Bounded-Monitor Placement in Normative Environments

Guilherme Krzisch †, Nir Oren ‡, and Felipe Meneguzzi †

† Pontifical Catholic University of Rio Grande do Sul, Brazil
‡ University of Aberdeen, Scotland, UK
guilherme.krzisch@acad.pucrs.br, n.oren@abdn.ac.uk, felipe.meneguzzi@pucrs.br

Abstract. In order to sanction non-compliant agents, norm violations must be detected, which in turn requires norm monitoring. This paper examines the problem of monitor placement within a normative multi-agent system under budgetary constraints. More specifically we consider a system containing (1) a set of possible monitors able to determine the state of a subset of the domain; (2) costs associated with deploying the monitors; and (3) a set of norms for which compliance must be monitored, and which, if violated, result in a penalty. We seek to identify which combination of monitors maximizes the system's utility. We formalize the problem and evaluate approximate solutions using several heuristics, empirically demonstrating their efficiency.

1 Introduction

Since agents within open multi-agent systems cannot be assumed to share goals, obtaining desirable outcomes requires a coordination mechanism, and *norms* [5] are often used to perform this coordination. While regimented norms – which prevent an agent by design from undertaking undesirable behavior – are widely often assumed, a substantial body of work has shown the advantages of using approaches based on enforcement [8, 10]. These latter approaches, which allow norms to be violated, require a mechanism that monitors for norm compliance or violation and applies sanctions when appropriate [11]. In turn, *norm enforcement* can be performed either by some organization or by other autonomous agents in the system [14].

Norm enforcement assumes that violation and compliance can always be detected [6, 7, 11]. Such an assumption is, however, clearly unrealistic. First, in a large system, the cost of monitoring is very high [15]. Second, there are often portions of the environment that are not fully observable, and which monitors cannot access. There is therefore a need to investigate how norms should be monitored.

Previous work on the monitoring of norms has considered how norms should be modified so as to be monitorable [2], and whether related states can be observed which may indicate upcoming norm violations [1]. In this paper, we consider instead how monitors should be deployed within a system, assuming that such deployments have a cost, so as to monitor the most important norms. Our approach builds on ideas from the planning literature, and in Section 2, we introduce the necessary concepts from this domain and formalize the notion of a norm. Section 3 introduces monitors and formally

describes the problem we are addressing. We describe several approaches to addressing the problem in Section 4. Finally, Section 5 evaluates our solutions, and we conclude by considering related work (Section 6) and directions for future research (Section 7).

2 Background

In this section we introduce concepts from the planning literature, used to formalise the system and our solution. Following this, we describe norms within our system.

2.1 Planning

We build on classical planning, which assumes finite, fully observable and deterministic systems, and adapt the definitions from Ghallab *et al.* [9, Ch. 2]. Systems that follow classical planning semantics assume that actions in the domain cause transitions between states, and are specified in terms of sets of predicates.

Definition 1 (Predicate and State). *A predicate in a first-order language $\mathcal{L}$ is composed of a symbol and zero or more terms. Each term can be either a constant or a variable; a predicate is ground if it does not contain variables. We denote as $|\mathcal{L}|$ the number of ground predicates in this language. A state is a set of ground literals (i.e., positive or negative predicates) in $\mathcal{L}$.*

We use the operator $\models$ to specify that a state (a set of predicates) satisfies a logical formula, i.e., that a formula is a model of the state.

Classical planning makes a closed-world assumption — that if a state does not specify a predicate, then this predicate does not hold in that state. We write $s \models P$ where P is a set of ground predicates, if the conjunction of these ground predicates is satisfied by s. We formalize action execution, and its effects on the environment as follows.

Definition 2 (Planning Operator and Actions). *A planning operator is represented by a triple $o = \langle name(o), pre(o), eff(o) \rangle$, where $name(o)$ is the description of o. $pre(o)$ and $eff(o)$ are set of predicates representing the planning operator's preconditions and effects. An action a is a ground instance of a planning operator, and is* applicable *in state s only if $s \models pre(a)$. The result of applying action a to state s is a new state s', such that $s' = (s / eff^-(a)) \cup eff^+(a)$, where $eff = eff^+ \cup eff^-$ (such that $eff^+ \cap eff^- = \emptyset$) and eff^- is the set of negated predicates and eff^+ is the set of positive predicates.*

We specify the dynamics of our multiagent system in terms of a transition function following classical planning semantics in Definition 3, and the initial states of the system in terms of planning problem instances, as per the following definitions.

