Automated Synthesis of Social Laws in STRIPS

Ronen Nir, Alexander Shleyfman, Erez Karpas
Technion — Israel Institute of Technology
Haifa, Israel
{ronenn, karpase}@technion.ac.il, shleyfman.alexander@gmail.com

Abstract

Agents operating in a multi-agent environment must consider not just their actions, but also those of the other agents in the system. Artificial social systems are a well-known means for coordinating a set of agents, without requiring centralized planning or online negotiation between agents. Artificial social systems enact a social law which restricts the agents from performing some actions under some circumstances. A robust social law prevents the agents from interfering with each other, but does not prevent them from achieving their goals. Previous work has addressed how to check if a given social law, formulated in a variant of MA-STRIPS, is robust, via compilation to planning. However, the social law was manually specified. In this paper, we address the problem of automatically synthesizing a robust social law for a given multi-agent environment. We treat the problem of social law synthesis as a search through the space of possible social laws, relying on the robustness verification procedure as a goal test. We also show how to exploit additional information produced by the robustness verification procedure to guide the search.

Introduction

When operating in a multi-agent environment one must take into consideration not only their own goals and actions, but also the possible actions of the other agents in the system, and their possible effects. Often in multi-agent systems, a plan that would have allowed an agent to reach its goals had it operated in isolation, leads to some sort of failure due to other agents' actions. For example, an autonomous car that drives from location A to location B without taking into consideration that there are other cars on the road will likely end up crashing into another car.

The literature offers varying approaches for coordination of multiple agents. One can, for example, design rules of encounter that will determine the behavior and the negotiation protocol for these agents, if their activities are bound to interfere. Unfortunately, these rules might be inefficient and lead to repeated negotiations between the agents for conflict resolution. One other drawback of this approach is purely applicational – it requires the physical embodiment of the agents to have means of communication and time to do it, which is not always feasible. To avoid this, we can, for instance, institute central control over the agents. This approach, being useful in numerous domains, might, however, suffer from limitations, such as bottlenecks at the central site or system-wide vulnerability to failure.

One way to avoid centralized planning or online negotiation between agents, allowing for each agent to plan independently, is an artificial social system that enacts a social law which restricts the agents from performing some actions in certain circumstances. Social laws aim to prevent agents from disrupting each other, while still allowing them to achieve their personal goals. Intuitively, a social law is an offline restriction on the actions legally available to the agents, which should remove the online conflicts and the need to negotiate. In the car example we mentioned earlier, the social law enacted for the benefit of society is the traffic code, which allows agents efficient travel while avoiding substantial damages.

The artificial social systems approach became a canonical approach to achieve coordination between agents (Wooldridge 2001; Shoham and Leyton-Brown 2009; Horling and Lesser 2004; d'Inverno and Luck 2004; Klusch 1999). In particular, one of the early works proposed a STRIPS-like representation of multi-agent environments to establish the connection between social systems and modern planning techniques (Tennenholtz and Moses 1989). Recently, Karpas, Shleyfman, and Tennenholtz (2017) followed in this direction, proposing a formalism to represent and reason over social laws using classical planning. This work proposed an algorithm for the verification of the robustness of a given social law. This has also been extended to robustness verification in the temporal planning setting (Nir and Karpas 2019). However, these assume a social law is given.

In this paper, we describe an approach for automatic synthesis of robust social laws. We treat the problem of social law synthesis as a search problem, with the above-mentioned robustness verification procedure as a goal test. We formulate a search space whose transitions consists of making small changes in the social law, and show that a naive search approach does not scale well. We also provide some techniques for guiding the search, which are based on exploiting information from the robustness verification procedure.

Although previous work (Morales et al. 2018) has addressed social law synthesis in the context of normative systems (using a different representation of the multi-agent setting). However, they used evolutionary algorithms, which cannot guarantee completeness, while our algorithm is complete – it is guaranteed to find a robust social law if one exists. Finally, we empirically show that our approach is able to synthesize robust social laws (or proves they do not exists) on multiple problems from multi-agent planning benchmarks.

Preliminaries

Since the multi-agent planning setting considered in this work does not assume coordination between agents, we use the modification of MA-STRIPS (Brafman and Domshlak 2008) presented by Karpas *et al.* (2017) that includes a goal for each agent, rather than an overall goal for the whole problem.

Formally, a problem in the multi-agent planning setting in question is defined by a tuple $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$, where: $n \in \mathbb{N}$ represents the number of agents, F is a final set of facts, $I \subseteq F$ is the initial state, the $G_i \subseteq F$ defines the goal of agent i, and A_i is the set of actions of the correspondent agent.

