Planning for Cooperative Multiple Agents with Sparse Interaction Constraints

Guy Revach,[1] **Nir Greshler,**[2] **Nahum Shimkin**[2]
[1] The Institute for Signal and Information Processing (ISI),
Department of Information Technology and Electrical Engineering (D-ITET), ETH Zurich, Switzerland
[2] Viterbi Faculty of Electrical Engineering, Technion, Haifa, Israel
revach@isi.ee.ethz.ch, nirgreshler@campus.technion.ac.il, shimkin@ee.technion.ac.il

Abstract

We consider the problem of cooperative multi-agent planning (MAP) in a deterministic environment, with a completely observable state. Most tractable algorithms for MAP problems assume *sparse interactions* among agents and exploitable problem structure. We consider a specific model for representing interactions among agents using *soft cooperation constraints (SCC)*, which enables a compact representation of symmetric dependencies. We present a two-step planning algorithm that breaks down a multi-agent problem with K agents, to multiple instances of independent single-agent problems, such that the aggregation of the single-agent plans is optimal for the group. We propose an efficient algorithm for computing the single-agent optimal plan under a given set of soft constraints, denoted as the *response function*. We then utilize a well-known graphical model for efficient min-sum optimization in order to find the optimal aggregation of the single agent response functions. The proposed planning algorithm is complete, optimal, and effective when interactions among the agents are sparse. We further indicate some useful extensions to the basic SCC formulation presented here.

1 Introduction

The problem of cooperative multi-agent planning (MAP) is motivated by many real-world applications in a variety of domains, such as military, logistics, and search-and-rescue. In these problems, agents must coordinate their decisions to maximize their (joint) team value. When the state of the environment and all agents is fully-observable by each agent, the planning problem can be formalized as a multi-agent Markov decision process (MMDP, Boutilier 1996). However, these models suffer from exponential increase in the size of the state and action spaces in the number of agents, which makes them computationally intractable in general. Specific structural assumptions are therefore required for an optimal solution to be feasible.

An important class of problems concerns high-level planning problems, where agents are essentially independent except for a prescribed set of possible interactions that can facilitate the plan execution. These types of problems are typically characterized by *loose coupling* and *sparse interactions* between agents, and some models exploit this fact to develop efficient algorithms. The complexity of such algorithms is often described by means of the problem coupling level. For instance, (Nissim, Brafman, and Domshlak 2010) propose a fully distributed planning algorithm, based on the MA-STRIPS (Brafman and Domshlak 2008) model, and (Melo and Veloso 2011) propose approximate algorithms based on the decentralized sparse-interaction MDPs model.

Another common approach is to exploit the problem structure by using a compact representation with factored models. An example of such a representation is the *coordination graph* (Guestrin, Koller, and Parr 2002), also referred to as *interaction graph* (Nair et al. 2005) or *collaborative graphical games* (Oliehoek, Whiteson, and Spaan 2012), which is solved using a graph-based optimization method, such as variable elimination (VE) (Guestrin, Koller, and Parr 2002; Larrosa and Dechter 2003), or by distributed methods as investigated in the field of distributed constraint optimization problem (DCOPs, Fioretto, Pontelli, and Yeoh 2018).

It is also possible to exploit locality of interactions (Oliehoek et al. 2008; Melo and Veloso 2011) and reward structure in transition-independent models, both centralized (Scharpff et al. 2016) and decentralized (Becker et al. 2004). More specifically, in (Scharpff et al. 2016) reward dependencies are represented using conditional return graphs (CRGs) which are solved by a branch-and-bound policy search algorithm. In (Becker et al. 2004) a general formulation is suggested to represent the reward structure, using the notion of *events*. A coverage set algorithm is presented to find optimal policies. Scalability can often be improved even on more complex models, such as Network Distributed Partially Observable MDP (ND-POMDP), by leveraging sparse and structured interactions among agents. For example, the CBDP (Kumar and Zilberstein 2009) algorithm is exponential only in the width of agents interaction graph.

In this paper, we focus on the multi-agent planning problem in a *deterministic* environment, where interactions between agents are symmetric and sparse. Possible interactions are captured using a notion of *soft cooperation constraints (SCC)*, where agents can affect the cost function by jointly satisfying prescribed constraints in state and time.

This formulation is akin to the event-based formulation of (Becker et al. 2004), although less general to allow more specific and explicit computation schemes for each agent.

Based on the SCC model, we present a complete and optimal two-step planning algorithm, effective mostly in cases where interactions among agents are sparse. It is a dynamic programming (DP)-based algorithm, that decouples a multi-agent problem with K agents to K independent single-agent problems, such that the aggregation of the single-agent plans is optimal for the group. More specifically, in the first step we independently compute each agent's *response function*, which is its optimal plan with respect to all possible assignments of the timing variables of its associated constraints. We present an explicit algorithm for computing the response function, and provide a detailed complexity analysis. The second step is a centralized global *plan merging*, in which an optimal assignment to the timing variables is found under the *minimum-sum* objective. A *factor graph*, which captures dependencies among cooperative agents and exploits the internal structure of the problem, is applied to the problem with a variable elimination algorithm for efficient min-sum optimization.

Complexity analysis shows that the proposed algorithm is linear in the number of agents, polynomial in the span of the time horizon, and depends exponentially only on the number of interactions among agents.

We present a simulation implementing our proposed algorithm on a specific multi-agent planning problem. Our simulations show that the algorithm is efficient for this particular multi-agent setup and scales well in the number of agents compared to a standard solution.

We finally outline possible extensions to our model, to represent more complex cooperation constraints. For details of these extensions we refer (Revach 2018).

The remainder of the paper is organized as follows. In section 2 we present the model used and the formulation of SCC. Section 3 presents a detailed description and implementation of our algorithm, followed by a complexity analysis. In section 4 we present experimental results for our algorithm. In section 5 we present an extension to our model to include asymmetric interactions between agents. Section 6 concludes the paper and suggests directions for extensions and future research.

