only difference between area-SSP algorithm and the above mentioned SSP algorithm is that the area-SSP algorithm uses the area of the Voronoi cell as the service load while the SSP algorithm uses the node count as the service load.

As shown in Figure 3, the area of a Voronoi cell \( i \) at time \( t \) is denoted as \( x_i(t) \). For cell \( i \), we assume there are \( M \) neighboring \( \mu \)ServBots and therefore there are \( M \) neighboring Voronoi cells. Due to the difference of area between cell \( i \) and any of its neighboring cells, for example cell \( j \), \( x_j(t) - x_i(t) \), the area-SSP algorithm will cause \( \mu \)ServBot \( j \) to move toward or away from \( \mu \)ServBot \( i \). The expelling or attracting force is proportional to the area difference.

\[
f_{ij}(t) = k(x_j(t) - x_i(t))\vec{n}_{ij}
\]
In time $\Delta t$, $\mu$ServBot $j$ moves from $p_j$ to $p_j'$. 

$$d(p_j, p_j') = \frac{1}{2} l_{ij}(t) (\Delta t)^2$$

The area change due to the movement of $\mu$ServBot is 

$$\Delta x_{ij}(t) = l_{ij}(t) d(p_j, p_j') = \frac{1}{2} l_{ij}(t) (\Delta t)^2$$

or 

$$\frac{\Delta x_{ij}(t)}{\Delta t} = \frac{1}{2} l_{ij}(t) \frac{f_{ij}(t)}{m_j} (\Delta t) = \frac{1}{2} l_{ij}(t) \frac{k(x_j(t) - x_i(t))}{m_j} (\Delta t)$$

or 

$$\dot{x}_{ij}(t) = \frac{1}{2} l_{ij}(t) \frac{k(x_j(t) - x_i(t))}{m_j} (\Delta t)$$

Summing up all the neighbors, we have 

$$\dot{x}_i(t) = \sum_{j=1}^{M} \dot{x}_{ij}(t) = \sum_{j=1}^{M} \frac{1}{2} l_{ij}(t) \frac{k(x_j(t) - x_i(t))}{m_j} (\Delta t)$$

The time step $\Delta t$ can be treated as a constant. Therefore we can rewrite the above equation as 

$$\dot{x}_i(t) = \sum_{j=1}^{M} \alpha_{ij}(t)(x_j(t) - x_i(t)) \tag{3}$$

where 

$$\alpha_{ij}(t) = \frac{1}{2} l_{ij}(t) \frac{k}{m_j} (\Delta t)$$

It can be clearly seen that equation (3) represents a linear dynamic system on a graph and the collective dynamics of the group of $\mu$ServBots following protocol (3) can be written as 

$$\dot{x} = -Lx \tag{4}$$

where $L = [l_{ij}]$ is the graph Laplacian of the network and its elements are defined as follows: 

$$l_{ij} = -\alpha_{ij}(t) \quad i \neq j, \quad l_{ii} = \sum_{j=1}^{M} \alpha_{ij}(t) \quad i = j.$$ 

According to Consensus algorithm theory [8], all $\mu$ServBots will reach an asymptotic consensus regarding the area of each cell, which implies that the area-SSP algorithm converges. Based on the convergence of the area-SSP algorithm, the convergence of the SSP algorithm is very similar. The only difference is that the service load is not the continuous area, but discrete number of sensor nodes. However, we can associate the number of sensors with the area through the following density expression: 

$$N(x, y) = \rho(x, y)A(x, y)$$

where $\rho(x, y)$ represents the node density at the local of position $(x, y)$. $N(x, y)$ represents the number of sensor nodes in the local area around position $(x, y)$. $A(x, y)$ represents the small area around position $(x, y)$. Then the discrete node count can be converted to the continuous area. Therefore the convergence of SSP algorithm can be guaranteed. 