Each subset $s \subseteq F$ is called a state, and $S = 2^F$ is the state space of Π. Each action $a \in A_i$ is described by preconditions $\mathsf{pre}(a) \subseteq F$, add effects $\mathsf{add}(a) \subseteq F$, and delete effects $\mathsf{del}(a) \subseteq F$. The action a is applicable in the state s if $\mathsf{pre}(a) \subseteq s$, and the result of this application will be denoted by $s[\![a]\!] := (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$. In what follows we assume unit-cost actions.

The projection of Π for agent i is the (single agent) STRIPS (Fikes and Nilsson 1971) planning problem $\Pi_i = \langle F, A_i, I, G_i \rangle$. We say that a sequence of actions $\pi_i = \langle a_1, \ldots, a_k \rangle$ is a plan for the task Π_i if, for each $1 \le j \le k$, $a_j \in A_i$ is applicable in state $I[\![a_1]\!] \ldots [\![a_{j-1}]\!]$, and $G_i \subseteq I[\![a_1]\!] \ldots [\![a_k]\!]$. In what follows, when it is appropriate, we may refer to plans as sets. An action set $L \in A_i$ is called disjoint action landmark (or landmark for short) if for each plan π_i for Π_i it holds $L \cap \pi_i \neq \emptyset$. Following, Fišer et al. (2019), we denote $\Pi_i \setminus L := \langle F, A_i \setminus L, I, G_i \rangle$.

In the execution model we consider here, each agent plans offline, and a scheduler executes the action one at a time by choosing which agents acts next. Thus, at any given time there are multiple agents ready to perform their next action (the number could be less than n if some agents are already at the end of their plan execution). We do not assume any particular order on the execution of actions by agents, i.e. this can be seen as if the agents act according to some external (unknown) scheduler. The agent chosen by the scheduler executes its pending action, causing the state of the world to change accordingly, then setting the next action in its plan as pending. Then the scheduler assigns the next agent repeating the process until all plans are finished. Note that we do not make any assumptions on the scheduler, in fact, considering it to be adversarial. This execution model may, and often will, result in *conflicts*[1] in execution of agents plans. Consider, for example, two agents with two pending actions a_1 and a_2, correspondingly, s.t. $\mathsf{pre}(a_1) \cap \mathsf{del}(a_2) \neq \emptyset$. The scheduler setting – (execute a_2, execute a_1) results in a conflict.

[1]We define a conflict to be a failure in an execution of the scheduling of the joint plans of all agents.

In previous work Karpas *et al.*, defined a *social law* l as a modification of the multi-agent setting Π, resulting in a setting Π^l. A social law is described by the modifications it applies to Π: It can add, delete or modify facts, actions, the initial state, or the goal. As Karpas *et al.* showed in their work, some restrictions require complex conditions, for which the original set of facts F is insufficient. Others are based on a wait-for mechanism. These restrictions require an agent to wait for a fact f to hold before it can perform an action a. Thus, in the case of MA-STRIPS, social law l can be described by the following:

(a) the modifications it applies to the agents' actions;

(b) the modifications it applies to the facts, the initial state or the goals;

(c) its annotations of certain $f \in \mathsf{pre}(a)$ as a wait-for precondition.

One of the plausible assumptions one may require is that these modifications do not make the problem meaningless (e.g., by changing the goal to an empty set).

In this work, we concentrate only on social laws that alter the actions of the agent without changing the initial set of facts. More specifically, similarly to the work on goal recognition design (Keren et al. (2014)) which disallowed specific actions, we will concentrate only on social laws that restrict actions that may lead to a conflict in the execution of a scheduling, while still trying to preserve the reachability of all agent goals.

Robustness Verification

In their work Karpas *et al.* formalize a *rational robustness* as the assumption that all agents are rational and want to achieve their goal, while asking the question whether there is *any* possible way for them to interfere with each other, assuming in a sense the adage formulated by Murphy's law – "Anything that can go wrong will go wrong". This assumption results in the following definition:

Definition 1. *A social law l for multi-agent setting $\Pi = \langle F, \{A_i\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ is robust to rational iff for all agents $i = 1 \ldots n$, for all individual solutions π_i for Π_i, for all possible action sequences π resulting from any arbitrary interleaving of $\{\pi_i\}_{i=1}^n$ which respects* wait-for *preconditions, π achieves $G_1 \cup \ldots \cup G_n$.*

Specifically, in this work we define a *social law* l as the set of restricted actions $A^l \subseteq A$. Such a law is *robust* if it allows *any* scheduling of the agents' plans: preventing any possible conflicts and ensuring that each agent will reach its goal at the end of the execution. Formally, we would like that any schedule of the individual agents' plans for the modified planning problem $\Pi^l = \langle F, \{A_i \setminus A^l\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$ will yield no conflict.

The verification algorithm IS_ROBUST[2] proposed by

[2]In the original work the algorithm was named VERIFY-RATIONAL.