2 Model

We consider the finite horizon multi-agent deterministic planning problem. Our starting point is an MMDP with a factored state space, defined by a tuple $\langle \mathcal{T}, \mathbf{G}, \mathcal{S}, \mathcal{A}, \mathcal{H}, \mathcal{C}, \sigma_I, \sigma_* \rangle$, where

- $\mathcal{T} = \{0, ..., T\}$ is the time domain of length T.
- $\mathbf{G} = \{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_K\}$ is a set of K agents.
- $\mathcal{S} = \mathcal{S}_1 \times ... \times \mathcal{S}_K$ is a finite state space, factored across agents, where $\mathcal{S}_k$ is the state space of agent $\mathbf{g}_k$.
- $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_K$ is a joint action space, similarly factored across agents.
- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic transition function.
- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \cup \{\infty\}$ is a real-valued cost function.
- $\sigma_I \in \mathcal{S}$ is the initial state $\vec{s}_0$, and $\sigma_* \in \mathcal{S}$ the goal state.

Our objective is to find an optimal group policy $\vec{\pi}^*$ such that $\mathbf{J}^{\vec{\pi}}$ is minimal, i.e., $\vec{\pi}^* \in \arg\min_{\vec{\pi} \in \Pi^K} \mathbf{J}^{\vec{\pi}}$, where $\vec{\pi} = (\pi_1, ..., \pi_K)$ is the joint policy, $\mathbf{J}^{\vec{\pi}}$ is the aggregate cumulative cost defined by

$$\mathbf{J}^{\vec{\pi}} = \sum_{t=0}^{T-1} \mathcal{C}\left(\vec{s}_t, \vec{a}_t\right) \tag{1}$$

if $\vec{s}_T = \sigma_*$, and $\mathbf{J}^{\vec{\pi}} = \infty$ otherwise. Here $\vec{a}_t = (a_{t,1}, a_{t,2}, ..., a_{t,K}) \in \mathcal{A}$ is the joint action at time t, such that $a_{t,k} = \pi_k\left(s_t, t\right)$.

We next describe the sparse interactions structure. We first assume transition and cost independence across agents, namely

$$\mathcal{H}\left(\vec{s}, \vec{a}\right) = \left(\mathcal{H}_1\left(s_1, a_1\right), \ldots, \mathcal{H}_K\left(s_K, a_K\right)\right) \tag{2}$$

and

$$\mathcal{C}\left(\vec{s}, \vec{a}\right) = \sum_{k=1}^{K} \mathcal{C}_k\left(s_k, a_k\right) \tag{3}$$

Coupling between agents is introduced via a set $\Psi = \{\psi_1, ..., \psi_L\}$ of *soft cooperation constraints*. Each constraint ψ_ℓ defines a single opportunity for cooperative interaction between agents. In particular, a constraint ψ_ℓ, $\ell \in \{1, ..., L\}$, is specified by the following tuple:

$$\psi_\ell = \left\langle \mathbf{G}_\ell, \Sigma_\ell, \mathcal{C}_\ell^-, \mathcal{T}_\ell \right\rangle \tag{4}$$

where

- $\mathbf{G}_\ell = \{\mathbf{g}_k,\ k \in K_\ell\}$, with $K_\ell = (k_{\ell,1}, \ldots, k_{\ell,n(\ell)})$, is the set of $n(\ell)$ agents interacting in constraint ψ_ℓ.
- $\Sigma_\ell = \{\sigma_{\ell,k}, k \in K_\ell\}$, with $\sigma_{\ell,k} \in \mathcal{S}_k$, is a set of local interaction states. Namely, for the constraint to hold, agent k is required to be in state $\sigma_{\ell,k}$ at some prescribed time.
- $\mathcal{C}_\ell^-$ is a (reduced) immediate cost for the group for interaction, applicable when the constraint is satisfied (see equation 5).
- $\mathcal{T}_\ell$ is the constraint time domain; i.e., it is a subset of time instances at which the interaction may take place: $\mathcal{T}_\ell \subseteq \{0, 1, ..., T-1\} \cup T_\emptyset$. Here $T_\emptyset$ is a special notation for the *null assignment*, where the constraint is not satisfied, i.e., there is no interaction.

Note that the SCC formulation can be extended to represent more general constraints. For instance, a constraint can have a set of time domains, one for each agent, such that each agent interacts at a different time. Moreover, a constraint can have a subset of interaction states (instead of a single state). While the ideas are similar, for concreteness and brevity we leave these extensions to future work.

2.1 Interaction-Dependent Cost

Agents are coupled only via the constraint set Ψ. Therefore, the group cost depends on the constraints satisfied, where each satisfied constraint ψ_ℓ represents an interaction which applies the group a reduced cost $\mathcal{C}_\ell^-$. We now describe the structure of the group cost under this formulation.

Let $\tau_\ell \in \mathcal{T}_\ell$ be an *interaction timing variable* that defines the timing of the interaction under constraint ψ_ℓ. For a given assignment to the timing variable τ_ℓ, we define an indicator function that is true if all interacting agents in $\mathbf{G}_\ell$ satisfy constraint ψ_ℓ:

$$\hat{\psi}_\ell(\tau_\ell; \vec{\pi}) = \mathbb{I}_{\{\tau_\ell \neq T_\emptyset\}} \prod_{k \in K_\ell} \mathbb{I}_{\{s_{\tau_\ell,k} = \sigma_{\ell,k}\}} \tag{5}$$

where $\mathbb{I}_A$ is the $0/1$ indicator of event A. Namely, constraint ψ_ℓ is satisfied given $\tau_\ell = \tau$ if $\tau \neq T_\emptyset$ and all interacting agents in ψ_ℓ arrive at their interaction state at time τ.

Furthermore, $\vec{\tau}$ is the interaction vector, and $\mathcal{D}$ is its domain, i.e., the cross space of all constraint time domains:

$$\vec{\tau} = (\tau_1, \tau_2, ..., \tau_L) \in \mathcal{T}_1 \times \mathcal{T}_2 \times ... \times \mathcal{T}_L \triangleq \mathcal{D} \tag{6}$$

$\vec{\tau}_k$ is the timing vector of all constraints involving agent $\mathbf{g}_k$ (with domain $\mathcal{D}_k$).