III. TASK ALLOCATION IN ASN

The use of multiple $\mu$ServBots makes it possible to service the sensors faster and more efficiently. In a large sensor network, simultaneous service requests may come from different sensors. For example, a certain number of sensors may need to be transported to other locations in order to change the density of the sensor network, or a certain number of sensors need to be charged. If a sensor can only be serviced by its associated local $\mu$ServBot, it may be very inefficient. Additionally, different $\mu$ServBots may have different power levels, which may prohibit some $\mu$ServBots from executing certain tasks at hand. Therefore a critical question is how to allocate the task to different $\mu$ServBots to achieve efficiency and reduce cost. Obviously it is not scalable to let the control center make such decisions for a large sensor network involving many $\mu$ServBots. Therefore, the task allocation should be conducted in a distributed fashion, which means that a $\mu$ServBot should be able to communicate with its neighbors to solve the task allocation problem. We propose the following distributed task allocation method, which is modified from the Contract Net Protocol (CNP) [9].

We first assume the set of task types is enumerated as 

$$TT = \{Power \ Supply, \ Node \ Transportation, \ Sensor \ Calibration, \ Hierarchical \ Routing,...\}$$. An atom task can be defined as a combination of a task type and the location to carry out this type of task, or $T =< TT_i, L >$ where $TT_i \in TT$ and $L$ is the associated location. A complex task, or simply task, is defined as a collection of atom tasks, or $T = \{T_1, T_2, ..., T_k\}$. Tasks can be initiated by the control center in a top-down fashion, or by sensors in a bottom-up fashion. For example, coverage holes maybe detected by the control center and the node transportation task can be naturally initiated by the control center. On the other hand, low battery level is usually detected by the sensors and charge requests can be sent to the associated $\mu$ServBot. Therefore we can assume there is a coordinator $\mu$ServBot that receives this task, either from the control center or nearby sensors, and manages to distribute the task among multiple neighboring $\mu$ServBots. As shown in Figure 4, There are four steps to allocate the task.

226
(1) Task announcement. The task, or the list of atom tasks $T = \{T_1, T_2, ..., T_k\}$ are sent to the $m$ $\mu$ServBots in the adjacent cells, or to the $m$ $\mu$ServBots within a fixed number of hops.

(2) Bid response. Each $\mu$ServBot $R_i$ receiving the announcement evaluates each individual atom task $T_j$ on the list and calculates a cost $C_{ij}$ that represents the total cost (such as energy and time consumption) for $R_i$ to carry out $T_j$. If a $\mu$ServBot does not have enough energy to complete this atom task, the cost will be infinity. Then the bid, or the list of costs are sent back to the coordinator.

(3) Task allocation. Upon receiving all the bid responses, the coordinator makes a decision on which $\mu$ServBot gets which atom tasks. Basically, it can be formulated as a weighted set partitioning problem [10].

$$\text{minimize } z = \sum_{i,j} C_{ij} \cdot x_{ij}$$

$$\text{s.t. } \sum_{i=1}^{m} x_{ij} = 1, j = 1, ..., k$$

$$x_{ij} \in \{0, 1\}$$

where variable $x_{ij} = 1$ means $R_i$ gets atom task $T_j$, $x_{ij} = 0$ otherwise. Equation (6) guarantees that every atom task will be executed by one and only one $\mu$ServBot.

(4) Task award. The coordinator notifies the $\mu$ServBots that are awarded the atom tasks.

If the service request is from the control center, it is easy to designate a coordinator that is close to the sensors concerned. If the service requests come from any of the nodes in a partition that a $\mu$ServBot is responsible for, the $\mu$ServBot in that partition will be the coordinator. So the moment a $\mu$ServBot receives service requests from any of the static sensor nodes in its partition, it decides to invite all the mobile nodes in the surrounding neighborhood to participate in a bid to execute the tasks. Based on their distances to the task locations, their current state (busy, idle, or dead), the energy level, and time delay before the robot can start executing the task, each robot computes its estimated cost to execute each of the tasks and sends its bid to the coordinator robot. After receiving all the bids, the coordinator robot determines for each task the $\mu$ServBot that can execute the task with the minimum cost possible and notifies those $\mu$ServBots that are awarded tasks.