Karpas *et al.* for a MA-STRIPS problem for Π^l can be seen as two independent procedures that both constitute classical planning problems:

1. IS_SOLVABLE_FORALL(Π^l) – checks whether the individual goal is reachable for each agent i, i. e., is the problem $\langle F, A_i, I, G_i\rangle$ solvable for each i.
2. FIND_CONFLICT(Π^l) – ensures that there are no conflicts in any possible schedule. Intuitively, it creates a planning problem Π' such that the solution of which is an execution of a joint plan that leads to a conflict. Thus, the social law is robust iff the planning problem Π' is *unsolvable*. Otherwise, its solution, the plan π^f produced by the will be deemed as a counter-example. Due to space limitations for the full compilation we refer the reader to Karpas *et al.*

Note that the solution to Π' provides a counter-example for the robustness of the social law.

In this setting, a robust social law l, for Π, can be seen as a special case of an action landmark A^l for the compilation Π' mentioned above, since the removal of A^l from Π' renders it unsolvable. Note, however, that not every action landmark will do for this purpose since we still require the planning tasks $\Pi_i \setminus A^l$ to be solvable, i. e. each agent should be still able to reach its goal. It is also important to note that we use a slight abuse of notation here, since the actions of Π^l may have additional auxiliary atoms both in preconditions and effects, thus when writing $A_i \setminus A^l$ we mean it label-wise.

Searching for a Social Law

In this section, we introduce a method that deals with the problem of finding a robust social law for a given multi-agent system, i.e. find a set of restrictions on the agents' behavior that will guarantee their successful coexistence.

One can think of numerous approaches to find a robust social law for a given domain. One approach is to call a domain-expert to manually formulate a set of restrictions on the agents. The union of these restrictions can be considered as a social law. This approach has its drawbacks, its products are domain-dependent, a domain-expert may not be always available and he may not be able to locate all of the possible conflicts.

In this section, we introduce a method that, for a given MA-STRIPS problem Π, returns a robust social law, if there is one. As mentioned before, the social law we consider is based on a restriction of a finite set of grounded actions. The result of enacting a social law l on a MA-STRIPS problem Π is denoted by Π^l – a similar MA-STRIPS problem, only without the grounded actions included in the social law. Our approach is based on a forward search through the space of all social laws until a robust social law is found. Firstly, we formulate the problem of finding a robust social law for a given MA-STRIPS problem as a search problem with the following components:

- *The initial state s_0:* we consider the original MA-STRIPS problem as the initial state, assuming it contains an empty social law, that is there are no restrictions on the agents' behavior, which means that the for the initial social law l_0 holds $A^{l_0} = \emptyset$. Note that the states of our search problem differ only in the set of restricted actions, therefore, when it is appropriate, we will refer to states as sets of restricted actions.
- *The successor generating function $succ(s)$:* this procedure provides us with all the direct successors of the state s. We define the successors to be an addition of a single-action restriction to the social law of s. This restriction makes one grounded action unavailable in the resulted state, i. e., given a state s that corresponds to a social law l, the set of the restricted actions of the successors of s will be $A^l \cup \{a\}$ for each $a \in A \setminus A^l$. Note that the edges of the search graph corresponds label-wise to the actions of the original problem, moreover, since each state correspond to a set of forbidden actions each action can be restricted at most once, and the restriction process is commutative.
- *The goal test function $is\_goal(s)$:* this function is used in order to identify whether a certain state is a goal state or not. Here we rely on IS_ROBUST, the social law robustness verification method proposed by Karpas *et al.*, and use it as a goal test function.

Theoretically, a search through the space of social laws should tackle this problem. However, the size of this space is $2^{|A|}$ where A is the set of all agents' actions so the naive search approach does not scale well. Next, we establish techniques to guide the search by pruning, heuristics and preferred operators identification.

Pruning Social Laws

With a huge search space, the first thing we tackle is reducing the size of the search tree branching factor. This can be done efficiently with pruning. However, pruning could also remove possible solutions. We can accept pruning only if it is *safe* (Ghallab, Nau, and Traverso 2004), e. g., it does not remove all of the solutions.

Definition 2. *Let $succ'$ be a pruning functions s.t. $succ'(s) \subseteq succ(s)$ for each search state s. We say that $succ'$ is safe if for every state s at least one reachable solution preserved in the pruned state space induced by $succ'$. We say that $succ'$ is completely safe if all solutions that are reachable in the original state space induced by $succ$ are preserved in the state space induced by $succ'$.*

For pruning, we use the results of our goal test function IS_ROBUST(Π) that is based on the work of Karpas *et al.*.

Prune social laws by counter example When testing the robustness of a social law in Π the first part of our goal test function, that is FIND_CONFLICT(Π) may return a counter example π^f, which is a joint-plan that leads to a conflict. One can show that every robust social law for Π has to contain at least one action from π^f.

