Autonomous Demand-Side Management System Based on Monte Carlo Tree Search

Edgar Galván-López [#1], Colin Harris [#1], Leonardo Trujillo [*2], Katya Rodriguez-Vazquez [+3], Siobhán Clarke [#1] and Vinny Cahill [#1]

[#] *Distributed Systems Group, School of Computer Science & Statistics, Trinity College Dublin, College Green, Dublin 2*
[1] edgar.galvan, colin.harris, siobhan.clarke, vinny.cahill @ scss.tcd.ie

[*] *Doctorado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, México*
[2] leonardo.trujillo@tectijuana.edu.mx

[+] *Instituto en Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, México*
[3] katya.rodriguez@iimas.unam.mx

***Abstract*—Smart Grid (SG) technologies are becoming increasingly dynamic, motivating the use of computational intelligence to support the SG by *predicting* and *intelligently* responding to certain requests (e.g., reducing electricity costs given fluctuating prices). The presented work intends to do precisely this, to make intelligent decisions to switch on electric devices at times when the electricity price (prices that change over time) is the lowest while at the same time attempting to balance energy usage by avoiding turning on multiple devices at the same time, whenever possible. To this end, we use Monte Carlo Tree Search (MCTS), a real-time decision algorithm. MCTS takes into consideration what might happen in the future by approximating what other entities/agents (electric devices) might do via Monte Carlo simulations. We propose two variants of this method: (a) max^n MCTS approach where the competition for resources (e.g., lowest electricity price) happens in one single decision tree and where all the devices are considered, and (b) two-agent MCTS approach, where the competition for resources is distributed among various decision trees. To validate our results, we used two scenarios, a rather simple one where there are no constraints associated to the problem, and another more complex, and realistic scenario with equality and inequality constraints associated to the problem. The results achieved by this real-time decision tree algorithm are very promising, specially those achieved by the max^n MCTS approach.**

***Index Terms*—Demand-Side Management Systems, Monte Carlo Tree Search.**

I. Introduction

A Smart Grid (SG) is defined as a type of electrical power grid whose goal is to respond to the behaviour and actions of energy suppliers and consumers to efficiently deliver economic, reliable and sustainable electricity services.

A research area that has been very popular within SGs is Demand-Side Management (DSM), as shown by the increasing number of publications over recent years [1], [2], [3], [4], [5], [6], [7], [8], [9]. Figure 1 depicts a number of scientific papers, per annum, published on 'demand-side management' systems. DSM is a set of measures to improve the energy system at the consumer side. DSM ranges from improving energy efficiency through the use of better insulation or better materials up to the use of autonomous systems to control energy resources [7].

We focus our attention on the latter. That is, we make an effort to develop an autonomous DSM system to control energy consumption at the user side by load-shifting energy consumption. Most of the existing DSM works have focused their attention on the analysis of load-shifting energy consumption at a broad scale (e.g., control of multiple electric vehicles) and the interaction between a utility company and its users [6]. The latter has the consequence that a decision made for a specific electric device might not be the optimal because it does not consider what other devices might do.

In this paper, we take a more fine-grained approach by using a household scenario where the goal is to automatically control electric devices to load-shift energy consumption by trying to reduce electricity costs. To do so, we use a real-time decision algorithm denominated Monte Carlo Tree Search (MCTS). This is a sampling method for finding *optimal decisions* by performing random samples in the decision space and building a tree according to partial results. The decision tree is built by taking into consideration what other agents or entities might do (in this particular case, the MCTS controls each electric device by turning it on or off depending on a set of conditions, such as electricity cost, time needed by the appliance to finish a task at a particular moment – details of this are given in Section III). Furthermore, this paper makes a bridge between the use of MCTS to support a DSM system. To the best of our knowledge, this work and that presented by Wijaya et al. [10] are one of the first works that uses the MCTS algorithm in a DSM system.

The problem, formally defined in Section IV, consists in intelligently and automatically controlling electric devices within a household unit so that each of them finishes a task (e.g., washing machine) or finishes being charged (e.g., electric vehicle). We encourage an efficient energy consumption based on an electricity signal (e.g., a high correlation is observed

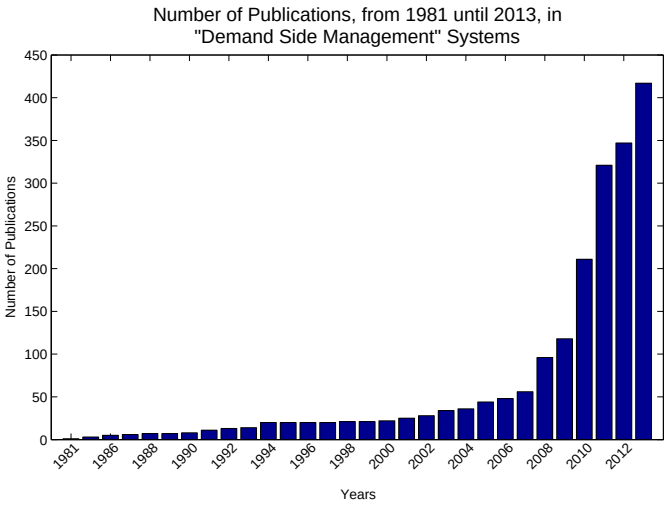

Fig. 1. Number of papers published from 1981 until 2013 in "Demand Side Management" Systems. Source: IEEE Xplore database.

