Restricting the Maximum Number of Actions for Decision Support under Uncertainty

Marcel Gehrke[1][0000−0001−9056−7673], Tanya Braun[1][0000−0003−0282−4284], and Simon Polovina[2][0000−0003−2961−6207]

[1] Institute of Information Systems, University of Lübeck, Lübeck Germany
{gehrke, braun}@ifis.uni-luebeck.de
[2] Conceptual Structures Research Group, Sheffield Hallam University, Sheffield, UK
s.polovina@shu.ac.uk

Abstract. Standard approaches for decision support are computing a maximum expected utility or solving a partially observable Markov decision process. To the best of our knowledge, in both approaches, external restrictions are not accounted for. However, restrictions to actions often exists, for example in the form of limited resources. We demonstrate that restrictions to actions can lead to a combinatorial explosion if performed on a ground level, making ground inference intractable. Therefore, we extend a formalism that solves a lifted maximum expected utility problem to handle restricted actions. To test its relevance, we apply the new formalism to enterprise architecture analysis.

1 Introduction

Supporting decision making often involves suggesting from a pool of actions the action with the highest expected reward based on some reward function. Two standard approaches are solving a maximum expected utility (MEU) problem in a probabilistic model to find the action with the highest expected utility [21] or solving a partially observable Markov decision process (POMDP) [2,4] yielding a policy that maps belief states to actions. To the best of our knowledge, in both approaches, external restrictions are not accounted for. However, resources are not limitless, leading to restrictions on actions. Consider a small company with five employees and ten tasks to be performed. If each employee can only perform one task at a time, delegating all ten tasks is not possible. Hence, the number of possible actions (delegating a task) actually is restricted to five.

That inference is intractable in general [5] becomes noticeable if modelling all tasks and employees as propositional random variables (randvars) with the number of tasks and employees reasonably high. Further, with each task an own randvar, restricting the number of executable tasks is not straightforward. In a lifted model with lifted computations, however, inference is at most polynomial in domain sizes [22], leading to tractable inference for models with many tasks and employees. Additionally, in lifted decision support, actions are executed for sets of indistinguishable individuals. Therefore, in this paper, we investigate how to restrict actions in lifted models to given numbers of individuals.

Specifically, we focus on solving MEU problems in parameterised probabilistic decision models (PDecMs) to support online decision support in contrast to the offline support provided by (relational) POMDPs. To this end, this paper contributes (i) a restricted version of PDecMs, which allows for specifying resources, overall as well as required per action, and restrictions on the number of times an action is executable, and (ii) an algorithm called ReLiA for restricted lifted assignments, which computes all possible lifted assignments in restricted PDecMs by building on the Ford-Fulkerson algorithm for computing a maximum flow in a network [8]. Given the assignments computed, one can solve the MEU problem in the underlying model. Using ReLiA leads to significantly fewer assignments to test for MEU in contrast to working with all permutations of assignments, lifted or ground. The contributions are accompanied by an ongoing case study to highlight an application. We end with an application of our formalism to enterprise architecture (EA) analysis.

2 Case Study Setup

In this section, we show how to support decision making with PDecMs. Along the way, we recapitulate parameterised probabilistic models (PMs), first introduced by Poole [23], and PDecMs, based on [10]. The case study involves a simple case of business process modelling with a pool of employees and a set of tasks, which need delegating in a way that as many tasks as possible are delegated while avoiding tasks being overdue. The following sections set up a fitting PDecM.

2.1 Parameterised Probabilistic Model

In PMs, parameterised randvars (PRVs) represent sets of indistinguishable randvars, with logical variables (logvars) as parameters. For the case study, we model tasks being done as a PRV using a randvar for done with a logvar for tasks. For the sake of simplicity, all tasks are equally important, making them indistinguishable. After defining PMs, we set up PRVs for the case study.

Definition 1 (PRV, parfactor, PM). *Let* $\mathbf{R}$ *be a set of randvar names,* $\mathbf{L}$ *a set of logvar names,* Φ *a set of factor names, and* $\mathbf{D}$ *a set of constants. All sets are finite. Each logvar* L *has a domain* $\mathcal{D}(L) \subseteq \mathbf{D}$*. A* constraint *is a tuple* $(\mathcal{X}, C_{\mathbf{X}})$ *of a sequence of logvars* $\mathcal{X} = (X_1, \ldots, X_n)$ *and a set* $C_{\mathcal{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X_i)$*. The symbol* $\top$ *for* C *marks that no restrictions apply, i.e.,* $C_{\mathcal{X}} = \times_{i=1}^{n} \mathcal{D}(X_i)$*. A* PRV $R(L_1, \ldots, L_n), n \geq 0$ *is a construct of a randvar* $R \in \mathbf{R}$ *possibly combined with logvars* $L_1, \ldots, L_n \in \mathbf{L}$*. If* $n = 0$*, the PRV is parameterless and forms a propositional randvar. The term* $\mathcal{R}(A)$ *denotes the possible values (range) of a PRV* A*. An* event $A = a$ *denotes the occurrence of PRV* A *with range value* $a \in \mathcal{R}(A)$*. We denote a* parametric factor (parfactor) g *by* $\phi(\mathcal{A})_{|C}$ *with* $\mathcal{A} = (A_1, \ldots, A_n)$ *a sequence of PRVs,* $\phi : \times_{i=1}^{n} \mathcal{R}(A_i) \mapsto \mathbb{R}^+$ *a function with name* $\phi \in \Phi$*, and* C *a constraint on the logvars of* $\mathcal{A}$*. A PRV* A *or logvar* L *under constraint* C *is given by* $A_{|C}$ *or* $L_{|C}$*, respectively. We may omit* $|\top$ *in* $A_{|\top}$*,* $L_{|\top}$*, or* $\phi(\mathcal{A})_{|\top}$*. A* PM G *is a set of parfactors* $\{g_i\}_{i=1}^{n}$*.*