Definition 3 (Planning Domain). *If $\mathcal{L}$ is a first-order language with finite sets of predicates and constants, a planning domain in $\mathcal{L}$ is a state-transition system $\Sigma = \langle S, A, \gamma \rangle$, where $S \subseteq 2^{|\mathcal{L}|}$ is a subset of all possible states; A is the set of all ground instances of planning operators; $\gamma(s, a)$ is a state-transition function defined as follows: if $a \in A$ and a is applicable to $s \in S$, it returns the next state $s' \in S$, which is the result of applying action a to state s.*

Note that S is closed under γ, i.e., given a state $s \in S$, all states reachable from applying action a in s are also in S.

Definition 4 (Planning Problem). *A planning problem is defined as a triple $\mathcal{P} = \langle \Sigma, s_0, g \rangle$, where Σ is the state-transition system, $s_0 \in S$ is the initial state of the problem and g, the goal, is a set of ground predicates.*

2.2 Norms

In open and dynamic societies, self-interested agents cannot be assumed to share the same set of goals. In this context, norms can be used to regulate and coordinate behavior [13, Ch. 14]. For our work, we adapt the definition of a norm from [12]. We consider two norm types: obligations and prohibitions; the former specifies behavior that must be followed by agents, while the latter specifies behavior that must be avoided.

Definition 5 (Norm). *A norm is a tuple $n = \langle \mu, \chi, \rho, C \rangle$, where:*

- *$\mu \in \{obligation, prohibition\}$ represents the norm's modality;*
- *χ is a set of ground predicates that represents the* context *to which a norm applies, i.e., a norm is applicable in state s if $s \models \chi$;*
- *$\rho \in A$ represents the* object *of the norm's modality;*
- *C is the cost or penalty* to the society *which occurs if the norm is violated.*

Example 1. The following norm requires an agent to drive on the left side of the road if they are in England; a violation causes harm to the society worth 20 units of utility.

$$n_0 = \langle obligation, at(England), driveLeft(a, b), 20 \rangle$$

We now describe when a norm is considered to be violated by an agent.

Definition 6 (Norm Violation). *A norm $n = \langle \mu, \chi, \rho, C \rangle$ is violated in state s by an agent a iff:*

- *$s \models \chi$; and*
- *agent a either: executes action ρ in state s and $\mu = prohibition$; or does not execute action ρ in state s and $\mu = obligation$.*

A violated norm has an undesirable impact on the society, as encoded by its cost C. To dissuade agents from violating norms, when such a violation is detected, an enforcer applies a penalty to the agent. In this work, we do not consider the nature of this penalty, assuming instead that it is sufficiently large to prevent the agent from violating a norm *if such a violation can be detected*. We must therefore consider how to place monitors so that violations are detected, while minimizing monitor costs. We refer to this as the *bounded-monitor placement problem*, and describe it in more detail in the next section.

3 Bounded-Monitor Placement Problem

We consider a set of monitors able to determine whether some combination of predicates is, or is not satisfied. Formally, we define a monitor as follows.

Definition 7 (Monitor). *A monitor $m = \langle P, D \rangle$ consists of a set of predicates P, and a* deployment cost *$D \in \mathbb{R}$. We refer to the predicates of a monitor m as P_m, and to its cost as D_m.*

A set of monitors can be used to monitor more complex combinations of predicates. Some monitors can conceivably detect the status of a predicate related to multiple norms, or when combined, can be used to determine the status of a norm that individual monitors cannot. We formalize the combination of monitors aiming to cover sets of norms as a *Monitor Placement*.