between high electricity cost and a high number of devices being switched on at the same time). So, we let the MCTS manage each of the devices in such a way that there is a low number of devices being switched on at the same time, whenever possible. To make the problem more interesting, we set some equality and inequality constraints that must be considered when controlling these devices.

This paper is organised as follows. In the next section, we set the foundations of our work by presenting how MCTS works. In Section III we describe how we have used it for our energy problem. Section IV presents the experimental setup used to conduct our experiments. In Section V we present and discuss our findings. Finally, in Section VI we draw some conclusions.

II. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a sampling method for finding *optimal decisions* by performing random samples in the decision space and building a tree according to partial results. In a nutshell, Monte Carlo methods work by approximating future rewards that can be achieved through random samplings. The evaluation function of MCTS relies directly on the outcomes of simulations. Thus, the accuracy of this function increases by adding more simulations, so the optimal search tree is guaranteed to be found with infinite memory and computation [11]. However, in more realistic scenarios (e.g., limited computer power), MCTS can produce very good approximate solutions.

MCTS has gained a lot of popularity thanks to its recent success in the board game of computer Go [12], where the space of solutions is 10^{170} and up to 361 legal moves. A problem, that until recently, was highly difficult for Artificial Intelligence (AI), and considered much more harder than Chess, and where MCTS has obtained excellent results, including achieving dan – master – level at the 9 x 9 board game. This is, perhaps, the reason why MCTS has been used heavily in two-player board games.

However, the diversification of MCTS in other research areas is already a reality. For instance, MCTS has been explored in combinatorial optimisation (e.g., traveling salesman problem [13]), constraint satisfaction (e.g., mathematical expression [14], constraint problems [15]), and other types of games [12], [16] (see [17] for a more detailed list of applications using MCTS). The use of MCTS in different research areas can give a good idea on the success of MCTS on challenging and interesting problems.

A. The Mechanics Behind MCTS

MCTS relies on two key elements: (a) that the true value of an action (in our problem, an action could be either turning an electric appliance on or off at a particular time based on a pricing signal, as briefly discussed in Section I) can be approximated using simulations; and (b) that these values can be used to adjust the policy towards a best-first strategy. The algorithm, explained later in this section, builds a partial tree, guided by the results of previous explorations of that tree.

The algorithm iteratively builds a tree until a condition is reached or satisfied (e.g., number of simulations, time given to perform Monte Carlo simulations), then the search is halted and the best performing action is executed. In the tree, each node represents a state, and directed links to child nodes represents actions leading to subsequent states.

Like many AI techniques, MCTS has several variants. Perhaps, the most accepted steps involved in MCTS are those described in [17] and are the following: (a) *Selection,* a selection policy is recursively applied to descend through the built tree until an expandable (a node is classified as expandable if it represents a non-terminal state, and also, if it has unvisited child nodes) node has been reached; (b) *Expansion,* normally one child is added to expand the tree subject to available actions; (c) *Simulation,* from the new added nodes, a simulation is run to get an outcome (e.g., reward value); and (d) *Back-propagation,* the outcome from the simulation step is backpropagated through the selected nodes to update their statistics.

Simulations in MCTS start from the root state (in our case, from the current time when an action for an electric device should be made) and are divided in two stages. When the state is added in the tree, a *tree policy* is used to select the actions (the selection step is a key element and it is discussed in detail later in this section). Otherwise, a *default policy* is used to roll out simulations to completion. Both stages are depicted in Figure 2. More formally, Algorithm 1 presents the typical steps involved in designing a MCTS method.