The term $rv(P)$ refers to the set of PRVs with their constraints in a parfactor or PM, $lv(P)$ to the logvars. The term $gr(P)$ denotes the set of all instances of P w.r.t. given constraints. An instance is an instantiation (grounding) of P, substituting the logvars in P with a set of constants from given constraints. If P is a constraint, $gr(P)$ refers to the second component $C_{\mathbf{X}}$. Given a parfactor $\phi(\mathcal{A})_{|C}$, ϕ is identical for the propositional randvars in $gr(\mathcal{A}_{|C})$.

Given $\mathbf{R} = \{Done, Overdue\}$, $\mathbf{L} = \{X\}$, and $\mathcal{D}(X) = \{x_1, \ldots, x_{100}\}$, we build boolean PRVs $Done(X)$ and $Overdue(X)$. With $C = ((X), \{(x_1), (x_2)\})$, $gr(Done(X)_{|C}) = \{Done(x_1), Done(x_2)\}$. The set of $gr(Done(X)_{|\top})$ also contains the instances $Done(x_3) \ldots Done(x_{100})$.

The semantics of a model is given by grounding and building a full joint distribution. In general, a query asks for a probability distribution of a randvar given the full joint of a model and fixed events as evidence. Answering a query then requires eliminating all randvars in G not occurring in the query.

Definition 2 (Semantics, query). *With Z as normalising constant, a model G represents the full joint distribution $P_G = \frac{1}{Z} \prod_{f \in gr(G)} f$. The term $P(Q|\mathbf{E})$ denotes a* query *in G with Q a grounded PRV and $\mathbf{E}$ a set of events.*

PMs allow for modelling relational aspects between objects including recurring patterns in these relations. PDecMs build on PMs, also containing actions and utilities to support decision making.

2.2 Parameterised Probabilistic Decision Model

For the case study, we need to encode in our model that overdue tasks lead to a punishment, i.e., negative utility, and done tasks lead to a reward, i.e., positive utility. To this end, we need to form a PDecM, which contains actions and utilities [10]. Actions are modelled using PRVs with the actions in its range. Utilities are modelled with PRVs as well, which are identical for groups of indistinguishable objects, leading to utility parfactors, defined as follows.

Definition 3 (PDecM). *Let Φ_u be a set of utility factor names. A parfactor with a utility PRV U as output is a* utility parfactor $\mu(\mathcal{A})_{|C}$ *where C is a constraint on $lv(\mathcal{A})$ and μ is given by $\mu : \times_{A \in \mathcal{A} \setminus \{U\}} \mathcal{R}(A) \mapsto \mathbb{R}$, with $\mu \in \Phi_u$. The output of μ is the value of U. A* PDecM *G is a PM with an additional set G_u of utility parfactors. The term $rv(G_u)$ refers to all probability PRVs in G_u. G_u represents the combination of all utilities $U_G = \sum_{v \in \times_{r \in rv(gr(G_u))} \mathcal{R}(r)} P_{G_u}(v)$.*

The semantics already shows how lifting can speed up performance: The calculations for each $f \in gr(g_u), g_u \in G^u$ are identical, allowing for rewriting summing over all groundings $f \in gr(g_u)$ into a product of $|gr(g_u)|$ and g_u.

For the case study, we introduce a boolean action PRV $Delegate(X)$ to delegate tasks and a parameterless utility PRV $Util$ to amass rewards and punishments. Figure 1 shows the model with $Delegate(X)$ and $Util$ in grey next to regular PRVs $Done(X)$ and $Overdue(X)$. Further, a utility parfactor g^u (crosses) and a regular parfactor g_0 are depicted, which connect $Done(X)$ and

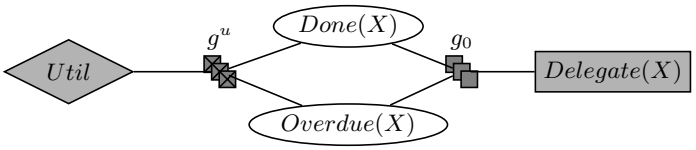

Fig. 1. Delegating tasks with action and utility nodes in grey

$Overdue(X)$ with $Delegate(X)$ and $Util$, respectively. The potentials in g^u encode that a task done on time gets a high positive utility, an overdue task done gets a small positive utility, an overdue task not done gets a high negative utility, and a task that is not done but also not overdue gets a small negative utility.

On such a PDecM, one can solve an MEU problem to determine the best actions, i.e., how to delegate tasks (without any restrictions). To define the MEU problem on a PDecM, we need to define expected utilities in a PDecM. The MEU problem asks for those action assignments that lead to the *maximum* expected utility, defined as follows.