Under this new formulation, given an initial state $\sigma_I \in \mathcal{S}$, a goal state $\sigma_* \in \mathcal{S}$, and a constraint set Ψ, our objective is to find the optimal group policy where $\mathbf{J}^{\vec{\pi}}$ in equation 1 is now

$$\mathbf{J}^{\vec{\pi}}(\vec{\tau}) = \sum_{t=0}^{T-1} \sum_{k=1}^{K} \mathcal{C}_{0,k}(s_{t,k}, a_{t,k}) + \sum_{\ell=1}^{L} \hat{\psi}_\ell(\tau_\ell; \vec{\pi}) \mathcal{C}_\ell^- \tag{7}$$

where $\mathcal{C}_{0,k}$ is the single-agent independent immediate cost with no consideration of interactions. Namely, it is the sum of all agents' independent immediate cost plus the sum of the reduced costs of all satisfied constraints.

Note that now the multi-agent optimal policy $\vec{\pi}$ is a parametric policy with respect to timing variables, and the aggregate cumulative cost $\mathbf{J}^{\vec{\pi}}$ is a function of the timing variables. Effectively, there may be a different optimal policy for each assignment of timing variables. Furthermore, $\mathbf{J}_k^*(\vec{\tau}_k)$ is the optimal response function (i.e., the optimal cumulative cost) for agent $\mathbf{g}_k$ given an assignment of the timing vector $\vec{\tau}$.

Our objective is to minimize the multi-agent cumulative cost under L interaction constraints:

$$\mathbf{J}^* = \min_{\vec{\tau} \in \mathcal{D}} \min_{\vec{\pi} \in \Pi^K} \left\{ \mathbf{J}^{\vec{\pi}}(\vec{\tau}) \right\} \tag{8}$$

where $\mathbf{J}^{\vec{\pi}}(\vec{\tau})$, defined by equation 7, is decomposable, and where each single agent cost function depends only on the single agent policy. Therefore, we may switch the order of summation to compute independently for each agent:

$$\mathbf{J}^{\vec{\pi}}(\vec{\tau}) = \sum_{k=1}^{K} \sum_{t=0}^{T-1} \mathcal{C}_{0,k}(s_{t,k}, a_{t,k}) + \sum_{\ell=1}^{L} \hat{\psi}_\ell(\tau_\ell; \vec{\pi}) \mathcal{C}_\ell^- \tag{9}$$

provided that $\vec{s}_T = \sigma_*$ and $\mathbf{J}^{\vec{\pi}}(\vec{\tau}) = \infty$ otherwise.

We can then minimize each single agent cost independently for any given assignment of the timing vector $\vec{\tau} \in \mathcal{D}$ (and specifically $\vec{\tau}_k$ for each agent $\mathbf{g}_k$). After the optimal single agent response functions are found, we need to find the optimal assignment for the timing variables. Let us observe that the multi-agent problem decomposition results in a min-sum optimization problem:

$$\vec{\tau}^* \in \arg\min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^*(\vec{\tau}_k) \tag{10}$$

that is, the sum of optimal response functions. We can use this structure to our advantage by applying an efficient optimization algorithm.

3 *DIPLOMA* - Distributed Planning and Optimization Algorithm for Multiple Agents

In this section we present the *DIstributed PLanning and Optimization algorithm for Multiple Agents (DIPLOMA)*, which addresses the previous multi-agent interaction model and optimizes cost and policy. Using this model, we are able to decompose a global multi-agent planning problem into a two-step problem. First, K distributed independent single-agent planning problems are solved. Second, we optimize the global solution with respect to the cooperation constraints by selecting a plan for each agent.

We now describe the steps of our proposed algorithm, presented in algorithm 1:

1. **Response Function Computation**

   For every agent $\mathbf{g}_k \in \mathbf{G}$, compute the single agent response function independently,

$$\forall \vec{\tau}_k \in \mathcal{D}_k \ , \ \mathbf{J}_k^*(\vec{\tau}_k) = \min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k}(\vec{\tau}_k) \tag{11}$$

   It may be computed using various dynamic programming algorithms, and more specifically using the algorithms described next, in detail. This step can be parallelized over agents.

2. **Plan Merging**

   Compute the optimal total multi-agent cost by minimizing the sum single agent response with respect to the constraint variables. More specifically:

$$\mathbf{J}^* = \min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^*(\vec{\tau}_k) \tag{12}$$

   The minimization process can be carried out efficiently using factor graph modeling and a variable elimination algorithm, as described below. Let $\vec{\tau}^*$ denote the optimal assignment of the constraint variables.

3. **Policy Backtracking**

   (a) For every agent $\mathbf{g}_k \in \mathbf{G}$, backtrack the single agent optimal policy independently:

$$\pi_k^* \in \arg\min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k}(\vec{\tau}_k^*) \tag{13}$$

   (b) The global optimal multi-agent policy is then given by

$$\vec{\pi}^* = \{\pi_1^*, \pi_2^*, ..., \pi_k^*, ..., \pi_K^*\} \tag{14}$$

Algorithm 1 DIPLOMA

```
1: returns $\vec{\pi}^*$, the optimal group policy
2: inputs: MMDP, $\Psi$
3: for all $\mathbf{g}_k \in \mathbf{G}$ do          ▷ response function computation
4:     for all $\vec{\tau}_k \in \mathcal{D}_k$ do
5:         $\mathbf{J}_k^*(\vec{\tau}_k) = \min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k}(\vec{\tau}_k)$
6: $\mathbf{J}^* = \min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^*(\vec{\tau}_k)$          ▷ plan merging
7: $\vec{\tau}^* \in \arg\min_{\vec{\tau} \in \mathcal{D}} \sum_{k=1}^{K} \mathbf{J}_k^*(\vec{\tau}_k)$
8: for all $\mathbf{g}_k \in \mathbf{G}$ do
9:     $\pi_k^* \in \arg\min_{\pi_k \in \Pi_k} \mathbf{J}_k^{\pi_k}(\vec{\tau}_k^*)$
10: $\vec{\pi}^* = \{\pi_1^*, \pi_2^*, ..., \pi_k^*, ..., \pi_K^*\}$
11: return $\vec{\pi}^*$
```