One element that contributed to enhance the efficiency in MCTS was the selection mechanism proposed in [11]. The main idea of the proposed selection mechanism was to design a Monte Carlo search algorithm that had a small probability error if stopped prematurely and that converged to the optimal solution given enough time. That is, a selection mechanism

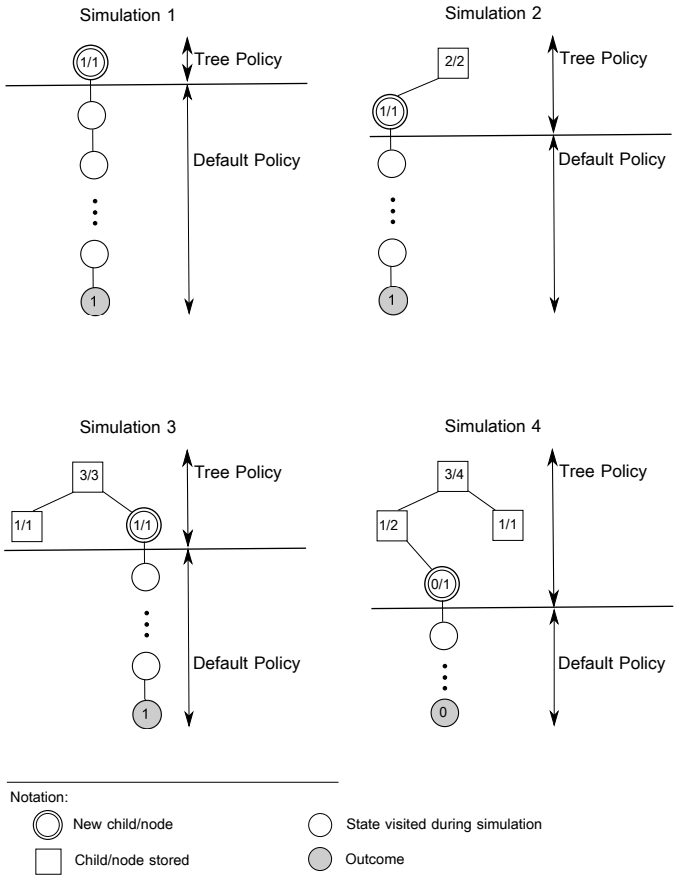

Fig. 2. Four simulations. In Simulation 1, a new node is added to the tree search and its statistics are updated (e.g., outcome and number of visits). Simulation 2, stores the first node and adds a new one. Simulation 3 adds a new node according to the other action (in this case there are only two actions, as indicated by the number of children). Simulation 4 selects (in this case it is a tie, so a random selection is used) and beneath the selected node, a new one is added.

that nicely balances exploration *vs.* exploitation, explained in the following paragraphs.

B. Upper Confidence Bounds for Trees

As indicated previously, MCTS works by approximating "real" values of the actions (in this work an action can be turning a device on or off) that may be taken from the current state. This is achieved through building a search or decision tree. The success of MCTS depends heavily on how the tree is built and the selection process plays a fundamental role in this. One particular selection mechanism that has proven to be very reliable is the UCB1 tree policy [11]. Formally, UCB1 is defined as:

$$UCT = \bar{X}_j + 2K\sqrt{\frac{2 \cdot ln \cdot n}{n_j}} \quad (1)$$

where n is the number of times the parent node has been visited, n_j the number of times child j has been visited and $K > 0$ is a constant. In case of a tie for selecting a child node, a random selection is normally used [11].

Algorithm 1: MCTS Algorithm.

```
//N might be between 100 and 1,000,000
while n < N do
    //set up data structure to record
    line of play
    visited = new List<Node>()
    //select node to expand
    node = root
    visited.add(node)
    while node is not a leaf do
        node = select(node, node.children) //e.g.,
        UCT selection
        visited.add(node)
    //add a new child to the tree
    newChild = expand(node)
    visited.add(newChild)
    value = rollOut(newChild)
    foreach node:visited do
        //update the statistics of tree
        nodes traversed
        node.updateStatus(value)
    n++
```

Thus, this selection mechanism works due to its emphasis on balancing both exploitation (first part of Eq. 1) and exploration (second part of Eq. 1). That is, every time a node is visited, the denominator of the exploration part increases resulting in decreasing its overall contribution. If, on the other hand, another child node of the same parent node is visited, the numerator increases, so the exploration values of unvisited children increase. The exploration term in Eq. 1 guarantees that each child node has a selection probability greater than zero, which is essential given the random nature of the playouts.

III. Modelling our Problem in MCTS

MCTS has several variants and there is no general consensus on how to best apply it. However, there are common elements in MCTS, as presented in Section II, that can give good initial insights. As mentioned before, MCTS has been used heavily in two-player based board games of different complexity, ranging from easy (e.g., phantom tic-tac-toe [18]) to really difficult (e.g., Computer Go [12]). For a situation where the number of players or agents is greater than two, which is the kind of problem used in this work (use of multiple electric devices), one needs to extend or modify the MCTS algorithm.

We explored two variants of the MCTS algorithm: (a) a two-agents approach, as done in two-player based board games, and (b) a max^n approach, that considers multiple entities (e.g., electric devices) competing against each other to maximise their respective payoffs.

Before describing our approach with these two variants, it is necessary to formally define our problem.