Definition 4 (Expected utility, MEU). *Given a PDecM G, events $\mathbf{E}$, action assignments $\mathbf{a}$, the* expected utility *of G is defined by*

$$eu(\mathbf{E}, \mathbf{a}) = \sum_{v \in \times_{r \in rv(G)} \mathcal{R}(r)} P(v|\mathbf{E}, \mathbf{a}) \cdot U(v, \mathbf{E}, \mathbf{a}) \tag{1}$$

with $U(\mathbf{V})$ the utility *of PRVs $\mathbf{V}$. Then, the* MEU problem *is given by*

$$meu[G|\mathbf{E}] = (\arg\max_{\mathbf{a}} eu(\mathbf{E}, \mathbf{a}), \max_{\mathbf{a}} eu(\mathbf{E}, \mathbf{a})). \tag{2}$$

The inner product in Eq. (1) calculates a belief state $P(v|\mathbf{E}, \mathbf{a})$ and combines it with corresponding utilities $U(v, \mathbf{E}, \mathbf{a})$. By summing over the range of all PRVs of G, one obtains a scalar representing an expected utility. Equation (2) suggests a naive algorithm for solving an MEU problem, namely by iterating over all possible action assignments, solving Eq. (1) for each assignment. However, with lifting, the complexity of computing Eq. (2) is no longer exponential in the number of ground actions, enabling tractable inference in terms of domain sizes [22]. Instead of the domain sizes, the complexity is exponential in the number of groups forming due to evidence, which is usually very much lower than the number of constants in domains.

Assume that we observe whether a task is overdue. Observations are of the range of $Overdue(X)$ (boolean). Thus, X can be split into three groups, with observed values of $true$, of $false$, or no observation. For each group individually, $Delegate(X)$ can be set to either $true$ or $false$. Thus, there are 2^1 to 2^3 action assignments depending on evidence.

In Fig. 1, simply delegating all tasks leads to the highest expected utility. However, with limited resources or other more general restrictions, it might not be possible to perform all actions. E.g., employees can only perform one task at a time and there is not an unbounded number of employees. Such restrictions currently are not captured in PDecMs.

3 ReLiA: Restricting Lifted Action Assignments

In our case study, we have a pool of employees that can perform tasks, which so far have no effect on the model. Therefore, we introduce two types of restrictions. The first type restricts how often an action can be performed, i.e., an action can be performed at most five times. The second type restricts resources, e.g., an action can only be performed if sufficient resources are available. Within the boundaries of restrictions, ReLiA constructs possible action assignments. Solving the corresponding MEU problem only requires iterating over the assignments computed by ReLiA, saving unnecessary computations. Algorithm 1 shows an overview of the steps of ReLiA, which we present in the next sections.

3.1 Restricting Actions

To restrict actions, we need a way to specify resources required for an action and how often an action is executable. We first introduce resources to the model.

Definition 5 (Resources). *Let* $\mathbf{B}$ *be a set of resource names. Each resource* B *has assigned a number of available resources* $v \in \mathbb{N}$*, denoted by* $B = v$*.*

To restrict actions, we introduce action parfactors. An action parfactor specifies resources required and restrictions on executions for one action PRV.

Definition 6 (Action parfactors). *Let* Θ *be a set of* action *factor names. We denote an* action parfactor g *by* $\theta(A)_{|C}$ *with* A *an action PRV,* $\theta : \mathcal{R}(A) \mapsto ((B, \mathbb{N}), \mathbb{N})$ *a function with name* $\theta \in \Theta$ *and* $B \in \mathbf{B}$*, and* C *a constraint on the logvars of* A*. The first element of the tuple,* $(B, \mathbb{N})$*, denotes how many resources of* B *are required to set* A *to the corresponding range value for one grounding. The second element of the tuple determines how often the corresponding range value can be selected. The symbol* $\perp$ *indicates that no restrictions apply. A* restricted PDecM G *is a PDecM, which also contains a set of action parfactors* G_a*.*

Action parfactors are ignored during calculations for query answering since they do not form a part of a full joint. Given $\mathbf{B} = \{Employee\}$, we specify that there are 15 employees by setting $Employee = 15$. Further, we specify an action parfactor for our action PRV $Delegate(X)$. We specify that setting the action to true requires one employee with $(Employee, 1)$ and that the action can be set to true at most 20 times, i.e., $true \mapsto ((Employee, 1), 20)$ Setting the action to false does not require any resources and can be set to false as often as desired, leading to $\perp$ in both cases, i.e., $false \mapsto (\perp, \perp)$.

Algorithm 1 ReLiA: Construct Lifted Action Assignments under Restrictions

```
function ReLiA(Restricted model G)
    Resource graph R ← ConstructResourceGraph(G)
    Assignments A ← ObtainAssignments(R, |G_a|)
    return A                          ▷ Input to an algorithm solving an MEU problem
```

With the pool of employees incorporated, our restricted PDecM fully represents our initial setting. But, we still need a way to efficiently identify valid assignments to the action PRV, which we present next. Afterwards, we are able to iterate over these assignments to solve the corresponding MEU problem.

3.2 Computing All Action Assignments given Restrictions

Our case study has 15 employees and 100 tasks. Assume that 10 tasks are overdue (evidence), which leads to splitting X into two groups, X' for the 90 tasks without evidence and X'' for the overdue tasks. A valid action assignment would be $Delegate(X'') = true$, requiring 10 employees, $Delegate(X') = true$ for 5 X' instances (5 employees), and $Delegate(X') = false$ for the remaining 85 instances. Another assignment is $Delegate(X'') = false$, $Delegate(X') = true$ for 15 X' instances, and $Delegate(X') = false$ for the remaining 75 instances. The assignments $Delegate(X'') = true$, $Delegate(X') = false$ as well as $Delegate(X'') = false$, $Delegate(X') = false$ do not use all resources available.