3.1 Response Function Computation

The main step of our proposed algorithm is computing the single agent response function with respect to constraint timing variables. We now describe algorithms to compute $\mathbf{J}_k^*$ efficiently for each agent. Before describing the algorithms in detail, we present a few basic definitions and notations:

- The algorithms presented are from a single agent perspective; therefore, we omit the index k from the notation wherever possible.
- $\mathcal{V}^*(\sigma, \sigma_*, \tau)$, the *cost-to-state*, is the optimal cumulative cost from state σ at time $t = \tau$ to the target state σ_* in $T - \tau$ time steps.
- $\mathcal{J}^*(\sigma_I, \sigma, \tau)$, the *cost-from-state*, is the optimal cumulative cost from initial state σ_I at time $t = 0$ to state σ in τ time steps, and $\mathbf{J}^* = \mathcal{J}^*(\sigma_I, \sigma_*, T)$.
- More generally, $\mathcal{V}_\Diamond^*(s_I, s_g, \tau_I, \tau_g)$ is the optimal cumulative cost from state s_I at time $t = \tau_I$ to state s_g at time $t = \tau_g$ in $\tau_g - \tau_I$ time steps.

We start with the following computation in algorithm 2 of the cost-to-state and cost-from-state. The result is required only for intermediate states in the agent's constraint set. A natural implementation by Dynamic Programming (DP) computes these costs via a single pass for all states and times.

Algorithm 2 Costs to and from states

```
1: for all $\tau \in \{1, .., T\}$ and $\sigma \in \{\sigma_\ell\}$ do
2:     Compute $\mathcal{J}^*(\sigma_I, \sigma, \tau)$ iteratively using DP.
3: for all $\tau \in \{T-1, .., 0\}$ $\sigma \in \{\sigma_\ell\}$ do
4:     Compute $\mathcal{V}^*(\sigma, \sigma_*, \tau)$ iteratively using DP.
5: Cache all results for later use.
```

The single agent response function is the optimal cumulative cost with respect to the timing variables, i.e., the optimal plan from the initial state to the goal state, while satisfying the constraints in times specified by the timing variables. To simplify the presentation, we start by showing how to compute the single-agent response function with a single interaction (i.e., a single constraint), and then follow with the general case of L interactions.

Let τ_ℓ be a single timing variable (i.e., $L = \ell = 1$), and $\mathbf{J}_\sigma^*(\tau_\ell)$ the optimal cumulative cost from initial state σ_I to goal state σ_* in T time steps, via the intermediate state σ_ℓ; i.e., $s_{\tau_\ell} = \sigma_\ell$. For a given assignment of τ_ℓ, we can compute the response function in this simple case as:

$$\mathbf{J}_\sigma^*(\tau) = \begin{cases} \mathbf{J}^*, & \tau = T_\emptyset \\ \mathcal{J}^*(\sigma_I, \sigma_\ell, \tau) + \mathcal{V}^*(\sigma_\ell, \sigma_*, \tau), & \text{otherwise} \end{cases} \tag{15}$$

Namely, it is computed by two parts: planning from the initial state σ_I in time $t = 0$ to the constraint state σ_ℓ in time $t = \tau_\ell$, and from the latter to the goal state at T (i.e., for $T - \tau$ time steps). If $\tau_\ell = T_\emptyset$, the constraint is not imposed. Hence, $\mathbf{J}_\sigma^*(\tau_\ell) = \mathbf{J}^*$, i.e., the optimal cost with no consideration of interactions.

The generalization for $L = L_k$ constraints (the number of constraints agent k is involved in) follows the same idea. We need to compute the response function $\mathbf{J}_{\sigma_1,...,\sigma_L}^*(\tau_1, ..., \tau_L)$ for every assignment of L timing variables. We present an incremental scheme that efficiently avoids repeated computation of given segments:

1. Pre-compute the state-to-state cost functions by dynamic programming, and cache the results for later use:

1.1. Apply algorithm 2.

1.2. Pre-compute $\mathcal{V}_\Diamond^*$ using algorithm 3.

2. Build the response function from the bottom up using the previously cached values that were pre-computed in the previous step, using algorithms 4 and 5. For simplicity we use the following concise notation, for $1 \le \ell \le L$:

$$\mathbf{J}^*\{\ell\} \triangleq \mathbf{J}_{\sigma_1,...,\sigma_\ell}^*(\tau_1, ..., \tau_\ell) \tag{16}$$

In algorithm 5 we show how to compute $\mathbf{J}^*\{\ell+1\}$ for all assignments to $\tau_{\ell+1}$, given $\mathbf{J}^*\{\ell\}$ for a specific assignment to $\tau_1, \ldots, \tau_\ell$. The idea is essentially to replace $\mathcal{V}_\Diamond^*(\sigma_{\ell_1}, \sigma_{\ell_2}, \tau_{\ell_1}, \tau_{\ell_2})$ by the sum $\mathcal{V}_\Diamond^*(\sigma_{\ell_1}, \sigma_{\ell+1}, \tau_{\ell_1}, \tau_{\ell+1}) + \mathcal{V}_\Diamond^*(\sigma_{\ell+1}, \sigma_{\ell_2}, \tau_{\ell+1}, \tau_{\ell_2})$ when constraint $\ell + 1$ is added with timing assignment $\tau_{\ell+1}$ between existing τ_{ℓ_1} and τ_{ℓ_2}.