IV. SIMULATION RESULTS

We simulate our SSP algorithm and the CNP algorithm in MATLAB.

A. Service Set Partition

To verify the SSP algorithm, we select the following parameter settings: force constant $k = 0.04$, viscous friction constant $\lambda = 0.5$, mass of $\mu$ServBots $m = 1.0 Kg$, maximum acceleration $a_{\text{max}} = 5.0 m/s^2$, maximum velocity $v_{\text{max}} = 10.0 m/s$, time step $\Delta T = 0.1 s$. The algorithm is allowed to run for a maximum of 1000 iterations. But during the iterations if the maximum displacement executed at some point is less than a small preset value of $\varepsilon = 0.001$, then the program will terminate.

Figure 5 shows the plots of the locations of the initial deployment (obtained from the application of the potential field method) of seven $\mu$ServBots in a total service area of $40m \times 40m$. As it could be visually verified this distribution gives an approximately equal partitioning of the total service area into 7 cells. But the 50 randomly distributed sensor nodes give an unbalanced distribution of 7, 5, 9, 6, 10, 9, and 4 nodes in the seven cells. Then the SSP algorithm was applied to achieve a new partition of the total service area with a more balanced distribution of 7, 7, 7, 8, 7, 7, and 7 nodes in the seven respective service cells, as shown in Figure 6. The figure also shows the trajectories followed by each $\mu$ServBot during the execution of the SSP to arrive at the final deployment.

To verify the effectiveness of the SSP algorithm over a
TABLE I
SIMULATION RESULTS OF THE SSP ALGORITHM

<table>
<thead>
<tr>
<th>Service area</th>
<th>m</th>
<th>n</th>
<th>20 × 20</th>
<th>25 × 25</th>
<th>30 × 30</th>
<th>40 × 40</th>
<th>50 × 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial deployment</td>
<td>4</td>
<td>5</td>
<td>7, 3, 11, 3, 6</td>
<td>6, 6, 6, 6, 6</td>
<td>6, 5, 7, 9, 8, 8</td>
<td>7, 7, 8, 7, 7, 7</td>
<td>6, 7, 8, 8, 7, 7, 8</td>
</tr>
<tr>
<td>Final deployment</td>
<td>5</td>
<td>6</td>
<td>7, 3, 11, 3, 6</td>
<td>6, 6, 6, 6, 6</td>
<td>6, 5, 7, 9, 8, 8</td>
<td>7, 7, 8, 7, 7, 7</td>
<td>6, 7, 8, 8, 7, 7, 8</td>
</tr>
</tbody>
</table>

Fig. 7. Position of the static sensors (blue circles) and the \( \mu \text{ServBots} \) (the red colored Ri's) during test case 4. The CNP-based task allocation algorithm assigned tasks T1, T2, T3 respectively to ServBots R5, R2, R3; while the arbitrary task assignment assigned the tasks to R3, R5, R2, respectively.

wide range of test cases we run the program on different sizes of service area, number of \( \mu \text{ServBots} \), number of static sensor nodes, and achieved the results as shown in Table I.

We also verified the adaptivity of the SSP algorithm in the face of changing load conditions, for example due to random node failures. To emulate the load changes the system randomly picks one static sensor at a time and kills it and allows the SSP algorithm to recompute new service set partitions in response to the load changes. This continues at a preset death rate of the static nodes. We have found that the SSP algorithm is adaptive to the changing load conditions and is able to modify the partitions so as to maintain the load balance.

B. Task Allocation

The CNP algorithm is implemented using the following parameter settings: (1) Number of static sensors = 50; (2) Number of \( \mu \text{ServBots} = 5; \) (3) Dimension of the region over which the sensors are distributed = 40 x 40. To evaluate the merit of this CNP-based task allocation algorithm compared with other methods, the following performance criteria are used: (1) Total distance traveled by all the robots in executing a given set of tasks; (2) Total time to complete the tasks (given task execution times and assuming the travel time of the robots to the task locations to be proportional to the distances).