To construct such action assignments, we reformulate our problem as a max-flow problem [12] to benefit from the well-understood problem of computing a maximum flow in a network with capacities. To formulate our problem as a max-flow problem, we build a resource graph from our resource restrictions. Ford and Fulkerson [9] propose a well-known algorithm to solve the max-flow problem. However, we are not only interested in one set of paths that maximises the flow, but all assignments that maximise the flow. Thus, we present how ReLiA builds a resource graph and then identifies all flows.

Constructing a Resource Graph The resource graph needs to account for the number of groundings, restrictions of how often an action is applicable, and resource restrictions. Algorithm 2 outlines how ReLiA constructs such a resource graph R with a model G as input. G is a restricted PDecM, which has its parfactors already split based on evidence. First, ReLiA adds a source node S and a target node T to R. Second, ReLiA goes through all action parfactors g_a to span R. For an action parfactor g_a, ReLiA adds a node $temp^i$ as a successor to S to R. Then, ReLiA assigns the number of groundings of g_a as capacity. By definition, the set of PRVs of each action parfactor contains exactly one action PRV. Next, ReLiA iterates over all range values of the action PRV A in g_a to account for the given restrictions specified in the action parfactor.

For each range value r, ReLiA checks if a node $A = r$ for the assignment of action r to action PRV A is already included in R. This check is included as evidence may split parfactors including action parfactors, leading to multiple parfactors regarding different instances of the same action PRV and therefore, $A = r$ may already be in R. However, the restrictions defined in the action parfactors still apply over all split action parfactors combined. If r is not included in R, ReLiA ensures that the restrictions of r are represented in R. To this end, ReLiA adds a node $A = r$ to R. Further, ReLiA adds a node $temp^r$ to R to be able to represent that a range value as well as a resource can be restricted while accounting for the fact that a resource can be used in multiple actions.

Algorithm 2 Constructing Resource Graph

function ConstructResourceGraph(Restricted model G)
  Resource graph R with starting node S and target node T
  for $g_a \in G_a$ of G **do**
    Add node $temp^i$ and edge $S \longrightarrow temp^i$ to R
    Assign $|gr(g_a)|$ as capacity to $S \longrightarrow temp^i$
    Action PRV $A \leftarrow rv(g_a)$
    for $r \in \mathcal{R}(A)$ **do**
      if $A = r$ not a node in R **then**
        Add node $A = r$ to R
        Add node $temp^r$ and edge $A = r \longrightarrow temp^r$ to R
        Get $((B, n), m)$ from $\theta(A = r)$
        Assign m as capacity to $A = r \longrightarrow temp^r$
        if B not a node in R **then**
          Add node B and edge $B \longrightarrow T$ to R
          Assign v from resource restriction $B = v$ as capacity to R
        Add edge $temp^r \longrightarrow B$ to R
        Assign capacity ∞ to $temp^r \longrightarrow B$ and n
      Add edge $temp^i \longrightarrow A = r$ to R
      Assign capacity ∞ to $temp^i \longrightarrow A = r$
  return R

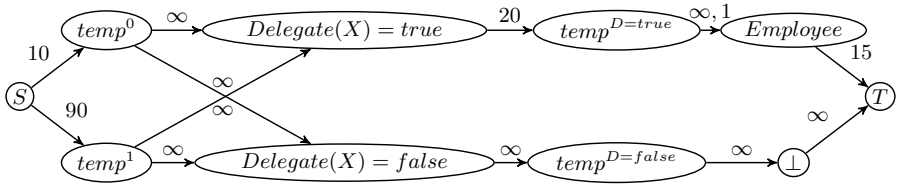

Fig. 2. Resource graph for our example

For the resource restriction, ReLiA obtains the name of the resource, B, the number of the resource used for executing that range value for one grounding, n, and the restriction of how often r can be selected, m. Then, ReLiA adds an edge from $A = r$ to $temp^r$ to R and assigns m as capacity to the edge. Next, ReLiA checks if there already exists a node B corresponding to this resource in R. In case there is no node B, ReLiA adds a node B to R as well as an edge from B to T with the corresponding resource restriction v of the model, $B = v$, as capacity. Having B in the model, ReLiA adds an edge between $temp^r$ and B while storing how many resources selecting the range value requires, i.e., storing n. When calculating the maximum flow, ReLiA has to multiply all values arriving at $temp^r$ with n to obtain the resources used at B from r. After ReLiA has added the nodes and edges for the resource restrictions and resources, it adds an edge between $temp^i$ and $A = r$ to R without any capacity limitations to connect this path to S over $temp^i$. In the last step, after having iterated over all resources, range values, and actions parfactors, ReLiA returns R.

Figure 2 shows the corresponding resource graph for our case study. We can see that our tasks are split into 2 groups. The first group with 10 tasks that are overdue corresponds to $temp^0$ and the other group with 90 tasks, where we have no additional information, corresponds to $temp^1$. For both groups, we have the very same action PRV, leading to both $temp^0$ and $temp^1$ being connected to the same range values without any capacity restrictions. Overall, delegating a task can be done at most 20 times as shown on the edge between $Delegate(X) = true$ and $Temp^{D=true}$. Further, delegating a task requires 1 employee, evident at the edge between $Temp^{D=true}$ and $Employee$. Additionally, there are 15 employees as depicted on the edge between $Employee$ and T. For not assigning a task, no restrictions apply, which can be seen in the lower part of the network with the paths going over $Delegate(X) = false$.