Algorithm 3 Multiple constraint response - step 1.2

```
1: for all $\sigma_i, \sigma_j \in \{\sigma_1, \sigma_2, ..., \sigma_L\}$ do
2:     for all $\tau_i \in \mathcal{T}_i$ do
3:         for all $\tau_j \in \mathcal{T}_j, \tau_j > \tau_i$ do
4:             Compute $\mathcal{V}_\Diamond^*(\sigma_i, \sigma_j, \tau_i, \tau_j)$
5:             Cache the results for later use.
```

Algorithm 4 Multiple constraint response - step 2

```
1: for all $\ell \in \{1, 2, ..., L-1\}$ do
2:     for all $\tau_1, \tau_2, ..., \tau_\ell$ do
3:         For a given assignment to $\tau_1, \tau_2, ..., \tau_\ell$
4:         Such that $\tau_{i_1} \le \tau_{i_2} \le ... \le \tau_{i_\ell}$
5:         Compute $\mathbf{J}^*\{\ell+1\}$ from $\mathbf{J}^*\{\ell\}$ for all $\tau_{\ell+1} \in$
   $\mathcal{T}_{\ell+1}$, using Algorithm 5
```

Algorithm 5 Multiple constraint response - inner algorithm

```
1: Assume $\tau_{i_1} \leq \tau_{i_2} \leq ... \leq \tau_{i_\ell}$
2: $p = 1$    ▷ Initialize pivot index
3: $b = 1$    ▷ Set *baseline* flag
4: for all $\tau_{\ell+1} \in \{0, 1, ..., T-1\}$ do
5:     if $p = 1$ then
6:         if $b = 1$ then
7:             $\mathbf{J}^*_{base} = \mathbf{J}^*\{\ell\} - \mathcal{J}^*(\sigma_I, \sigma_{i_1}, \tau_{i_1})$    ▷ Initialize a *baseline* value for $\mathbf{J}^*\{\ell+1\}$
8:             $b = 0$    ▷ Reset *baseline* flag
9:         if $\tau_{\ell+1} < \tau_{i_1}$ then
10:            $\mathbf{J}^*\{\ell+1\} = \mathbf{J}^*_{base} + \mathcal{J}^*(\sigma_I, \sigma_{\ell+1}, \tau_{\ell+1}) + \mathcal{V}^*_\Diamond(\sigma_{\ell+1}, \sigma_{i_1}, \tau_{\ell+1}, \tau_{i_1})$
11:        else    ▷ $\tau_{\ell+1} = \tau_{i_1}$
12:            $\mathbf{J}^*\{\ell+1\} = \infty$    ▷ There is no valid plan that meets the constraints
13:            $p = 2$    ▷ Increment pivot index
14:            $b = 1$    ▷ Set *baseline* flag
15:    else if $1 < p \leq \ell$ then
16:        if $b = 1$ then
17:            $\mathbf{J}^*_{base} = \mathbf{J}^*\{\ell\} - \mathcal{V}^*_\Diamond(\sigma_{i_{p-1}}, \sigma_{i_p}, \tau_{i_{p-1}}, \tau_{i_p})$
18:            $b = 0$    ▷ Reset *baseline* flag
19:        if $\tau_{\ell+1} < \tau_{i_p}$ then
20:            $\mathbf{J}^*\{\ell+1\} = \mathbf{J}^*_{base} + \mathcal{V}^*_\Diamond(\sigma_{i_{p-1}}, \sigma_{\ell+1}, \tau_{i_{p-1}}, \tau_{\ell+1}) + \mathcal{V}^*_\Diamond(\sigma_{\ell+1}, \sigma_{i_p}, \tau_{\ell+1}, \tau_{i_p})$
21:        else    ▷ $\tau_{\ell+1} = \tau_{i_p}$
22:            $\mathbf{J}^*\{\ell+1\} = \infty$    ▷ There is no valid plan that meets the constraints
23:            $p = p + 1$    ▷ Increment pivot index
24:            $b = 1$    ▷ Set *baseline* flag
25:    else    ▷ $p > \ell$
26:        if $b = 1$ then
27:            $\mathbf{J}^*_{base} = \mathbf{J}^*\{\ell\} - \mathcal{V}^*(\sigma_{i_\ell}, \sigma_*, \tau_{i_\ell})$    ▷ Initialize a *baseline* value for $\mathbf{J}^*\{\ell+1\}$
28:            $b = 0$    ▷ Reset *baseline* flag
29:        $\mathbf{J}^*\{\ell+1\} = \mathbf{J}^*_{base} + \mathcal{V}^*_\Diamond(\sigma_{i_\ell}, \sigma_{\ell+1}, \tau_{i_\ell}, \tau_{\ell+1}) + \mathcal{V}^*(\sigma_{i_{\ell+1}}, \sigma_*, \tau_{i_{\ell+1}})$
30: $\mathbf{J}^*\{\ell+1\}(T_\emptyset) = \mathbf{J}^*\{\ell\}$    ▷ $\tau_{\ell+1}$ is equal to the *null* assignment, namely no constraint
```

3.2 Plan Merging

The plan merging step of our proposed algorithm requires finding an optimal assignment to the timing variables while optimizing the global cost function $\mathbf{J}^*(\vec{\tau})$. This is a weighted constraint satisfaction programming problem, which is $\mathcal{NP}$-hard in the general case (Larrosa and Dechter 2003). In the special case of the min-sum objective (equation 12), we can reduce optimization complexity by using models that consider the internal structure of the dependency among agents. A graphical model, called factor graph (Loeliger 2004), describes the interaction among agents, and captures agent dependency or independency, therefore leading to more efficient optimization algorithms.

A factor graph contains variable nodes representing constraint variables (the timing variables), and factor nodes representing single-agent cost functions $\mathbf{J}^*_k(\vec{\tau}_k)$. Edges connect a cost function to all the variables associated with the constraints involved in that cost function. Figure 1 illustrates how the min-sum optimization problem is represented using a factor graph. In this example, we have three cooperation constraints, where $\mathbf{G}_1 = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}$, $\mathbf{G}_2 = \{\mathbf{g}_1, \mathbf{g}_3\}$, and $\mathbf{G}_3 = \{\mathbf{g}_3, \mathbf{g}_4\}$.

On the factor graph we apply a variable elimination (VE) algorithm, which is used mainly for exact inference (Koller and Friedman 2009). VE exploits the internal structure of the problem and reduces computations (Larrosa and Dechter 2003).