The whole region is partitioned into 5 areas using the SSP algorithm to equally divide the number of sensors that each \( \mu \text{ServBot} \) is responsible for. Three tasks are generated from three randomly chosen sensors in one of the partitions of the region. The scenario is shown in Figure 7. The tasks are assigned normalized execution times from normally distributed random numbers with a mean of 0.5 and a standard deviation of 0.1. We then randomly assign one of three possible current states to each of the 5 \( \mu \text{ServBots} \). The possible states are: Dead (D), Idle (I), or Busy (B). For the busy \( \mu \text{ServBots} \), we assign a randomly generated number as the time delay for the node to be available after it finishes its current task. The value of this time delay could be between 0.0 to 1.0 and is considered as a fraction of the maximum possible task execution time. For those \( \mu \text{ServBots} \) in Idle or Busy states, we randomly assign a current energy level (a number between 0.0 and 1.0). A value of 0.0 corresponds to a robot with no energy left, and a value of 1.0 is one with full energy. For the purpose of the simulation we assume that a \( \mu \text{ServBot} \) with 100% full energy level on the average will be able to execute 10 tasks of the average duration (1.0).

Each \( \mu \text{ServBot} \) that bids for tasks computes estimates of the energy cost of executing each of the tasks. This is done by combining the cost components of distance, energy level and the \( \mu \text{ServBot} \)'s state of availability and the time delay before it finishes its current task and become ready to take on a new task. For each task, before placing a bid, a \( \mu \text{ServBot} \) determines whether its available energy is enough to execute the task. This decision is based on the execution time of the task, its distance from the \( \mu \text{ServBot} \) and the \( \mu \text{ServBot} \)'s energy level. If a \( \mu \text{ServBot} \) wouldn’t have enough energy to execute a certain task, it wouldn’t bid for that particular task. To travel to the task locations, we assume that the \( \mu \text{ServBot} \) spends energy that is proportional to the distance.

For simplification, a greedy algorithm is developed to solve the CNP-based task assignment problem, although more complicated set partition algorithms can be used [10]. The winner of the bid for each of the tasks will be the \( \mu \text{ServBot} \) with the smallest net cost. Only one task will be assigned to one \( \mu \text{ServBot} \). Thus the task assignment strategy is to evaluate all the submitted bids and pick the one with the smallest cost and assign it to the corresponding \( \mu \text{ServBot} \) that made this offer. This procedure is repeated until all the tasks have been assigned.

To evaluate the performance of our CNP-based task allocation algorithm we run the system a number of times, where in the different runs the program uses random numbers for the parameters such as the task locations, their execution time, the states of the \( \mu \text{ServBots} \), their energy level, and the time delay before any of the \( \mu \text{ServBots} \) could become available to take on new tasks. The performance criteria stated earlier, i.e. the total distance traveled by all the \( \mu \text{ServBots} \) to accomplish all the tasks and the time it takes to complete all the tasks (this includes the time required to travel to and from the task locations and the task execution times), are used.

For comparison, we also implemented a simplified arbitrary task assignment algorithm as follows. The objective of the simplified task assignment strategy is to keep the task assignment procedure very simple, it doesn’t care...
about computing energy associated with distances, execution times, delays, etc. It makes the bold assumption that if the energy level of a ServBot is greater than a certain threshold level (assumed to be 0.2 in the simulation), then it will be able to execute any task. So the strategy is first to check if the ServBot that is responsible for the region where all the tasks are generated has an energy level greater than the threshold value. If this is the case, then this host ServBot gets one of the tasks. Otherwise, the host ServBot would not be assigned any task. Any remaining tasks are then assigned arbitrarily to the other ServBots only if they have energy higher than the threshold value.