Based on such a graph, ReLiA computes all action assignments within the boundaries of the restrictions, which is a max-flow problem on such a graph.

Calculating Assignments of Maximum Flow To obtain all action assignments that lead to the maximum flow in a given resource graph, ReLiA has to iterate over all action PRVs and their corresponding range values. The rough idea is that for each action parfactor, ReLiA has to select paths from source to target. By combining all paths, ReLiA obtains valid action assignments. An assumption we make for ReLiA is that each action PRV always has an unrestricted default case. In the case study, the unrestricted default case in $Delegate(X) = false$, which has no further restrictions as denoted by the $\perp$ symbols.

Algorithm 3 outlines how ReLiA obtains assignments. Inputs are a resource graph R of a model G and the number num of action parfactors in G, i.e., $num = |G_a|$. First, ReLiA fills a list $\mathbf{n}$ of nodes to traverse with all $temp^i$ nodes, which correspond to action parfactors. Second, ReLiA calls a function named COMPILE with $\mathbf{n}$ and an empty list of paths, $\mathbf{p}$, to compute assignments.

In COMPILE, ReLiA gets the first node, $currentNode$, from $\mathbf{n}$ and computes all possible paths from S to T over $currentNode$ that assign an action to all instances of the action parfactor behind $currentNode$. The paths are constructed in a way that they always use as many instances as possible. For each path, ReLiA also has to obtain assignments for all other action parfactors. Therefore, ReLiA calls COMPILE again with the current path(s) and the remaining nodes to traverse. Last, when ReLiA has no more nodes to traverse, it obtains the assignments and the corresponding capacities by propagating the used capacities backwards from the target to the source. Overall, ReLiA roughly computes max^{num} action assignments, where max refers to the highest number of range values in any of the action PRVs and $num = |G_a|$ as above.

Let us explain Algorithm 3 in more detail by having a look at the resource graph in Fig. 2. ReLiA starts by adding $temp^0$ and $temp^1$ to $\mathbf{n}$. Then, ReLiA calls the helper function COMPILE with $\mathbf{n}$ and an empty list of paths $\mathbf{p}$. In COMPILE, $temp^0$ becomes $currentNode$. The capacity between S and $temp^0$ is 10. As $temp^0$ comes from the action parfactor with two values in the range of its action PRV, $temp^0$ has two assignments, $Delegate(X) = true$ and $Delegate(X) = false$, as

Algorithm 3 Obtaining All Assignments

```
function ObtainAssignments(Resource graph R, number of action parf. num)
    n empty list of nodes
    p empty list of paths
    i := 0
    for i < num do
        n = n + temp^i
    return compile(n, p)

function compile(Nodes to traverse n, current paths p)
    currentNode := pop n
    for each successor A = r of currentNode do
        tempP := path from S over currentNode and r to T
        p' := tempP + p
        if tempP uses all capacities of currentNode then
            if n not empty then
                return compile(n, p')
            else
                a := get all capacities and their assignments from p'
                return a
        else
            for each possible completion to use all capacities of currentNode do
                p' := p'+ completion path(s)
                if n not empty then
                    return compile(n, p')
                else
                    a := get all capacities and their assignments from p'
                    return a
```

successors. For the first assignment, $Delegate(X) = true$, ReLiA finds a path from S over $temp^0$ and $Delegate(X) = true$ to T. Thus, the path is added to $\mathbf{p}'$. That path is able to let all 10 instances of the capacity flow to T. Then, ReLiA again calls compile.

This time, $\mathbf{n}$ only contains $temp^1$ and $\mathbf{p}'$ contains one path. Now, $temp^1$ becomes $currentNode$, which again has two assignments as successors, over which ReLiA iterates. For the first assignment, $Delegate(X) = true$, ReLiA finds a path from S over $temp^1$ and $Delegate(X) = true$ to T. Thus, the path is added to $\mathbf{p}'$, leading to two paths being in $\mathbf{p}'$. This newly added path does not send all 90 instances, but only 5 since 5 is the remaining capacity on the last edge going into T. The remaining 85 instances can be send from S over $temp^1$ and $Delegate(X) = false$ to T, which then is added to $\mathbf{p}'$. In case there would be other paths to use all instances, ReLiA would have to iterate over them. Having no more nodes to travers, ReLiA computes the assignment for the paths in $\mathbf{p}'$. Here, ReLiA outputs $Delegate(X) = true$ for the 10 overdue tasks, $Delegate(X) = true$ for 5 tasks, where we have no additional information, and $Delegate(X) = false$ for the remaining 85 tasks.

We now jump back to the point where ReLiA iterates over all successors of $temp^1$ with $\mathbf{p}'$ containing only the path from S over $temp^0$ and $Delegate(X) = true$ to T sending 10 instances. The node $temp^1$ has another assignment, namely $Delegate(X) = false$, as successor and ReLiA finds a path from S over $temp^1$ and $Delegate(X) = false$ to T. Hence, that path is added to $\mathbf{p}'$ in addition to the one path from $temp^0$ in $\mathbf{p}$. That path uses all 90 instances, leading to an assignment of $Delegate(X) = true$ for the 10 overdue tasks and $Delegate(X) = false$ for 90 tasks, where we have no additional information.