Definition 8 (Monitor Placement). *A monitor placement MP is a tuple $\langle M_1, M_2 \rangle$, where M_1 and M_2 are sets of monitors, representing the capability of observing predicates in the current and in the next state, assuming an action was performed.*

A monitor placement detects *a norm violation $\langle \mu, \chi, \rho, C \rangle$ iff given a state s and s' such that s' is the result of applying an action ρ at s; and $s \models \chi$, one of the following holds:*

1. *$\mu = prohibition$ and $s \models \bigwedge\{P_{m_1} | m_1 \in M_1\}$ and $s' \models \bigwedge\{P_{m_2} | m_2 \in M_2\}$.*
2. *$\mu = obligation$ and $s \models \bigwedge\{P_{m_1} | m_1 \in M_1\}$ and $s' \not\models \bigwedge\{P_{m_2} | m_2 \in M_2\}$.*

*The cost **C** of a monitor placement is $\sum_{m \in M_1 \vee m \in M_2} D_m$, while the utility **U** is $\sum_{n \in N} C$, where N is the set of norms detected by this placement.*

By introducing the concept of an available budget, we define our problem of placing monitors in a system as follows.

Definition 9 (Bounded-Monitor Placement Problem). *A bounded-monitor placement problem is encoded as a triple $\langle M, N, B \rangle$ where M is a set of monitors, N is a set of norms, and $B \in \mathbb{R}^+$ is a budget.*

A solution to the problem is a monitor placement MP such that its cost is smaller than, or equal to, the budget.

The set of possible solutions for a monitor placement problem is exponential in the worst case, and in the next section, we suggest several approaches to finding solutions.

4 Solution

4.1 Brute-force

A brute-force approach is a trivial solution to this problem. It considers all possible combinations of available monitors and returns the best one. We can clearly see that this is impractical for large problems, as its complexity increases exponentially given the size of the possible monitors set input: specifically, it has a time complexity of $O(2^{2|M|})$. We use this approach as a baseline against which we compare the remaining heuristics.

4.2 Mapping from norms to monitors

The main drawback of the brute-force approach is that it includes a large number of irrelevant solutions while enumerating all possible solutions. We can reduce this overhead by computing a mapping from norms to monitor placements, i.e., by finding the set of all monitor placements capable of monitoring each norm.

With this mapping, we can compute possible solutions by choosing one monitor placement for each norm. Generalizing, we have $\prod_{i=1}^{|N|}(|\mathbf{MP}_{n_i}|+1)$ possible solutions, where $\mathbf{MP}_{n_i}$ is the set of monitor placements able to monitor norm n_i; since we also need to consider whether it is practical to monitor a given norm (e.g., when there is no available budget to do so), we need to add the empty monitor placement set. In the best case we have only one possible monitor placement for each norm, and in the worst case we have all possible combinations of available monitors for MP_{M_1} and MP_{M_2}, for each norm. Therefore, the number of solutions ranges from $2^{|N|}$ to $4^{|M||N|}$.

Given this exponential complexity, it is clear that both the brute force and monitor mapping approaches cannot scale up to larger problem sets. We must therefore consider heuristics for addressing the problem, which we describe next.

4.3 Naive Approximate Solution

To improve performance compared to the brute-force approach we introduce a simple approximate solution whose purpose is to serve as a a baseline for comparing the accuracy of the other solutions. This solution iterates over monitors ranked by their expected probability of detecting norm violations. To rank monitors, we consider the number of norms that a single monitor can partially detect — a monitor can partially detect a norm if it has at least one predicate of the norm's context or of the preconditions of the norm's action ρ. The intuition here is that choosing monitors that can partially observe several norms leads to a final monitor placement that can detect many existing norms.

This approach, however, does not capture essential parts of the problem. First, it does not consider the norm's penalty and monitor's cost. Second, it has an overly strong assumption that joining monitors that can partially detect norms will lead to a monitor placement that can actually detect norms. We can therefore enhance this approach by using the mapping describe in Section 4.2 together with a greedy search.

4.4 Greedy Solution

We propose two approaches with different heuristics using the mapping structure introduced in Section 4.2. By using a heuristic to rank the best monitor placements, we can avoid searching through an exponential solution search space. More specifically, we select the best monitor placement candidate for each norm. The resulting heuristic sacrifices optimality for efficiency, running in linear time.