Lemma 1. *Let Π be a non-robust* MA-STRIPS *problem, π^f a joint plan that fails for Π, and l a social law, such that $\Pi^l := \Pi \setminus A^l$ is robust. Then, there exist some action $a \in \pi^f$ such that $a \subseteq A^l$.*

Proof. Let Π be a MA-STRIPS non-robust problem. Since Π is not robust, there is a joint plan FIND_CONFLICT$(\Pi) = \pi^f$ that leads to a failure. Now, assume in contradiction that there is a robust social law l for Π s.t. $A^l \cap \pi^f = \emptyset$. This, means that π^f is executable in Π^l, which leads to contradiction, since l is robust by assumption. □

Concerning the search for a robust social law, this Lemma provides us with an important intuition, that corresponds, in a sense, to the strong stubborn sets introduced by Wehrle and Helmert (2014). Note, that since each search space state has a set of restricted actions, each path that leads to this state in the search tree is fully commutative, i. e., there is no importance to the order in which the actions are restricted. This together with a previous lemma can be summed up as follows:

Corollary 1. *Let s be a search space state that corresponds to a non-robust social law l, and let π^f be the joint plan that leads to failure. Then, the successor generating function $succ_{\text{CE}}(s) := succ(s) \cap \pi^f$ is safe.*

Proof. Let s be a search state that corresponds to the social law l with a failure plan π^f. Let us also assume that there is a goal state s_* that corresponds to a robust social law l^* and is reachable from s. By Lemma 1, there exist an action $a \in \pi^f$ s.t. $A^l \cup \{a\} \subseteq A^{l^*}$, thus there is $s' \in succ_{\text{CE}}(s)$ that lies on the path from s to s_*. This argument can be applied inductively, since, by construction of the search state space, all transitions are commutative, and each action can be restricted at most once. □

With the above conclusion we can establish a powerful tool for social laws pruning towards a more effective search of robust social laws. For example, in a MA-STRIPS that has 800 grounded actions the branching factor starts with 800. Assume that our social law robustness verification function returns a plan π^f and $|\pi^f| = 30$. The result of pruning out the actions that were not included in π^f is a significant reduction of the search tree branching factor.

Intuitively, each robust social law has to include at least one action from a given counter example. If not, a conflict that lies in the counter example remains possible, making the social law not robust. Thus, pruning by counter example is safe. A similar concept of effective pruning of search space is called strong stubborn sets (Valmari 1989; Wehrle and Helmert 2014), which was also used to prune the search in Goal Recognition Design (Keren, Gal, and Karpas 2018) in a similar fashion to here.

Pruning infeasible social laws First, we note that IS_SOLVABLE_FORALL is a Boolean function that checks whether all of the agents can achieve their goal independently. An overly restrictive social law can prevent some of the agents in Π from reaching their goal. We address these kind of social laws as *infeasible social laws*. Note that if $\bar{l}$ is an infeasible social law, it is not robust by definition, and the function IS_SOLVABLE_FORALL(Π^l) returns $False$.

Lemma 2. *Let Π a MA-STRIPS problem, and let $l, \bar{l}$ be two social laws such that $A^{\bar{l}} \subseteq A^l$. Then, $\bar{l}$ is infeasible implies that l is infeasible.*

Proof. Let $\bar{l}$ be an infeasible social law. Then, there exists and agent i s.t. $\Pi_i^{\bar{l}} := \Pi_i \setminus A^{\bar{l}}$ is unsolvable. Let l be a social law s.t. $A^{\bar{l}} \subseteq A^l$. This directly implies that $\Pi_i \setminus A^l$ is unsolvable. Which means that l is an infeasible social law. □

The practical aspect of this observation with respect to search can be formulated as follows

Corollary 2. *Every successor generating function $succ_{\text{INF}}$ that marks the state s, that corresponds to $A^{\bar{l}}$, as a dead end, i. e., $succ_{\text{INF}}(s) = \emptyset$, does not prune any possible solution. Where $\bar{l}$ denotes an infeasible social law.*

Proof. Assume in contradiction that there are states $s, \bar{s}$, that correspond to the social laws $l, \bar{l}$, where s is a successor of $\bar{s}$, and let $\bar{l}$ be an infeasible social law. Then, by construction of the search we have that $A^{\bar{l}} \subseteq A^l$, which implies, by previous lemma, that l is infeasible. Hence, contradiction. The general case follows by induction on descendants of $\bar{s}$. □

To identify an infeasible social law we maintain a set of infeasible social laws. Then, throughout the search, every social law l that proves to be infeasible, is added to this set. Also, before we explore each social law l, we test whether it is feasible or not using the data we hold in the infeasible social laws set. Note that this test is much more computationally efficient then using the IS_SOLVABLE_FORALL(Π^l) function that has to solve n STRIPS planning problems where n is the number of agents.