ReLiA also traverses the path for $Delegate(X) = false$ for $temp^0$, for which ReLiA again has to traverse all paths from $temp^1$ as it is still contained in $\mathbf{n}$ at this point. Finally, ReLiA returns four action assignments to test for MEU. In addition to the two assignments above, the third assignment set reads $Delegate(X) = false$ for the 10 overdue tasks, $Delegate(X) = true$ for 15 tasks, where we have no additional information, and $Delegate(X) = false$ for the remaining 75 tasks. The forth assignment set contains $Delegate(X) = false$ for the 10 overdue tasks and $Delegate(X) = false$ for the 90 tasks, where we have no additional information. Hence, ReLiA computes the desired four action assignments with corresponding capacities that obey all restrictions.

Before we discuss theoretical aspects of ReLiA, we consider related work of lifted inference and relational decision support.

3.3 Related Work

We take a look at inference under uncertainty in relational models as well as relational decision support.

First-order probabilistic inference leverages relational aspects. For models with known domain size, it exploits symmetries in a model by handling indistinguishable instances with representatives, known as lifting [23]. Poole [23] introduces parametric factor graphs as relational models and proposes lifted variable elimination (LVE) as an exact inference algorithm on relational models. Other lifted inference algorithms include (i) the lifted junction tree algorithm (LJT) [6], (ii) first-order knowledge compilation [7], (iii) probabilistic theorem proving [11], and (iv) lifted belief propagation [1],

Nath and Domingos [18] introduce Markov logic decision networks (MLDNs), which are relational models with action and utility nodes. Nath and Domingos calculate approximate solutions to an MEU problem in a MLDN, grounding the model [20]. Another approach of Nath and Domingos includes unnecessary groundings [19]. Apsel and Brafman [3] propose an exact lifted solution to the MEU problem based on [18]. Gehrke et al. [10] extend LJT to meuLJT to solve MEU problems in PDecMs exactly while also supporting marginal queries.

Additional research focuses on sequential decision making by investigating first-order (PO)MDPs [25,15,26], which use lifting techniques from de Salvo Braz, Amir, and Roth [24]. In contrast to first-order POMDPs, which are solved offline using policy iteration, we propose to support online decision making, i.e., by solving an MEU problem. In this paper, we introduce resources and enable restriction actions to bring PDecMs closer to real-world applications.

3.4 Discussion

In this section, we discuss the assumption we make about restrictions as well as how to compute an MEU with the possible actions.

Assumptions about Restrictions An assumption we make is that each action parfactor has a default action, which is unrestricted. The implication of the assumption for ReLiA is that it has to iterate over all action parfactors one time. In case such an assumption would not hold, the difference would be that the order in which ReLiA iterates over the action parfactors could matter in the sense that a different iteration order could lead to other assignments for actions. Thus, without the assumption, ReLiA would need to iterate over each permutation of action parfactors to calculate the action assignments.

Extending our Case Study An interesting extension to our case study is to introduce hard tasks as a new logvar name with a corresponding PRVs and action parfactor. Assume that delegating a hard tasks requires two employees for execution. Although our model then has two different action PRVs and the two corresponding action parfactors require the same resource, ReLiA computes valid assignments given the number of employees available. While constructing the resource graph, ReLiA identifies that both action parfactors require the same type of resource, even though they concern different action PRVs. Thus, there only is one node for employees in the resource graph. While computing the assignments, ReLiA ensures that the capacity gets multiplied with 2 when taking the edge between the nodes $temp^{DelegateHardTask(Y)}$ and $Employee$ in the corresponding resource graph. Hence, the required resources to perform an action are accounted for. Similar to our case study so far, with splits due to evidence, both action parfactors pointing to the same resource poses no problem to ReLiA.

To complete our case study, we briefly describe how to obtain the best action for the action assignments next.

Computing an MEU Having the action assignments, any algorithm solving the lifted MEU problem, e.g., [10,3], can calculate best actions leading to the highest expected utility. The action assignments ReLiA computes are the actions to iterate over in Eq. (2). Thus, an algorithm solving the MEU problem does not have to generate (all) action assignments anymore. However, normally in a lifted MEU, a range value of an action is selected for each group of indistinguishable instances. The action assignments of ReLiA do not necessarily assign the same range value for all indistinguishable instances of a group as a result of the restrictions. In our case study, one assignment is $Delegate(X) = false$ for the 10 overdue tasks, $Delegate(X) = true$ for 15 tasks, where we have no additional information, and $Delegate(X) = false$ for the remaining 75 tasks. Thus, the tasks, for which we have no additional information, need to be split even further. As all instances are indistinguishable, it does not matter which instances are split off. Thus, an algorithm solving the MEU problem using the action assignments of ReLiA might need to split logvars before it can calculate the corresponding expected utility.

3.5 Theoretical Analysis

Let us now investigate the theoretical implications of ReLiA. Here, we focus on two points, namely whether always using the maximum capacity for paths to obtain assignments is reasonable as well as how ReLiA compares to calculating action assignments in a similar fashion for a ground model.