A. Problem Statement

We are interested in investigating whether it is possible to minimise electricity costs, by learning the optimal times to switch electric devices on or off, while at the same time trying to avoid turning on all electric devices at the same time whenever possible, by means of MCTS as a real-time decision algorithm. To this end, we propose two scenarios to test our approach, these are:

- Scenario I. In this scenario each device (four devices in total) needs to finish its task, by taking an action in a period of time with a granularity of 30 minutes. More specifically we used a clothes dryer, electric vehicle, clothes washer and dishwasher where each of them need one and half hours, two hours, one hour and 30 minutes to finish their tasks, respectively. Notice that in this scenario the only constraint is that the clothes dryer should start working after the clothes washer has finished.
- Scenario II. This is a more challenging and realistic scenario with a larger set of constraints, different range of times is specified where each of the devices can be used and different length of time needed for each of the devices is imposed. See Table I for details.

For both scenarios, we considered a 24-hour period where each of the devices can start being used or charged, so it is within this range of time that a decision can be made, subject to the constraints associated to the problem (see Table I).

B. Methodology

MCTS, as mentioned previously, has been used heavily in two-player board games. In this work, we started using this approach and then we used a variation of a max^n approach.

1) Two-Agent Approach: For this first approach, we made a slight variation of the algorithm by building two trees, each containing two devices (recall that we used four devices as explained earlier in this section) and made the devices, within each tree, to compete against each other, in order to find the best time to turn on a device at the lowest possible electricity price, and also, that each device finishes its corresponding task. Given the constraints of the problem (e.g., clothes dryer should start after the clothes washer has finished its task), we placed these two devices in the same tree, and the other two devices, electric vehicle and dish washer, in a different tree.

2) Max^n Approach: For the second approach, we modified our MCTS approach based on the max^n method [19]. That is, each of the four devices used in this work, tries to maximise their own payoffs (e.g., highest reward when the MCTS turns on a device at the lowest possible price), while at the same time considering what the rest of the agents (devices) might do, which are also trying to maximise their own rewards. This is achieved by adding all the devices to the decision tree, in the following way: a device is added to the tree and performs a roll-out (see Section II for a complete description of the algorithm) by competing against each of the devices, once this is finished, the next device is added to the decision tree in the same manner, and this continues until all the devices have been considered at that particular time. This process is repeated until a termination criterion is satisfied (this is explained in the following paragraphs). This method was inspired by the technique used by Samothrakis et al. [16], where the authors achieved extraordinary results in a dynamic environment.

C. Defining Rewards

As explained in Section II, MCTS relies heavily on two factors: (a) that the true values of an action (on or off) can be approximated using simulations, and (b) that these values can be adjusted to a policy towards a best-first strategy. These values correspond to the rewards assigned to each of the agents/devices based on a decision made at a particular time.

We defined these rewards by considering two elements: (a) devices that are turned on at the lowest electricity price while avoiding that a majority of them are turned on at the same time, and (b) by considering that the constraints, presented in the previous paragraphs, were satisfied (e.g., devices finishing their tasks). Thus, we rewarded agents 0.5 if they executed an action at the lowest electricity price and no other device was turned on at the same time, and 0.5 if they finished their corresponding tasks. Thus, it is clear that we gave equal importance to both objectives.

D. Termination Criteria

Since there is always the same number of actions at each time step, the tree does not naturally terminate as is the case in two-player board games. Therefore, the depth of the tree can grow astronomically in proportion to the number of simulations being run. This affects how the tree is able to effectively model the problem and could yield suboptimal results.

We therefore introduced some terminating conditions to limit the depth of the tree and tune the structure of the tree to model the problem more accurately. The tree is expanded until the last time slot is reached or until the optimal time required for the device to finish its task has been selected, whichever condition occurs first. We discovered that using both the total number of time slots (range period) and the required number of time slots to finish a task as terminating conditions to limit the depth of the tree improves the results significantly, and we have used these to conduct our experiments.

IV. Experimental Setup

We modelled the problem under investigation by running experiments for 100 days. Each day is divided into 30-minute slots, as indicated in Section III. Agents for each of the devices make a decision at each time slot, whether to switch a device on or off. The action taken at each time slot is determined by the algorithm and considers the price profiles, time required for the device to complete its task and other factors like the time range within the day that the device is constrained to work (that is, earliest start time and latest finish time, as indicated in Table I). We run 10,000 simulations at each time step (roll-out, see Figure 2).

TABLE I
EQUIPMENT AND ASSOCIATED VARIABLES. EST, LFT INDICATE EARLIEST START TIME AND LATEST FINISH TIME, RESPECTIVELY.