The factor graph structure and the VE elimination ordering have a major effect on the complexity and efficiency of the algorithm, which is out of the scope of this work (see Koller and Friedman 2009). However, in the next section we present several representative cases. The scheme for applying VE to solve the optimization problem is described in (Revach 2018).

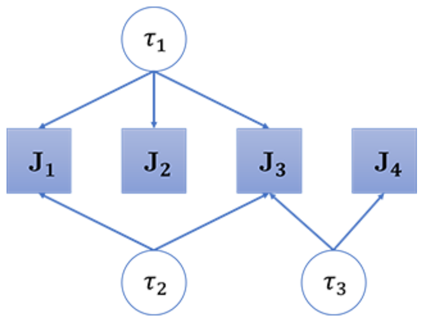

Figure 1: Factor graph example

3.3 Complexity Analysis

In this section we present an overall complexity analysis of our proposed algorithm. We first present the complexity of the response function computation, followed by an overall analysis of a few representative cases, and establish an upper bound on the complexity of planning problems. The complexity result is formulated in terms of the overhead of planning for a multi-agent system as a function of planning for each single agent in isolation, when considering the same problem structure. More specifically, we denote $\mathbf{T}(\mathcal{V}^*, T)$ and $\mathbf{T}(\mathcal{J}^*, T)$ as the time complexity of computing a single-agent cost-to-state and cost-from-state, respectively, over the time horizon T, and assume $\mathbf{T}(\mathcal{V}^*, T) = \mathbf{T}(\mathcal{J}^*, T)$.

For the response function computation, the first step of pre-computation (algorithms 2 and 3) is of the order of $L^2 \cdot \sum_{\tau_\ell \in \mathcal{T}_\ell} \mathbf{T}(\mathcal{V}^*, \tau_\ell) \leq L^2 \cdot \frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*, T)$, where L is the number of constraints in which the agent is involved. We can use an efficient algorithm for computing a single agent cost-to-state from every initial state to a fixed and specific goal state (e.g., using dynamic programming) and denote its time complexity as $\mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T)$. In that case, we may reduce the time complexity by a factor of L, compared to single agent planning, i.e., $L \cdot \frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T)$. The time complexity of the second step (algorithm 4) is dominated by $\mathcal{O}(T^L)$. Therefore, the overall complexity for computing the response function is

$$L \cdot \frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + \mathcal{O}(T^L) \tag{17}$$

In the case of a single constraint, this reduces to $2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T)$ (equation 15).

The complexity of the plan merging step, and more specifically the VE algorithm, depends on the scope size of each factor; that is, the number of variables to which each factor is connected. The total complexity has an exponential dependency in the scope size of the factors and it is of the order of $\mathcal{O}((K + L) \cdot d^m)$ where m is the maximal scope size of factors and d is the maximal number of values of each variable. For a detailed complexity analysis of the VE algorithm on a factor graph, see (Revach 2018; Koller and Friedman 2009).

We now present an analysis of the overall time complexity for several representative cases. We start with a very sparse case, where there are $2 \cdot L$ agents, each of which is involved in only one cooperation constraint, i.e., $K = 2 \cdot L$. The response function computation time complexity is dominated by $2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T)$ and is linear in the span of the time horizon. Each timing variable does not depend on any of the other variables. The time complexity of eliminating a single variable is dominated by $\mathcal{O}(T)$; i.e., it is also linear in the span of the time horizon. The overall complexity is $K \cdot [2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T)] + L \cdot \mathcal{O}(T) = 2 \cdot K \cdot \mathbf{T}(\mathcal{V}^*, T) + \frac{3}{2} \cdot K \cdot \mathcal{O}(T)$. Because of the inherent decoupling in this case, this is equal to solving $L = \frac{K}{2}$ independent problems.

In a dense case we consider two agents with $L \geq 2$ cooperation constraints between them (i.e., each agent is involved in L constraints). The time complexity of the response function computation is exponential in L. As all the timing variables belong to the same factors, they are therefore dependent. The time complexity of the plan merging is also exponential in L, but it is not the dominating part. The overall complexity is dominated by $2 \cdot L \cdot \left(\frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + \mathcal{O}(T^L)\right)$.

We now consider an hierarchical case, where each constraint involves two agents and the factor graph is a balanced N-tree with depth M. There are N^M agents (factors) that are represented as leaf nodes in the tree, and $K - N^M$ agents that are not represented as leaf nodes. Here, K is equal to $K = \sum_{m=0}^{M} N^m$; therefore, the total number of cooperation constraints is equal to $L = K - 1$. Each of the leaf agents is involved in only one cooperation constraint; therefore, the complexity of computing their response function is just linear: $N^M \cdot (2 \cdot \mathbf{T}(\mathcal{V}^*, T) + \mathcal{O}(T))$. An agent that is not a leaf node in the tree is involved in $N + 1$ cooperation constraints. Therefore, the complexity of computing their response function is $\left(K - N^M\right) \cdot (N + 1) \cdot \left(\frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + \mathcal{O}\left(T^{N+1}\right)\right)$. In the case of a tree, the plan merging is executed bottom up from the leaf nodes to the root node. Every factor that is not a leaf generates an $N + 1$ cliques (see Koller and Friedman 2009) of timing variables (i.e., all the variables on which the factor depends). Therefore, the complexity of plan merging is dominated by the size and number of cliques. The complexity of eliminating a clique by a VE algorithm is dominated by $\mathcal{O}\left(T^{N+1}\right)$, and the number of cliques is equal to $\mathbf{C_L} = K - N^M$. Note that the process of eliminating cliques in the same level of the tree can be distributed and parallelized.