Fewest Possible Action Sets ReLiA only obtains assignments with the highest possible capacity given the restrictions, e.g., $Delegate(X) = true$ for the 10 overdue tasks. In theory, ReLiA could also compute all other assignments, e.g., $Delegate(X) = true$ for 9 overdue tasks and $Delegate(X) = false$ for the one remaining overdue task, $Delegate(X) = true$ for 8 and $Delegate(X) = false$ for 2, and so on. The reason why ReLiA only obtains assignments with the highest capacity and not also all other possible assignment lies within the semantics of PDecMs. Computing an expected utility involves adding up all utilities at the end. Assuming that the two action range values map to different potentials and keeping in mind that the instances within a group are indistinguishable, one of the range values leads to a higher expected utility than the other, which is true for all instances of that group. Thus, we only need to check assigning all instances either the one value or the other. As a consequence, ReLiA only has to obtain assignments with the highest possible capacity. Preferably, ReLiA assigns all instances the same action but with restrictions, groups may be split further to stay within the boundaries of the restrictions. Hence, by only obtaining assignments with the highest capacity, ReLiA provides reasonable assignments and highly reduces the number of action assignments to reason over.

Comparison to the Ground Case While computing action assignments, ReLiA uses the fact that instances are indistinguishable. Calculating such action assignments on a ground model, a corresponding algorithm could not exploit this fact. Without indistinguishable instances, such an algorithm would have to model actions for each instance. Each instance would be connected to the source with an edge having a capacity of 1. ReLiA iterates over all of these nodes and then their range values. As mentioned above, ReLiA roughly computes max^{num}, where num is the number of action parfactors after splitting. In a ground case the number of assignments would be max^{num}, where num is the number of groundings of action parfactors, which can be a huge number. In our example num would be 100, so even for boolean range values a ground algorithm would have to compute roughly 2^{100} action assignments. Hence, there is a combinatorial blow up as all permutations of actions would need to be tested. Further, while solving an MEU problem, there are many redundant calculations, which is infeasible for large enough numbers. Therefore, restricting resources and actions in a lifted case allows for a practical formalism.

4 Case Study: Enterprise Architecture Analysis

Johnson et al. [13] present an Enterprise Architecture (EA) analysis, extending propositional influence diagrams, which essentially are Bayesian networks with

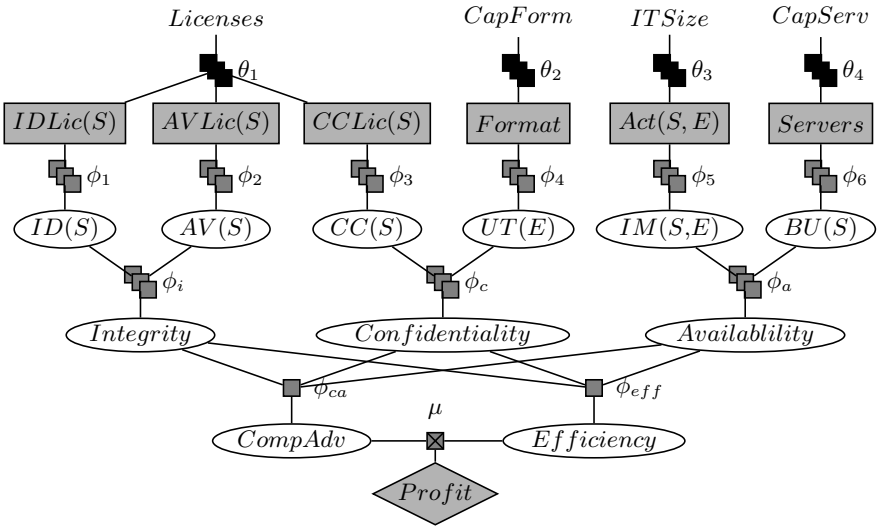

Fig. 3. A PDecM about IT security components (resources are represented as nodes without a border)

utility and action or decision nodes added. They do not consider the relational aspect, which blows up a propositional model if many components, processes, or employees are involved. Therefore, we can review EA analysis using PDecMs and consider what role ReLiA can play in this analysis. We take the case study in [13] and adapt it to the relational setting, which enables us to consider employees and work stations.

Figure 3 shows a PDecM for an EA scenario regarding IT security for a company where a decision maker has to set up an architecture for its IT security system. The model considers the following components of IT security as randvars: (i) intrusion detection applications (ID), (ii) anti-virus applications (AV), (iii) cryptographic control applications (CC), (iv) user training processes (UT), (v) incident management processes (IM), and (vi) back-up processes (BU). The randvars are parameterised with S for work stations and E for employees where appropriate. Regarding the applications, there are decisions to be made about the number of licenses to purchase, which may be limited. Regarding user training, the format of the training sessions is to be considered in terms of cost and number of people that can be trained at once. Regarding incident management, the number of people trained to handle incidents is limited. Regarding back-ups, the capacity of servers is limited. Overall, these components influence the integrity, confidentiality, and availability of a company, which in turn influence the competitive advantage as well as the efficiency emerging out of the decisions, which then influences the profit the company might make.

Given different scenarios for capacities, ReLiA computes the action assignments to consider for each scenario. Using these assignment sets, meuLJT solves

the corresponding MEU problems for each scenario, yielding one MEU assignment set for each scenario. The decision maker then can compare the results and incorporate further external factors in their final decision.