	Clothes Dryer	*Electric Vehicle*	*Clothes Washer*	*Dishwasher*
Earliest Start Time	EST_{CD}	EST_{EV}	EST_{CW}	EST_{DW}
Latest Finish Time	LFT_{CD}	LFT_{EV}	LFT_{CW}	LFT_{DW}
Time Constraints	EST_{CD}=10:30 LFT_{CD}=17:00	EST_{EV}=5:00 LFT_{EV}=10:30	EST_{CW}=4:00 LFT_{CW}=12:30	EST_{DW}=21:00 LFT_{DW}=23:30
Time Needed	1.30 hours	2 hours	1.30 hours	30 minutes
Slots Needed	(3 slots)	(4 slots)	(3 slots)	(1 slot)
Other Constraints	$\text{EST}_{CD} > \text{EST}_{CW}$	–	$\text{EST}_{CW} < \text{EST}_{CD}$	–

To test our approach, using both variants, two-agent and max^n approach, as explained in Section III, we used different pricing history files, where each of them contains three different prices (low, medium and high). By considering this, we are in complete control of analysing our results and indicating whether our approach was able or not to find the optimum solution.

The dynamic environment is simulated by using these pricing history files that are different for each of the 100 days simulated in our experiments. To avoid biasing our system, we guarantee that we have, on average, the same number of low, medium and high prices for the 24-hour period used. That is, by having a 24-hour period and the system being able to make a decision every 30 minutes, we have 16 number of time slots for each of the three prices defined in this work (low, medium and high) as indicated previously. This number of time slots also allows the system, in principle, to complete the indicated task for each of the four electric devices used in our experiments.

V. RESULTS AND ANALYSIS

We are interested in seeing if it is possible to reduce electricity costs by predicting prices, switching on electric devices at the lowest possible price and avoiding turning on multiple devices at the same time. This task would be straightforward to solve if it was not for the presence of the constraints associated to the problem, as expressed in Section III. The problem also gets harder in a dynamic environment (i.e., fluctuations of electricity prices).

A. Completed Tasks and Associated Constraints

Let us first focus our attention on how good is MCTS at controlling devices so that they can achieve a particular task (denoted as time needed in Table I). For Scenario I (no constraints as explained in Section IV), and using the two-agents approach, shown at the left-most hand side of Figure 3, it is clear that this MCTS approach is able to control all electric devices (i.e., clothes dryer, electric vehicle, clothes washer and dishwasher denoted as CD, EV, CW and DW, respectively) such that all of them finish their corresponding tasks. However, only one electric device was able to be used at times with the lowest electricity price (dish washer). The performance of the two-agent MCTS approach is also good for the CD and the EV. That is, for the CD, the MCTS is able to turn on the clothes dryer more often when the electricity cost is the cheapest and the same situation is observed to controlling the electric vehicle. However, it is also fair to mention that this approach prefers to switch on the clothes washer slightly more often (just over 50% of the time) when the electricity cost is the highest.

If we now turn our attention using our max^n MCTS approach in the same scenario, Scenario I as explained in Section III, we can see in the right-hand side of Figure 3, that this approach is able to use all the devices at times where the electricity price is the lowest, while at the same time guaranteeing that all tasks are completed.

From these findings, it is clear that the max^n MCTS approach is superior, in terms of finding optimum solutions, compared to the two-agent MCTS approach. This is not surprising, since as we discussed in Section III, the former approach takes into consideration all the agents involved in the process of learning at what time a device should be turned on based on electricity prices, so each agent tries to maximise its own payoff, while at the same time taking into consideration what the other agents might do.

Now, let us focus our attention on the MCTS' performance in Scenario II, using both the two-agents and the max^n approach. Scenario II is a more challenging problem compared to Scenario I, given the equality and inequality constraints associated to the problem, as explained in Section IV.

Figure 4 shows the results when using the two-agent approach (left-hand side) and the max^n approach (right-hand side) in this scenario. It is clear that the two-agent approach performs badly in this case. That is, it fails in ensuring that an electric device finishes its task. For example, the CD ran for less than 80% of the time, which means that it was unable to finish its task of fully drying clothes. The situation is worse for the DW where the two-player approach failed to turn on the device, regardless of the electricity price associated at any particular time. There is a mixed picture for those devices that were able to finish their tasks. For example, for the EV, one third of the time this approach preferred to turn it on at the lowest price. The rest of the task was completed by turning it on using the second lowest price and the highest price. Finally, the CW is able to finish its task, but it fails at running when the price is the cheapest.

If we now turn our attention to the max^n MCTS approach's performance in Scenario II (right-hand side of Figure 4), we can see that it has a much better performance compared to

the two-agent approach (left-hand side of Figure 4). That is, the max^n approach is able to turn a device on at the lowest electricity price all the time, except for the EV, where around one third of the time this approach picked the second cheapest electricity price. These results support our initial findings when using the max^n approach in Scenario I (see right-hand side of Figure 3), as explained before.