Our base algorithm (used for both of our heuristics) is described in Algorithm 1. It starts by creating a mapping between norms and monitor placements, as described in Section 4.2. After this, it adds monitors to an initially empty monitor placement $currentMP$, while budget is available. Within each iteration, it selects one norm, gets a monitor placement able to monitor this norm, and adds the already selected monitors

Algorithm 1 Greedy algorithm

```
1: procedure FINDAPPROXIMATESOLUTION(N:Norms)
2:     build mapping from norms to monitor placements
3:     currentMP ← {}
4:     while hasBudget do
5:         n ← extractMaxNorm(N)
6:         mp ← getMinMP(n)
7:         currentMP ← currentMP ∪ mp
       return currentMP
```

($currentMP$) to the monitor placement. We now describe two heuristics to speed up this algorithm.

4.4.1 Norm Independence Heuristic Our first heuristic considers norms to be monitored completely independently of each other when choosing monitors in order to substantially prune the search space of the problem. We first need to define which norm *extractMaxNorm(N)* in line 5 of Algorithm 1 chooses. Here, we select the norm with the highest penalty, in order to increase the value of **U** (the sum of each individual norm penalty), i.e., $extractMaxNorm(N) = \arg\max_{n \in N}\ C_n$.

The other decision required is which monitor placement is selected by the *getMinMP(n)* method (line 6). For this, we select the placement with the lowest cost, as it is a good monitor placement able to monitor norm n. Thus, $getMinMP(n) = \arg\min_{MP \in \mathbf{MP}_n}\ \mathbf{C}_{MP}$.

This approach does not consider the intersection of monitors that are able to monitor multiple norms; it selects monitor placements independently of the others. Consequently, we improve this solution in what follows by introducing the concept of *current cost* to try to find a better approximate solution for our problem.

4.4.2 Add and Update Heuristic The structure of this heuristic is similar to the previous approach. However, instead of using the cost of each monitor placement as the metric to select the one with lowest budget, we use its *current cost*. The current cost of a given monitor placement MP is computed as per Formula 1 below, and considers only the cost of monitors that were not already selected in a previous iteration (and thus not in the set of monitors of *currentMP*).

$$\mathbf{CC}_{MP} = \sum_{m \in M_{MP} \wedge m \notin M_{currentMP}} D_m \tag{1}$$

$$getMinMP(n) = \arg\min_{MP \in \mathbf{MP}_n}\ \mathbf{CC}_{MP} \tag{2}$$

$$extractMaxNorm(N) = \arg\max_{n \in N}\ \frac{C_n}{\mathbf{CC}_{getMinMP(n)}} \tag{3}$$

The *getMinMP(n)* method is implemented as per Formula 2, which we use to compute the next norm to be monitored using the *extractMaxNorm(N)* method, implemented in Formula 3. This heuristic chooses the norm with the highest value based on the ratio

between its penalty and the lowest current cost of the set of monitor placements. By using the current cost, we disregard the costs of monitors that have already been chosen in past iterations, yielding better estimates. In the next section we empirically evaluate the different approaches proposed in this section.

5 Experiments and Results

To empirically evaluate our approaches, we automatically generate sets of norms and sets of monitors with increasing complexity. In our experiments, we assumed the set of monitors M_2 of a monitor placement can in effect see all actions that were executed from the states visible by M_1; i.e., we assume that our monitor placement is always able to check the next state after applying a given action. While this is a strong assumption and makes our problem easier to solve, it still captures the exponential nature of the bounded-monitor placement problem. Our experiments are based on standard planning problems [], and our main domain is the *drink-driving* domain, where agents are able to drive between cities, and there is a norm stating that it is forbidden to drive while drunk; we also tested with *blocksworld, depots, dwr, easy_ipc_grid, gripper, logistics* and *robby* domains.

There are two metrics to consider when analyzing results: time performance and accuracy. When comparing time performance we use the brute-force approach as a baseline for the approximate approaches. In Figure 1 we can see that the brute-force approach becomes intractable for a relative small number of norms, while the approximate approaches remain linear as the number of norms increases.