Safety of combined pruning Finally, all is left to show for the completeness of our algorithm that it is guaranteed that the of the techniques mentioned in this section does not make the problem unsolvable.

Theorem 1. *The successor generating function $succ_{\text{CE}}(s) \cap succ_{\text{INF}}(s)$ is* safe.

Proof. Note that, by Corollary 1, $succ_{\text{CE}}(s)$ is strongly safe, and, by Corollary 2, $succ_{\text{INF}}(s)$ is safe. Thus, since $succ_{\text{CE}}(s)$ does not prune any solution, the intersection of these two is also safe. □

Additionally, we use duplicate social laws detection. We have implemented it using standard CLOSED list that contains all the social laws the algorithm encountered so far.

Heuristics in Social Laws Search

Using the pruning techniques above, we could use any uninformed search algorithms such as breadth-first or depth-first search. However, informed search algorithms which use a heuristic function to estimate the distance to the goal, are typically much more efficient. Thus, we also developed two heuristics which rely on the robustness verification:

Search Effort Based Heuristics The mechanism behind the function FIND_CONFLICT(Π) is based on formulating Π', a new STRIPS problem where any solution to Π' contains a joint plan that leads to a failure. Practically, to solve this new STRIPS problem we use an external planner.

The intuition behind our first heuristic is that, as we approach a robust social law, the planner has to invest more search effort in order to find a counter-example. Thus, we use the planner's search effort as a heuristic. Specifically, we define h_{se} as the number of states the external planner generates before it reaches a solution. In case Π contains a robust social law, meaning that the planning problem is unsolvable, the external planner will have to prove unsolvability, and thus h_{se} will probably be very high.

Note that using the function FIND_CONFLICT(Π) just to calculate $h_{se}(\Pi)$ can be computationally hard. Thus, we use a lazy search approach, and use the counter-example computed for each node to compute a heuristic value for its successors.

However, it is also possible that using h_{se} might lead to unwanted results. This is because this heuristic might cause the search to prefer social laws that make the solution more complicated, without really preventing cross agent interference. As our empirical evaluation will show, using this heuristic is not always helpful.

Counter Example Statistics Based Heuristics The second heuristic we introduce is based on the understanding that a social law might need to fix multiple potential conflicts. It is quite possible that one search branch has found how to solve one conflict, while another search branch has found how to solve a different conflict. A robust social law requires solving both conflicts. Thus, we propose to store some statistics about which operators were involved in conflicts in different regions of the search space, and prioritize restricting these operators over others.

Specifically, we maintain a table where we count how many times each grounded action appeared in the counter-example that was found by robustness verification. Furthermore, following the same intuition for preferring to restrict actions which occur before the failure, we increase the count for an action that appeared before the failure by 2, and for an action that appeared after the failure by 1.

Our heuristic then evaluate a candidate social law l by

$$h_{stat}(s) = \sum_{a \in A^l} \mathsf{count}(a)$$

where A^l is the set of grounded actions l restricts and, $\mathsf{count}(a)$ is the number of times a has appeared in previous π^f.

The advantage h_{stat} has over h_{se} is that h_{stat} should improve with time as the search continues. Moreover, experience gained in one branch of the search helps the search in other branches, similarly to clause learning in SAT (Beame, Kautz, and Sabharwal 2003). Of course, more advanced techniques for learning search guidance are available, e.g. (Xu, Fern, and Yoon 2010). We leave adapting these techniques to our setting to future work.

Preferred Operators

Identifying operators which are more likely to lead to a solution, i.e., *preferred* operators, has been shown to significantly improve search performance (Richter and Helmert 2009). We now show how a similar technique can be used when searching for a robust social law. We have formulated two criteria for identifying a preferred operator, based on the social law it leads to and the counter-example that was used to prove the social law in the current node is not robust:

early Intuitively, we would like to prevent a failure before it occurs. Thus, we will prefer to restrict actions that occurred in the counter-example *before* the failure.

public Following Brafman and Domshlak (2008) an action is called *internal* iff all $f \in \mathsf{pre}(a) \cup \mathsf{eff}(a)$ are defined as internal facts of agent i. A fact f is an *internal fact* of agent i iff no actions of another agent affect it or is affected by it. Intuitively, coordination between the agents should focus on the public (non-internal) actions, and thus we prefer to restrict only public actions which appeared in the counter-example.

Note that neither of these are a safe pruning method, as it could be possible to arrive at a robust social law by restricting internal actions that occur after the failure, since that could force one of the agents to choose an entirely different plan. However, as a heuristic method to prefer operators, these criteria can help, as our empirical evaluation shows.