B. Saving Electricity Costs

From the previous scenarios it is clear the success of MCTS in these kind of problems. Particularly, the max^n MCTS approach was able to find the optimum solutions (i.e., an electric device finishing its task by being switched on at the lowest electricity cost) for Scenario I (no constraints) and it achieved excellent results for Scenario II (constraints).

However, it remains unclear from the previous analysis the percentage of electricity cost reduction achieved by both MCTS variants. Thus, to measure this, we simulated a similar scenario that a user faces, that is, turning devices on or off without the user knowing the price at a given time (let us call this a "random approach"). Thus, we performed a random action-evaluation process. That is, we selected an action (i.e., either turn on/off the electric device) at any given time within the boundaries set by Earliest Start Time (EST) and Latest Finish Time (LFT), and kept a record of the prices associated to those particular times when the electric devices was turned on. Then, we averaged this and used it for comparison purposes.

Firstly, let us focus our attention on the saving costs achieved by both the max^n (denoted by green-filled square symbols in Figure 5) and the two-agent MCTS approach (denoted by red-filled circles in Figure 5), for Scenario I (left-hand side of Figure 5). As indicated previously, the max^n MCTS approach was able to turn on all the devices at the lowest electricity price and this is reflected in the percentage of electricity cost saving, where a 50% saving is achieved. A good percentage of saving for this same scenario is also achieved when the two-agent MCTS approach is used, except for the clothes washer, denoted by CW, where there is no gain in using MCTS against the random approach.

Now, let us turn our attention on the saving electricity costs achieved by our MCTS, both max^n and two-agent approach, for Scenario II (results are shown in the right-hand side of Figure 5). There is a significant percentage of electricity cost savings when the former MCTS approach is used in this challenging scenario (see Section IV for details). In almost all cases, except for the electrical vehicle, the MCTS approach is able to achieve 50% saving. This situation changes, when the two-player MCTS approach is used, where there is only one case that reports an electricity cost saving that is when the electric vehicle is charged (around 8% electricity cost saving). For the CW, the two-agent MCTS approach performs very badly compared to the random approach, as indicated by its negative result. Notice that the electricity cost savings for both the CD and DW are not shown for the two-player MCTS approach (right-hand side of Figure 5) as a result of these devices not been able to finish their corresponding tasks as discussed previously (see left-hand side of Figure 4).

C. Final Comments on the Results

As shown before, the MCTS approach, in particular the max^n variant, is able to significantly reduce electricity costs in both scenarios presented in Section III. Whereas it is hard to compare our results with other agent-based methods due to multiple factors, such as the real computational effort required to run the system, we can still get a good idea of the potential of this approach on DSM systems. For example, in our previous work [20], we used a Distributed W-Learning (DWL), which is a reinforcement learning based multi-agent system, in a similar dynamic environment. DWL was able to perform slightly better than a random search, where the lowest reduction electricity cost achieved by DWL was less than 10% compared to 50% obtained by the max^n MCTS variant proposed in this paper. We believe that MCTS achives these extraordinary results due to its capacity of "predicting" what other agents (devices) might do (see Section II for the details on how the algorithm works).

VI. Conclusions

In this paper, we propose the use of Monte Carlo Tree Search with two variants: a max^n and two-agent approach, in Smart Grid technologies with the ultimate goal of learning the optimal times to switch electric devices on or off to minimise electricity costs, by *learning* and *predicting* the electricity price, based on a pricing history, in a dynamic price environment (e.g., prices rates changing over a period of time).

We also considered two different scenarios, a rather simple scenario where the goal, as indicated before, was to minimise electric costs while at the same time ensuring that each device finished its task (Scenario I). We then extended it, by considering a more challenging and realistic scenario, with constraints associated to it (Scenario II). The results achieved by MCTS are truly remarkable for this dynamic problem, in both scenarios. Particularly when using the max^n MCTS approach, where it was able to find, in almost all cases, the optimum solution.

Acknowledgments

This research was supported by Science Foundation Ireland (SFI) under the Principal Investigator research program 10/IN.1/I2980 "Self-organizing Architectures for Autonomic Management of Smart Cities" and by SFI grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] A. Conejo, J. Morales, and L. Baringo, "Real-time demand response model," *Smart Grid, IEEE Transactions on*, vol. 1, no. 3, pp. 236 –242, dec. 2010.

[2] C. Harris, I. Dusparic, E. Galván-López, A. Marinescu, V. Cahill, and S. Clarke, "Set Point Control for Charging of Electric Vehicles on the Distribution Network," in *IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*. Washington, D.C., USA: IEEE, Feb 2014.