The simulation results are summarized in Tables II and III, respectively. The results show that the total distance traveled by all the ServBots in the CNP-based task assignment is smaller than that of the arbitrary task assignment. The CNP-based algorithm tends to minimize the energy cost of executing the tasks. The algorithm also results in quicker completion of task executions. Also, as one of the criteria in the cost computation is to give preference to ServBots that have higher energy levels, the algorithm tends to increase the life of the system by refraining from using ServBots with low energy levels.

V. CONCLUSIONS

This paper presents two algorithms related to the robot workload distribution in Active Sensor Networks. The goal is to maximize the performance of the ASN in terms of energy efficiency, responsiveness, lifetime, etc. The SSP algorithm deploys the service robots in the sensor network and partitions the sensor nodes equally. The convergence of the SSP algorithm resembles the convergence of the Consensus algorithm. The CNP algorithm assigns multiple tasks among multiple service robots. The assignment is carried out through the interaction between neighboring service robots and a local coordinator, which makes it scalable to large ASNs. A simple example and a greedy algorithm are developed to illustrate the task allocation idea in our simulation. We are going to investigate other task allocation algorithms in our future work.

REFERENCES


TABLE II

<table>
<thead>
<tr>
<th>Robots</th>
<th>State/Delay/Energy</th>
<th>Tasks/Ex. Time/Dist. from</th>
<th>Net Cost</th>
<th>CNP</th>
<th>Arbitrary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total round trip distance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>B/0.9247/0.8710</td>
<td>39.1257</td>
<td>25.2521</td>
<td>0.2524</td>
<td>0.2534</td>
</tr>
<tr>
<td></td>
<td></td>
<td>120.3</td>
<td></td>
<td>13.31</td>
<td>13.56</td>
</tr>
<tr>
<td>R2</td>
<td>B/0.9795/0.1830</td>
<td>20.7464</td>
<td>16.6838</td>
<td>0.3493</td>
<td>0.3496</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>1.574</td>
</tr>
<tr>
<td>R3</td>
<td>R/0.4248/0.7252</td>
<td>4.1048</td>
<td>0.0836</td>
<td>1.163</td>
<td>1.166</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>2.0711</td>
</tr>
<tr>
<td>R4</td>
<td>R/0.03/0.8585</td>
<td>14.2384</td>
<td>11.4929</td>
<td>0.0910</td>
<td>0.0856</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>40.2520</td>
</tr>
<tr>
<td>R5</td>
<td>R/0.01/0.165</td>
<td>20.2014</td>
<td>13.4802</td>
<td>1.3051</td>
<td>1.277</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>6.5349</td>
</tr>
</tbody>
</table>

TABLE III

<table>
<thead>
<tr>
<th>Robots</th>
<th>State/Delay/Energy</th>
<th>Tasks/Ex. Time/Dist. from</th>
<th>Net Cost</th>
<th>CNP</th>
<th>Arbitrary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total round trip distance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>R/Inf/0.0</td>
<td>43.1429</td>
<td>37.1142</td>
<td>1.163</td>
<td>1.166</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>70.58</td>
</tr>
<tr>
<td>R2</td>
<td>B/0.5865/0.3785</td>
<td>20.7464</td>
<td>14.5806</td>
<td>0.2639</td>
<td>0.2605</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>1.781</td>
</tr>
<tr>
<td>R3</td>
<td>R/0.01/0.7048</td>
<td>2.0711</td>
<td>4.1190</td>
<td>0.0615</td>
<td>0.0615</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>2.0711</td>
</tr>
<tr>
<td>R4</td>
<td>R/0.03/0.8461</td>
<td>12.6719</td>
<td>19.2056</td>
<td>0.0438</td>
<td>0.0416</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>2.0711</td>
</tr>
<tr>
<td>R5</td>
<td>R/0.1650/0.5863</td>
<td>19.1646</td>
<td>13.8989</td>
<td>1.163</td>
<td>1.166</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inf</td>
<td>2.0711</td>
</tr>
</tbody>
</table>