In order to exploit these preferred operator criteria, we use a multi-queue mechanism. Since we have 2 criteria we use a 4 queue mechanism, where one queue contains operators that are preferred according to both criteria, another contains only operators that are preferred according to *early*, a third contains only operators that are preferred according to *public*, and a fourth queue which contains all operators.

Empirical Evaluation

We now present an empirical evaluation of the techniques described above. To summarize, we have presented:

- Two safe pruning techniques: the first prunes every social law modification that restricts actions that are not part of the counter example π^f. The second prunes unfeasible social laws.
- Two possible heuristics: The first h_{se} is based on the amount of search effort invested to find a counter example. The second h_{stat} is based on past counter examples statistics and pushes to restrict actions that appeared more times in these examples.
- Two criteria that may indicate that a given social law is preferable. The first: $early$, points out social laws that only contain actions that were executed before the failure has occurred. The second: $public$, points out social laws that only contain actions that are public. As mentioned above, these are used with a 4 queue open list, except with depth-first search, as described below. We remark that the

empirical results using only one of these criteria are not substantially different, and thus were omitted for the sake of brevity.

We have integrated these into 3 different search algorithms:

GBFS Greedy Best First Search: an Informative state space search that uses the heuristics we have presented, with or without preferred operators

BFS Breadth-First Search, with or without preferred operators.

DFS Depth First Search, with and without preferred operators. Here, since we can not use a multi-queue mechanism, preferred operators are used to ranking the successors of every search node — giving a score of 2 to nodes which satisfy both criteria, 1 to nodes which satisfy only 1, and 0 to the others.

Thus, we have 8 different configurations to compare: BFS, DFS, GBFS with h_{se}, and GBFS with h_{stat} — all of these either with preferred operators (denoted PO) or without (denoted NP). The safe pruning techniques described above were used in all of these. Preliminary results show that without these pruning techniques, very few problems are solved.

Following the work on social law robustness verification (Karpas, Shleyfman, and Tennenholtz 2017), we evaluate these configurations on benchmarks from the first Competition of Distributed and Multiagent Planners (Komenda, Stolba, and Kovacs 2016), attempting to automatically synthesize a robust social law for each planning problem. Note that these benchmarks are for cooperative planning, thus we had to make sure that there are solutions that do not rely on agents' cooperation. Benchmarks that do not meet this condition (such as DEPOT, SOKOBAN) were dropped. Also, we modified the ZENOTRAVEL domain and removed the need for passengers-planes cooperation by letting planes control the passengers (with the BOARD and DEBARK actions). A similar technique was used for the TAXI domain's problems. We assigned each goal fact to one of the agents randomly, checking that, indeed the assigned agent can achieve this goal. Additionally, we included a machine fixing domain that was introduced in Karpas *et al.* (2017). All in all, we have 91 planning problems to test from 5 different domains.

The question of which planner to use in our goal test function IS_ROBUST(Π) is interesting. As previous work on verification showed, proving robustness is better done with a planner geared towards proving unsolvability while finding a counter example is better done with a planner geared towards finding a solution quickly. Thus, in each search node's goal test we first run *Fast Downward Stone Soup 2014* (Röger, Pommerening, and Seipp 2014) with a timeout of 1800 seconds, to quickly find a counter example. If the planner times out, we then switch to running SymPA (Torralba 2016), a planner from the 2016 unsolvability International Planning Competition (IPC), which is often able to prove unsolvability. We set SymPA with a similar timeout of 1800 seconds. The search stopped exploring nodes after 1800 seconds. All planners had a memory limit of 8GB on an Intel Xeon E5-2695 CPU running at 2.10GHz (limited to one core only). We ran 16 instances at a time with the help of *GNU Parallel* (Ferrer-Mestres, Francès, and Geffner 2017).

We consider a search to be successful if it either found a robust social or proved a one does not exist. The number of successful searches in each domain, by each configuration, is reported in Table 2. The most successful configuration was GBFS with h_{stat} and without using PO with 18 solved problems. The GBFS with h_{stat} and with using PO successfully solved 17 problems.

All in all, the configurations we presented solved 22 social law search problems. Search times and the length of the returned social law are reported in Table 1. 6 problems were solved by all of the configurations. They were employed to calculate the averages reported in the last row of Table 1[3].

As our results show the BFS based search configurations are relatively slow but they bring the shortest social law length (as expected). DFS based search configurations solve more instances and are faster than BFS based configurations in most cases. However, they tend to find longer social law lengths than the other configurations, as expected.

The performance of GBFS with h_{se} based is similar to BFS concerning the number of solved problems (see Table 2). However, in most cases, the GBFS with h_{se} is quicker (see Table 1) and it seems that it works better with preferred operators (PO). Overall, it seems that h_{se} does not bring a significantly improved performance. A possible explanation, as mentioned above, is that h_{se} was designed to choose nodes that caused greater computational effort during the robustness verification run. This can be done by deleting helpful but not necessary actions, which will cause the robustness verification to work harder to find individual plans for the agents, but will not necessarily fix any conflicts.