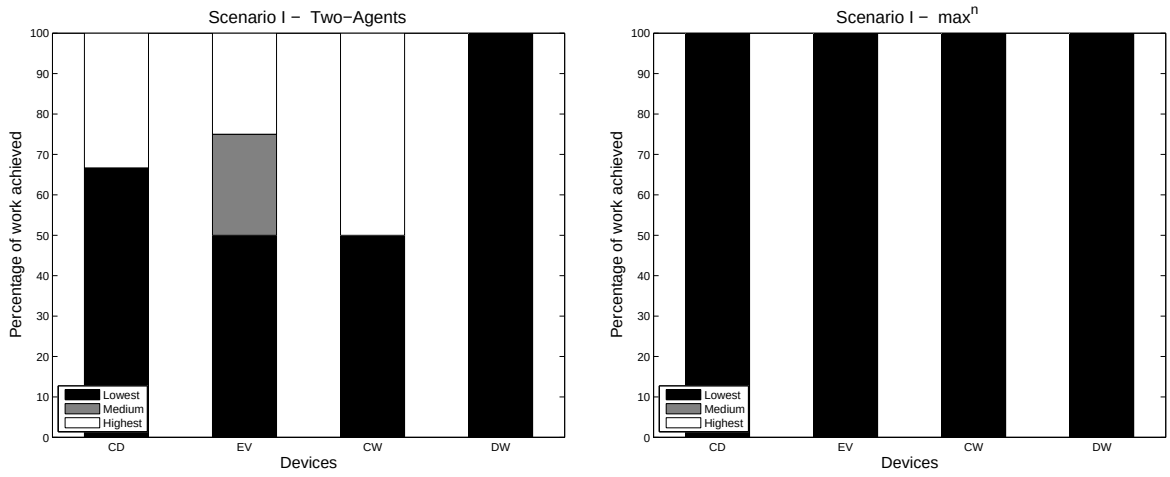

Fig. 3. Percentage of work achieved by our Monte Carlo Tree Search in Scenario I – No Constraints, using a two-agent approach (left-hand side) and a $\max^n$ approach (right-hand side), as explained in Section III.

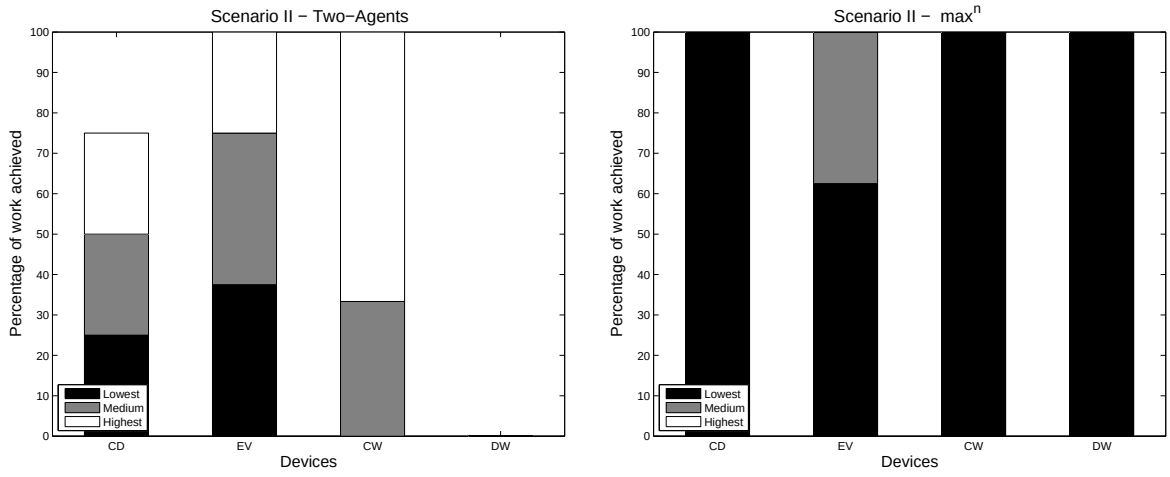

Fig. 4. Percentage of work achieved by our Monte Carlo Tree Search in Scenario II – Constraints, using a two-agent approach (left-hand side) and a $\max^n$ approach (right-hand side), as explained in Section III.

[3] E. Galván-López, A. Taylor, S. Clarke, and V. Cahill, "Design of an Automatic Demand-Side Management System Based on Evolutionary Algorithms," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. Gyeongju, Korea: ACM, March 2014.

[4] G. M. Masters, *Renewable and Efficient Electric Power Systems*. Wiley-Interscience, 2004.

[5] D. Miorandi and F. De Pellegrini, "Demand-side management in smart grids: An evolutionary games perspective," in *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on*, Oct., pp. 178–187.

[6] A. Mohsenian-Rad, V. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia, "Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid," *Smart Grid, IEEE Transactions on*, vol. 1, no. 3, pp. 320 –331, dec. 2010.

[7] P. Palensky and D. Dietrich, "Demand side management: Demand response, intelligent energy systems, and smart loads," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 3, pp. 381–388, Aug. 2011.