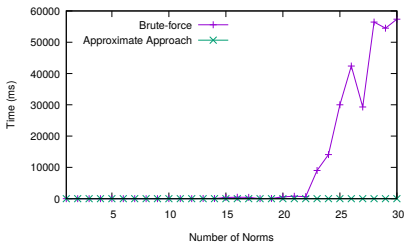

Fig. 1. Time Efficiency of Brute-Force and Approximate approaches, with timeout of 60 seconds

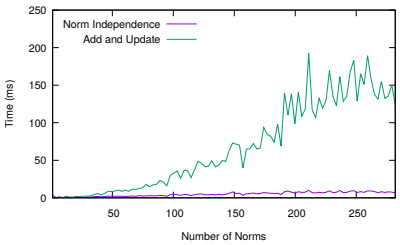

Fig. 2. Time Efficiency of both greedy approaches, smoothed using a sample of 100 data points and interpolated using splines

Figure 2 shows the time performance of both greedy solutions. The *Add and Update Solution* is worse in this metric than the *Norm Independence Solution* because it needs to recompute the current cost at each iteration. It is still fast compared to the brute-force approach, being able to find a solution for a problem with almost 300 norms in under one second.

We perform two experiments to compare the accuracy of the results of the approximate approaches. First, in Figure 4, we show the relative accuracy compared with an optimal solution to the problem using the brute-force approach. Note that, as we are

comparing with the brute-force approach, these results are limited to small problems that this approach can solve[1]. In this experiment, both greedy approaches outperform the naive solution; between the two greedy approaches, the second one (*Add and Update Heuristic*) has, in almost all domains, greater accuracy, and — in all cases — a smaller standard deviation. For the *depots*, *gripper* and *logistics* domain, the second solution achieves optimal accuracy; this can be explained as these domains become intractable even for small number of norms, and thus the number of problems in this experiment, for these domains, is also small. The increase in performance from the first to the second greedy approach is relatively small for these domains; while it is relatively large for the *drinkdriving* and *robby* domains.

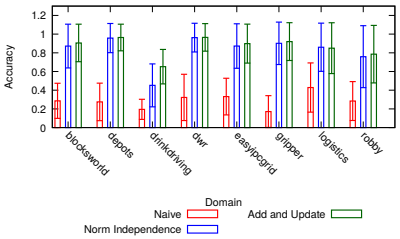

Fig. 3. Accuracy of approximate approaches, compared to the maximum U possible; error bars represent one standard deviation of uncertainty

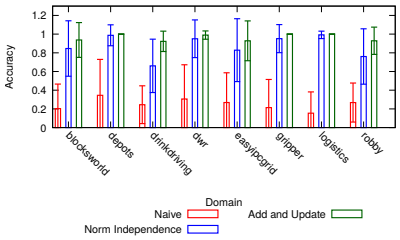

Fig. 4. Accuracy of approximate approaches, compared to a brute-force solution; error bars represent one standard deviation of uncertainty

To investigate if the relationships found for the first experiment hold for large problems, we perform additional experiments, shown in Figure 3. As these problems cannot be solved in a timely manner using a brute-force approach, in order to calculate their accuracy we compare them with a perfect solution that could monitor all norms, but that does not necessarily need to respect the available budget. This perfect solution has the maximum value of **U**, which can be unattainable for actual solutions to these problems; therefore, in this experiment we are interested in the relative accuracy between the approximate approaches, and not in how close they are to the perfect solution.

We can see the same pattern in this experiment; the greedy approaches perform better than the naive solution, with a slight advantage for the *Add and Update Heuristic*. The increase in performance from the first to the second greedy approach remains large for the *drinkdriving* domain, while for other domains it is relatively small.

From the experiments we conclude that brute-force solution is intractable for all but small problems, while approximate approaches can solve large problems. The accuracy of the greedy approaches is better than the naive solution, with a slight advantage to the second greedy approach. This advantage is more noticeable for small problems; for large problems — as we do not have the value for the optimal solution — this difference is smaller.

[1] We set a timeout of 30 seconds for this experiment.

6 Related Work

Other authors also dropped the assumption that a monitor has full observability in the system. Criado's [4] approach is similar to our work; their norms are more expressive, and for their solution they use the CEF algorithm which uses a greedy approach. However, they do not perform an empirical evaluation, or consider whether their approach is able to actually monitor any norm, as there are no guarantees proven for their proposed algorithm. Alechina *et al.* [2] models a set of queries that a monitor can ask in a state, i.e., a monitor may not be able to distinguish between two states. They then modify the set of norms to a new set of approximate norms that can be optimally monitored given a set of monitors and queries, therefore approaching the problem from another perspective.