5 Conclusion

We introduce restrictions to PDecMs as a representation for models to support decision making. Restrictions are crucial as not every action can be performed as often as desired as well as resources are not limitless. We presnt ReLiA to compute all possible lifted assignments in restricted PDecMs. ReLiA computes all required action assignments, which an algorithm that solves the MEU problem in a lifted way can iterate over to identify best actions given restrictions. Further, ReLiA significantly reduces the assignment space to iterate over, making explicit that calculating assignments under restrictions is only feasible in lifted models.

Future work includes investigating whether actions can be learnt online, as for example Morgenstern does for agents using a specific logic [17]. Another interesting path would be to look into the situation calculus [16] and investigate whether additional restrictions can be included for decision support without adding another layer of logic on top. Additionally, we look into applying the presented theory to business process modelling. Johnson et al. [14] enabled us to show the usefulness of the theory in EA analysis for extended influence diagrams, going beyond propositional probabilistic models. With many instances, a lifted approach such as the presented one appears to be indispensable.

References

1. Ahmadi, B., Kersting, K., Mladenov, M., Natarajan, S.: Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. Machine learning **92**(1), 91–132 (2013)
2. Aoki, M.: Optimal control of partially observable Markovian systems. Journal of the Franklin Institute **280**(5), 367–386 (1965)
3. Apsel, U., Brafman, R.I.: Extended Lifted Inference with Joint Formulas. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence. pp. 11–18. AUAI Press (2011)
4. Åström, K.J.: Optimal Control of Markov Processes with Incomplete State Information. Journal of Mathematical Analysis and Applications **10**(1), 174–205 (1965)
5. Boyen, X., Koller, D.: Tractable Inference for Complex Stochastic Processes. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence. pp. 33–42. Morgan Kaufmann Publishers Inc. (1998)
6. Braun, T., Möller, R.: Lifted Junction Tree Algorithm. In: Proceedings of KI 2016: Advances in Artificial Intelligence. pp. 30–42. Springer (2016)
7. Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., De Raedt, L.: Lifted Probabilistic Inference by First-order Knowledge Compilation. In: IJCAI11 Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence. pp. 2178–2185. AAAI Press/International Joint Conferences on Artificial Intelligence (2011)

8. Ford, L.R., Fulkerson, D.R.: Maximal Flow through a Network. Canadian Journal of Mathematical **8**, 399–404 (1956)
9. Ford, L., Fulkerson, D.: Maximal flow through a network. Canadian Journal of Mathematics **8**, 399–404 (1956)
10. Gehrke, M., Braun, T., Möller, R., Waschkau, A., Strumann, C., Steinhäuser, J.: Lifted Maximum Expected Utility. In: Proceedings of Artificial Intelligence in Health. pp. 131–141. Springer (2019)
11. Gogate, V., Domingos, P.M.: Probabilistic Theorem Proving. In: UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011. pp. 256–265. AUAI Press (2011)
12. Harris, T., Ross, F.: Fundamentals of a method for evaluating rail net capacities. Tech. rep., RAND CORP SANTA MONICA CA (1955)
13. Johnson, P., Lagerström, R., Närman, P., Simonsson, M.: Enterprise Architecture Analysis with Extended Influence Diagrams. Information System Frontiers **9**, 163–180 (2007)
14. Johnson, P., Lagerström, R., Närman, P., Simonsson, M.: Enterprise Architecture Analysis with Extended Influence Diagrams. Information Systems Frontiers **9**(2-3), 163–180 (2007)
15. Joshi, S., Kersting, K., Khardon, R.: Generalized First Order Decision Diagrams for First Order Markov Decision Processes. In: IJCAI09 Proceedings of the 21st International Joint Conference on Artifical Intelligence. pp. 1916–1921. Morgan Kaufmann Publishers Inc. (2009)
16. McCarthy, J.: Situations, Actions, and Causal Laws. Tech. rep., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE (1963)
17. Morgenstern, L.: Knowledge Preconditions for Actions and Plans. In: Readings in Distributed Artificial Intelligence, pp. 192–199. Elsevier (1988)
18. Nath, A., Domingos, P.: A language for relational decision theory. In: International Workshop on Statistical Relational Learning (2009)
19. Nath, A., Domingos, P.: Efficient Lifting for Online Probabilistic Inference. In: Proceedings of the 6th AAAI Conference on Statistical Relational Artificial Intelligence, AAAIWS'10-06. pp. 1193–1198. AAAI Press (2010)
20. Nath, A., Domingos, P.M.: Efficient Belief Propagation for Utility Maximization and Repeated Inference. In: AAAI10 Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. pp. 1187–1192. AAAI Press (2010)
21. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press (1944)
22. Niepert, M., Van den Broeck, G.: Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In: AAAI14 Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. pp. 2467–2475. AAAI Press (2014)
23. Poole, D.: First-order probabilistic inference. In: IJCAI03 Proceedings of the 18th International Joint Conference on Artificial Intelligence. pp. 985–991. Morgan Kaufmann Publishers Inc. (2003)
24. de Salvo Braz, R., Amir, E., Roth, D.: Mpe and partial inversion in lifted probabilistic variable elimination. In: AAAI. pp. 1123–1130. AAAI Press (2006)
25. Sanner, S., Boutilier, C.: Approximate Solution Techniques for Factored First-order MDPs. In: 17th International Conference on Automated Planning and Scheduling. p. 288–295. AAAI Press (2007)
26. Sanner, S., Kersting, K.: Symbolic Dynamic Programming for First-order POMDPs. In: AAAI10 Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. pp. 1140–1146. AAAI Press (2010)