Finally, GBFS with h_{stat} outperforms all other approaches overall. Here, the preferred operators hurt the number of problems solved in total but do help in the DRIVERLOG domain. One possible reason why preferred operators hurt performance in the rovers domain is that ROVERS does not have any internal actions, making the preferred operators less powerful.

We now describe in more detail what happened during the search on the 22 commonly solved problems. A total of 51,965 nodes were generated during our empirical evaluation for these 22 problems. Of these, not all agents were able to achieve their goal individually in 10,623 nodes, and thus these nodes were dead-ends for the search. An additional 7,783 nodes were identified as infeasible (because their social laws were supersets of the social laws for the dead-end nodes) and thus they were also declared as dead-ends.

The robustness verification procedure was thus called on the remaining 33,559 nodes. As previously mentioned, we first used Fast Downward Stone Soup 2014, which was able to find counter examples for (that is, solve) 33,476 of these. For another 57 nodes, Fast Downward was able to prove no solution exists, and thus the social law is robust. Thus, SymPA was called on 26 nodes. Of these, it was able to verify robustness for 25 of these. Only in 1 node, neither of the

[3]The complete results are available in: https://zenodo.org/record/3547384

Domain & Problem	Search Time								Social Law Length							
	BFS		DFS		GBFS (h_{se})		GBFS (h_{stat})		BFS		DFS		GBFS (h_{se})		GBFS (h_{stat})	
	PO	NP	PO	NP	PO	NP	PO	NP	PO	NP	PO	NP	PO	NP	PO	NP
ROVERS-0	TO	TO	TO	2765	TO	TO	TO	3022.3	-	-	-	38	-	-	-	58
ROVERS-1	TO	TO	2016.9	2036.1	TO	TO	2187.9	2023.7	-	-	42	47	-	-	31	31
ROVERS-2	2480.5	2380.7	1854.1	2526.7	TO	TO	1827.5	1815.3	2	2	12	13	-	-	4	3
ROVERS-3	TO	TO	2347.4	3360.2	TO	TO	TO	2933.5	-	-	58	48	-	-	-	81
ROVERS-5	TO	TO	1973	1973.5	TO	TO	TO	TO	-	-	18	18	-	-	-	-
FIX-0	84	282.2	72.6	61.3	22.2	28.6	84.1	45.7	3	3	4	19	4	6	4	25
FIX-1	85	278.5	73.1	61	22.1	28.3	85	45.4	3	3	4	19	4	6	4	25
DLOG-0	189.5	189.3	12.2	11.5	17.2	16.7	11	10.8	3	3	11	11	5	5	7	7
DLOG-2	TO	TO	TO	TO	TO	TO	119.6	118.8	-	-	-	-	-	-	12	12
DLOG-3	TO	TO	TO	TO	446.2	TO	1266.6	1633.1	-	-	-	-	12	-	14	14
DLOG-5	TO	TO	1613.1	TO	TO	TO	TO	TO	-	-	29	-	-	-	-	-
DLOG-6	TO	TO	3307.6	3312.9	TO	TO	861.6	TO	-	-	41	43	-	-	38	-
DLOG-9	TO	TO	TO	TO	TO	TO	2105	2123.7	-	-	-	-	-	-	48	53
ZNTL-0	561	557	784.6	986.2	405.8	402.5	405.5	401.3	4	4	48	49	7	7	16	14
ZNTL-1	1029.6	1028.9	549.9	556.7	433.9	433.6	372.3	372.1	4	4	45	46	11	11	20	20
ZNTL-2	TO	TO	TO	TO	TO	TO	1755.6	1754.5	-	-	-	-	-	-	20	20
ZNTL-3	TO	TO	2292.3	2200.9	TO	TO	1878.1	1876.7	-	-	142	139	-	-	29	29
TAXI-0	5.5	5.6	5.6	5.5	5.7	5.6	5.7	5.6	NSL	NSL	NSL	NSL	NSL	NSL	NSL	NSL
TAXI-1	8.3	7.9	9.6	7.8	10.8	9.4	9.6	9.9	NSL	NSL	NSL	NSL	NSL	NSL	NSL	NSL
TAXI-2	12.9	13.1	10.2	11.1	12.2	12	10.7	13.4	NSL	NSL	NSL	NSL	NSL	NSL	NSL	NSL
TAXI-5	TO	1791.3	TO	TO	TO	TO	TO	TO	-	NSL	-	-	-	-	-	-
TAXI-6	25.3	25.4	16.1	10.1	15.9	19.5	14.1	12.6	2	2	6	3	5	5	6	6
AVERAGE	329.1	393.6	251.4	281.1	152.9	154.9	162.0	148.0	3.2	3.2	19.7	24.5	6.0	6.7	9.5	16.2