Alechina *et al.* [1] define norms using LTL (linear temporal logic) formulas. They introduce the concept of a guard, which uses lookahead mechanisms to detect future norm violations. The size of the lookahead window is bounded to reduce the amount of computation in the future (they have complete knowledge of the past), and have similarities with the concept of monitor cost in our work. The main difference is that, while we use a combination of monitors to be able to detect a norm violation, they increase their lookahead window size to increase their monitoring capabilities, also increasing its computational cost.

Finally, our work is also similar to that of Bulling *et al.* [3]; both include the concept of monitors and combination of monitors. While we use a relatively simple norm formalism and optimize the cost to monitor these norms, their specification uses LTL-formulas, focusing on its properties and relations. Their current framework does not include norms or monitors costs.

7 Conclusion and Future Work

In this paper we extend the state of the art in norm monitoring by dropping the assumption that a monitor system has full observability, i.e., that monitors can observe all actions performed. Adding the notion of a set of available monitors and an associated cost results in the problem of finding a monitor placement in order to maximize the number of detectable violations. While brute-force solutions are impractical because the possible solution search space is exponential in the size of input, we propose heuristics that use a mapping between norms and monitors to find approximate solutions. Our empirical of runtime performance and accuracy shows that these heuristics are both practical in computational terms and approach optimal performance for many realistic domains from the planning literature.

We aim to extend the work in at least three ways. The first is to allow monitors to dynamically modify their placement during execution. In this setting, monitors would be able to observe agent actions and move to locations where more norm violations occur. The second extension would be to consider different agents being able to perform concurrent actions, and how to build monitors able to correctly identify which agent violated a given norm. The third is to allow more complex expressions representing both what monitors can observe and how monitors can be combined, as currently we

only consider conjunctions of monitors. This can increase the richness of our approach, and we intend to investigate heuristics for the use of such more expressive monitors.

Acknowledgments

This work is partially supported by grants from CNPq/Brazil numbers 132339/2016-1 and 305969/2016-1.

References

1. Alechina, N., Bulling, N., Dastani, M., Logan, B.: Practical run-time norm enforcement with bounded lookahead. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 443–451 (2015)
2. Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: Proceedings of the 2014 International Conference on Autonomous Agents and Multiagent Systems. pp. 117–124 (2014)
3. Bulling, N., Dastani, M., Knobbout, M.: Monitoring norm violations in multi-agent systems. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems. pp. 491–498 (2013)
4. Criado, N.: A practical resource-constrained norm monitor. In: Proceedings of the 2017 International Conference on Autonomous Agents and Multiagent Systems. pp. 1508–1510 (2017)
5. Dignum, F.: Autonomous agents with norms. Artificial Intelligence and Law 7(1), 69–79 (1999)
6. Esteva, M., Rosell, B., Rodriguez-Aguilar, J.A., Arcos, J.L.: Ameli: An agent-based middleware for electronic institutions. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 236–243 (2004)
7. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: Implementing norms in electronic institutions. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 667–673 (2005)
8. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraint rule-based programming of norms for electronic institutions. In: Proceedings of the 2009 International Conference on Autonomous Agents and Multiagent Systems. pp. 186–217 (2009)
9. Ghallab, M., Nau, D., Traverso, P.: Automated planning: theory & practice. Elsevier (2004)
10. Luck, M., d'Inverno, M.: Constraining autonomy through norms. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 674–681 (2002)
11. Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., Luck, M.: A framework for monitoring agent-based normative systems. In: Proceedings of the 2009 International Conference on Autonomous Agents and Multiagent Systems. pp. 153–160 (2009)
12. Oren, N., Panagiotidi, S., Vázquez-Salceda, J., Modgil, S., Luck, M., Miles, S.: Towards a formalisation of electronic contracting environments. In: Coordination, Organizations, Institutions and Norms in Agent Systems IV, pp. 156–171 (2009)
13. Ossowski, S.: Agreement technologies, vol. 8. Springer Science & Business Media (2012)
14. Savarimuthu, B.T.R., Cranefield, S.: Norm creation, spreading and emergence: A survey of simulation models of norms in multi-agent systems. Multiagent and Grid Systems 7(1), 21–54 (2011)
15. Sutinen, J.G., Andersen, P.: The economics of fisheries law enforcement. Land economics 61(4), 387–397 (1985)