Finally, we define a coupling measure ρ for the system as the maximal number of constraints in which each agent is involved,

$$\rho = \max_k L_k, \quad k = 1, \ldots, K \tag{18}$$

where L_k is the number of constraints in which agent $\mathbf{g}_k$ is involved. An upper bound for the complexity is linear in K, polynomial in T, and exponential only in ρ:

$$\mathcal{O}\left(K \cdot \rho \cdot \left(\frac{T}{2} \cdot \mathbf{T}(\mathcal{V}^*_{\mathcal{B}}, T) + T^\rho\right)\right) \tag{19}$$

4 Experiments

In this section we present the results of basic experiments performed using DIPLOMA, in order to validate its correctness and test its time complexity. We compare the algorithm's performance against a centralized DP algorithm solving the underlying MMDP. We use the same DP algorithm for calculating $\mathcal{J}^*(\sigma_I, \sigma, \tau)$ and $\mathcal{V}^*(\sigma, \sigma_*, \tau)$ in algorithm 2. All simulations were performed on an Intel i7-8700 CPU @ 3.20Ghz machine with 16.0 GB RAM.

We ran our simulations on a simple grid world example where several agents have to travel from an initial location to a goal location in T time steps while collecting as many boxes as possible. Each box has its own reward and associated agent, and some boxes can be picked by two agents together in order to gain a double reward. In order for agents to pick a box together, they have to meet at the box location at the same time. Agents can move *up*, *left*, or *right*, and collect the reward upon moving up from their box location. Our goal is to find an optimal joint plan such that the group reward is maximized. Note that in this example we use reward instead of cost used in the model; however, replacing between the two is trivial by taking negative rewards. This problem is depicted in figure 2 for a grid of 10×10 and four agents ($K = 4$). This problem is quite simple but can represent scheduling problems, box-pushing, search-and-rescue and more.

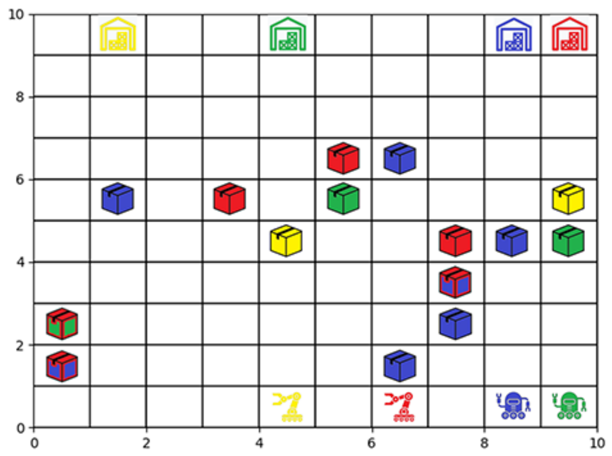

Figure 2: A box collecting problem on a 10×10 grid with four agents ($K = 4$), denoted by four different colors. All agents start in the bottom row and have to arrive to the corresponding warehouse at the top row within T time steps. Each agent can only pick boxes of its color. Agents can cooperate in three different locations ($L = 3$), illustrated by a two-colored box. For instance, in the box located in $(0, 2)$, the red agent can pick the box alone and receive a reward of 3, or it can pick it with the assistance of the blue agent and receive a reward of 6.

We ran our simulation on a fixed grid size of 10×10 $\left(|\mathcal{S}| = 100^K\right)$, a fixed horizon of $T = 20$ time steps, and different values for number of agents (K), and constraints (L). For every value of K we generated 20 random environments (with a random number of constraints) and measured the runtime of the centralized reference algorithm and DIPLOMA. Figure 3 demonstrates how our proposed algorithm scales in the number of agents, and depends on the coupling measure ρ (see equation 18). We also compare the runtime of our algorithm using the VE algorithm for the plan merging step, compared to a brute-force (BF) optimization. Elimination ordering for VE was determined by a simplified min-neighbors criteria (Koller and Friedman 2009). The reference algorithm does not depend on the coupling measure (i.e., number of constraints), but for $K = 3$, has a runtime higher by two orders of magnitude than DIPLOMA. A value of $K = 4$ makes it practically infeasible to run. DIPLOMA, on the other hand, scales well in the number of agents and depends mostly on the coupling level. Furthermore, using VE optimization for the plan merging step (compared to a brute-force optimization), reduces runtime significantly when the coupling measure increases.

5 Extension to Asymmetric Interactions

In this paper we focus on simple symmetric interactions between agents, i.e., meeting constraints where all agents must arrive at the same time for the group to benefit from the interaction. Our model, however, can be extended to include asymmetric and more complex temporal constraints, enabling a compact representation of such constraints. Furthermore, it enables the development of efficient planning algorithms that exploit the linear time complexity of solving an MDP. This can be done by applying the group interaction cost $\mathcal{C}_\ell^-$ to a specific interacting agent, called the *affected* agent, and embedding an *activation function* of the form $f_\ell : \mathcal{T} \times \mathcal{T}_\ell \to \{0, 1\}$ into the affected agent's immediate cost. The activation function defines a set of time instances where the interaction cost is applicable.

As an example, one can consider a scenario where a *facilitating* agent can arrive at a certain state in time $\tau \in \mathcal{T} = \{1, ..., 10\}$, which opens a 10 time steps window following time τ, allowing the second agent to receive an additional reward for each time step (within this time window) in which it is in a related state. If we formulate this interaction using a distinct constraint for each possible state-time pair, we need $10^2 = 100$ constraints, and thus checking about $2^{100} \approx 10^{30}$ different combinations of constraints. By formulating this interaction with an SCC, using a step activation function, we would have only 10 constraints. We would need to check only $2^{10} = 1024 \approx 10^3$ combinations of constraints of the first agent, and for each one, solve a single induced MDP for the second agent. Thus, we obtain an improvement of many orders of magnitude in this simple case.

A detailed formulation and implementation of this extension is presented in (Revach 2018), including an efficient asymmetric planning algorithm using a step activation function.

Another rather trivial extension to asymmetric interactions is to use a different interaction timing for every interacting agent in a constraint. Since we calculate the agents' response for each $\tau \in \{0, \dots, \mathcal{T}\}$ (see algorithm 5), we can choose a different value of τ for each agent in the plan merging step.

6 Discussion and Future Work

In this paper, we address the problem of fully cooperative multiple agents high-level planning problems in deterministic environments. We focus on problems where interactions between agents are symmetric and sparse, and present a framework for representing all interactions as *soft cooperation constraints* (SCC). This framework enables a compact representation of temporal constraints and can be further extended and generalized to include more types of constraints. Considering the SCC formulation, only those agents that are subject to the same cooperation constraint are coupled, forming a dependency only in a specific context.