Table 1: Search Time on IPC Benchmarks (Right) & Length of the Resulted Social Laws (Left). The average is based on 6 instances that were solved by all search configurations. (DLOG = driverlog, ZNTL = zenotravel, PO = using preferred operators, NP = not using preferred operators, TO = timeout, NSL = robust social law does not exist)

SEARCH TECHNIQUE	ROVERS	FIX	DLOG	ZNTL	TAXI	TOTAL
BFS-PO	1	2	1	2	4	10
BFS-NP	1	2	1	2	5	11
DFS-PO	4	2	3	3	4	16
DFS-NP	5	2	2	3	4	16
GBFS-h_{se}-PO	0	2	2	2	4	10
GBFS-h_{se}-NP	0	2	1	2	4	9
GBFS-h_{stat}-PO	2	2	5	4	4	17
GBFS-h_{stat}-NP	4	2	4	4	4	18

Table 2: Number of Successful Searches on IPC Benchmarks (DLOG = driverlog, ZNTL = zenotravel, PO = using preferred operators, NP = not using preferred operators)

planners returned any solution, in which case we declared that we were unable to find a robust social law.

Conclusion

In this paper, we have described a technique for the automatic synthesis of social laws for a given MA-STRIPS planning problem. We model this problem as a search problem in the space of possible social laws. We use the robustness verification method that was described in Karpas *et al.* as a goal test. Additionally, we describe some techniques to guide the search: two pruning techniques that together constitute a safe pruning technique, two heuristics that are based on the result of the above-mentioned robustness verification method, and two criteria that may indicate that a given social law is preferable. Our empirical evaluation shows that these techniques can find robust social laws in multiple IPC Benchmarks.

In future work, we intend to explore new ways to enhance social law search techniques by finding new heuristics, prune techniques and more general preferable operators. Also, we intend to assume more powerful social laws that can, for example, denote some action preconditions as wait-for preconditions. Our technique can probably tackle robust social law synthesis more realistic settings, assuming there are appropriate methods for social law robustness verification method, e.g. using the method that was introduced by Nir and Karpas (2019) for robust social law synthesis for temporal multi-agent planning problems.

Acknowledgments

The work of Alexander Shleyfman was supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

References

Beame, P.; Kautz, H. A.; and Sabharwal, A. 2003. Understanding the power of clause learning. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 1194–1201.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 28–35.

d'Inverno, M., and Luck, M. 2004. *Understanding agent systems*. Springer.

Ferrer-Mestres, J.; Francès, G.; and Geffner, H. 2017. Combined task and motion planning as classical AI planning. *CoRR* abs/1706.06927.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2(3/4):189–208.

Fiser, D.; Torralba, Á.; and Shleyfman, A. 2019. Operator mutexes and symmetries for simplifying planning tasks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, 7586–7593.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Horling, B., and Lesser, V. R. 2004. A survey of multi-agent organizational paradigms. *Knowledge Eng. Review* 19(4):281–316.

Karpas, E.; Shleyfman, A.; and Tennenholtz, M. 2017. Automated verification of social law robustness in STRIPS. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, 163–171.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Keren, S.; Gal, A.; and Karpas, E. 2018. Strong stubborn sets for efficient goal recognition design. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, 141–149.

Klusch, M. 1999. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Berlin, Heidelberg: Springer-Verlag, 1st edition.

Komenda, A.; Stolba, M.; and Kovacs, D. L. 2016. The international competition of distributed and multiagent planners (codmap). *AI Magazine* 37(3):109–115.

Morales, J.; Wooldridge, M. J.; Rodríguez-Aguilar, J. A.; and López-Sánchez, M. 2018. Off-line synthesis of evolutionarily stable normative systems. *Autonomous Agents and Multi-Agent Systems* 32(5):635–671.

Nir, R., and Karpas, E. 2019. Automated verification of social laws for continuous time multi-robot systems. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, 7683–7690.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Röger, G.; Pommerening, F.; and Seipp, J. 2014. Fast downward stone soup 2014. *The 2014 International Planning Competition*.

Shoham, Y., and Leyton-Brown, K. 2009. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Tennenholtz, M., and Moses, Y. 1989. On cooperation in a multi-entity model. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, 918–923.

Torralba, Á. 2016. SymPA: Symbolic Perimeter Abstractions for Proving Unsolvability. In *UIPC 2016 planner abstracts*, 8–11.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings]*, 491–515.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Woolridge, M. 2001. *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc.

Xu, Y.; Fern, A.; and Yoon, S. W. 2010. Iterative learning of weighted rule sets for greedy search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 201–208.