[8] A. Taylor, E. Galván-López, S. Clarke, and V. Cahill, "Accelerating Learning in Multi-Objective Systems through Transfer Learning." in *In a Special Session on Learning and Optimization in Multi-Criteria Dynamic and Uncertain Environments at the International Joint Conference on Neural Network 2014 (IEEE IJCNN)*. Beijing, China: IEEE, 2014.

[9] A. Taylor, I. Dusparic, E. Galván-López, S. Clarke, and V. Cahill, "Transfer Learning in Multi-Agent Systems Through Parallel Transfer," in *Workshop on Theoretically Grounded Transfer Learning at the 30th International Conference on Machine Learning (Poster)*, vol. 28, Atlanta, USA, 2013.

[10] T. K. Wijaya, T. G. Papioannou, X. Liu, and K. Aberer, "Effective Consumption Scheduling for Demand-Side Management in the Smart Grid using Non-Uniform Participation Rate," in *Sustainable Internet and ICT for Sustainability (SustainIT)*, 2013. [Online]. Available: https://github.com/tritritri/effective-dsm

[11] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, 2006, pp. 282–293.

[12] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with patterns in Monte-Carlo Go," INRIA, France, Tech. Rep. 6062, Nov. 2006. [Online]. Available: http://hal.inria.fr/docs/00/12/15/16/PDF/RR-6062.pdf

[13] D. Perez, P. Rohlfshagen, and S. M. Lucas, "The physical travelling salesman problem: Wcci 2012 competition," in *IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.

[14] T. Cazenave, "Nested monte-carlo expression discovery," in *Proc. of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. The Netherlands: IOS Press, 2010, pp. 1057–1058. [Online]. Available: http://dl.acm.org/citation.cfm?id=1860967.1861203

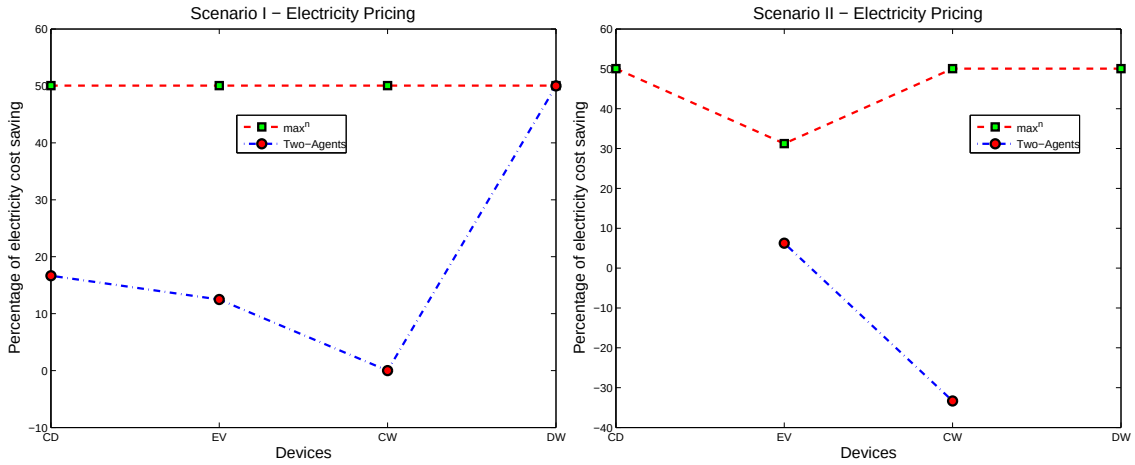

Fig. 5. Percentage of electricity cost savings achieved by our Monte Carlo Tree Search in two different scenarios, Scenario I – No Constraints (left-hand side) and Scenario II – Set of Constraints (right-hand side).

[15] S. Baba, Y. Joe, A. Iwasaki, and M. Yokoo, "Real-time solving of quantified csps based on monte-carlo game tree search," in *IJCAI*, 2011, pp. 655–661.

[16] S. Samothrakis, D. Robles, and S. Lucas, "Fast approximate max-n monte-carlo tree search for ms pac-man," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011.

[17] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1 –43, March 2012.

[18] D. Auger, "Multiple tree for partially observable monte-carlo tree search," in *Proc. of the 2011 international conference on Applications of evolutionary computation*. Berlin: Springer, 2011, pp. 53–62. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008402.2008410

[19] C. Luckhart and K. B. Irani, "An algorithmic solution of n-person games," in *AAAI*, T. Kehler, Ed. Morgan Kaufmann, 1986, pp. 158–162.

[20] E. Galvan, C. Harris, I. Dusparic, S. Clarke, and V. Cahill, "Reducing electricity costs in a dynamic pricing environment," in *Proc. Third IEEE International Conference on Smart Grid Communications (SmartGridComm)*. Tainan, Taiwan: IEEE Press, november 2012, pp. 169 – 174.