The SCC model presented is quite general and useful in practice, and can express constraints used in realistic scenarios. The main use case is coordination of high-level actions

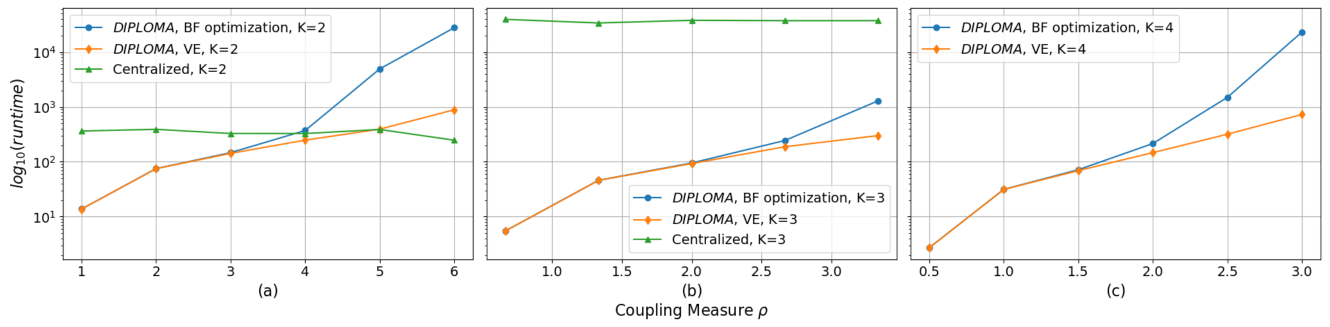

Figure 3: Simulation results for $K = 2$ (a), $K = 3$ (b), and $K = 4$ (c) on a logarithmic scale. The centralized reference planner does not depend on the coupling measure (i.e., number of constraints in the problem), but scales poorly on the number of agents, and for $K = 4$ is practically infeasible. DIPLOMA algorithm achieves an improvement of 2 orders of magnitude, and depends on the coupling level.

among autonomous agents. Example problems are the coordination of rescue or military forces, the Mars rover exploration (discussed in Becker et al. 2004) or the coordinated target tracking (discussed in Kumar and Zilberstein 2009). We can extend several combinatorial optimization problems, such as the vehicle routing problem (VRP) or the multiple traveling salesman problem, to include potential meetings between agents that provide additional rewards for the group. An SCC can also express a conflict (or collision) constraint (specifically in multi-agent path finding problems) by setting a positive or infinite interaction cost (see section 2), and using the extended formulation presented in section 5. In a similar way, we can also represent resource constraints, such as ”use at most 1 of this resource at the same time”, by adding constraints to states where the resource is used by agents.

Using this model, we are able to describe an efficient algorithm, *DIPLOMA*, which is both complete and optimal. The proposed algorithm is a two-step algorithm: a dynamic programming-based planning step and an optimization step.

In the first step, each agent plans independently and computes its response function to the associated constraints with respect to interaction timing variables. We show non-trivial and efficient algorithms for computation, which can also be distributed and parallelized. The time complexity per agent strongly depends on the span of the time horizon and the number of cooperation constraints relevant to this particular agent.

In the second step, we use a variable elimination algorithm on a factor graph to find the optimal assignment to timing variables. The algorithm exploits the internal structure of the problem and independence among agents to efficiently solve the min-sum optimization problem.

A theoretical time complexity analysis is presented, showing that the overall algorithm is linear rather than exponential in the number of agents, polynomial in the span of the time horizon, and dependent on the number of interactions among agents.

Simulations show that the algorithm is efficient compared to a standard solution and scales well in the number of agents.

An immediate direction for future research is the extension to more expressive interaction constraints, as discussed in section 5. Other possible directions for future research include generalizing the formulation of constraints by expanding the state and time domains of each constraint, defining types of agents (rather than specific agents) in a constraint, approximate methods for computing the response functions, and simulating a real-world large-scale MAP problem.

7 Acknowledgements

We thank our two anonymous reviewers for their useful and constructive comments.

References

Becker, R.; Zilberstein, S.; Lesser, V.; and Goldman, C. V. 2004. Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research* 22:423–455.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, 195–210. Morgan Kaufmann Publishers Inc.

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*, 28–35.

Fioretto, F.; Pontelli, E.; and Yeoh, W. 2018. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research* 61:623–698.

Guestrin, C.; Koller, D.; and Parr, R. 2002. Multiagent planning with factored mdps. In *Advances in neural information processing systems*, 1523–1530.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: MIT Press.

Kumar, A., and Zilberstein, S. 2009. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the International Joint*

Conference on Autonomous Agents and Multiagent Systems, AAMAS.

Larrosa, J., and Dechter, R. 2003. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints* 8(3):303–326.

Loeliger, H.-A. 2004. An introduction to factor graphs. *IEEE Signal Processing Magazine* 21(1):28–41.

Melo, F. S., and Veloso, M. 2011. Decentralized MDPs with sparse interactions. *Artificial Intelligence*.

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *AAAI*, volume 5, 133–139.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*.

Oliehoek, F. A.; Spaan, M. T.; Vlassis, N.; and Whiteson, S. 2008. Exploiting locality of interaction in factored dec-pomdps. In *Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, 517–524.

Oliehoek, F. A.; Whiteson, S.; and Spaan, M. T. 2012. Exploiting structure in cooperative bayesian games. In *Uncertainty in Artificial Intelligence - Proceedings of the 28th Conference, UAI 2012*.

Revach, G. 2018. Planning for cooperative multiple agents with sparse interactions. Master's thesis, Technion - Israel Institute of Technology, Haifa, IL.

Scharpff, J.; Roijers, D. M.; Oliehoek, F. A.; Spaan, M. T.; de Weerdt, M. M.; et al. 2016. Solving transition-independent multi-agent mdps with sparse interactions. In *AAAI*, 3174–3180.