2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

A Particle Filter for Hybrid Relational Domains

Davide Nitti[1], Tinne De Laet[2], Luc De Raedt[3]

Abstract— **We introduce a probabilistic language and a fast inference algorithm for state estimation in hybrid dynamic relational domains with an unknown number of objects. More specifically, we apply Particle Filters to distributional clauses. The particles represent (partial) interpretations of possible worlds (with discrete and/or continuous variables) and the filter recursively updates its beliefs about the current state. We use backward reasoning to determine which facts should be included in the partial interpretations. Experiments show that our framework can outperform the classical particle filter and is promising for robotics applications.**

I. Introduction

Robotics research has made important achievements in problems such as state estimation, planning and learning. However the majority of probabilistic models used, such as Bayesian networks, cannot easily represent relational information, that is, objects, properties as well as the relations that hold between them. Relational representations allow one to encode more general models, to integrate background knowledge about the world and to convert low-level information into human-readable form. Probabilistic programming languages [1] and statistical relational learning techniques (SRL) [2] have been developed to provide such representations and have been successful in many application areas ranging from natural language processing to bioinformatics.

This paper extends probabilistic programming techniques to deal with dynamic relational domains involving both discrete and continuous random variables. Our approach employs a dynamic variation of distributional clauses [3], a recent extension of Sato's distribution semantics [4] to deal with continuous variables. Each state of the environment will be represented as an interpretation, that is, a set of ground facts that define a possible world. It is for dynamic distributional clauses that we develop a particle filter that allows one to recursively estimate the state over time given some observations. Particle filters [5] are widely applied in domains such as probabilistic robotics [6] and it is our goal to develop a particle filter that integrates relations and random variables in a flexible framework. The statistical relational learning literature already contains several approaches to temporal models and to particle filters; see Section VIII for a detailed discussion. However, few frameworks are suitable for robotics applications, indeed most of them support only discrete domains or are too slow for online applications.

Distinguishing features of our particle filter are that: 1) it provides a powerful relational template to define random variables and their distributions (discrete or continuous); 2) it exploits partial interpretations as particles allowing for a potentially infinite state space; and 3) it employs a relational representation to represent independence (context-specific) assumptions, and exploits that representation to perform fast inference; 4) it is suitable for robotics applications; and 5) it can integrate background knowledge and ontologies to perform complex queries about static or dynamic variables/relations.

This paper is organized as follows: we first review the particle filter (Section II-A) and introduce distributional clauses (Section II-B). Next we define dynamic distributional clauses (Section III) and use that representation to define a relational particle filter (Section IV). Then we introduce a magnetism scenario in Section V. After that we discuss inference optimizations (Section VI). Finally we show experiments (Section VII), related work (Section VIII), and conclude (Section IX).

II. Background

A. Particle Filters

The problem of filtering is concerned with the estimation of the current state of an agent in a dynamic environment where the world is not directly observable but only through observations obtained from sensors. Filtering is concerned with estimating the belief, that is, the probability density function $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$, where x_t is the current state, $z_{1:t}$ is the set of observations, and $u_{1:t}$ the actions (inputs) performed from time step 1 to t. The Bayes filter computes recursively the belief at time $t+1$, starting from the belief at t, the last observation z_{t+1}, and the last action performed u_{t+1} through $bel(x_{t+1}) = \eta p(z_{t+1}|x_{t+1}) \int p(x_{t+1}|x_t, u_{t+1}) bel(x_t) dx_t$, where η is a normalization constant. Thus the underlying model consists of a prior distribution $p(x_0)$, the state transition model $p(x_{t+1}|x_t, u_{t+1})$, and the measurement model $p(z_{t+1}|x_{t+1})$. Since the above integral is only tractable for specific combinations of distributions $bel(x_t)$, $p(x_{t+1}|x_t, u_{t+1})$, and $p(z_{t+1}|x_{t+1})$ (see for instance the Kalman filter [7]), one has to resort to approximations of the real belief, $bel(x_t)$. One solution is to resort to Monte-Carlo techniques for approximating the integral, resulting in the particle filter [5]. The key idea of particle filtering is to represent the belief by a set of weighted

*This work is supported by the European Community's 7th Framework Programme, grant agreement First-MM-248258.

[1] Davide Nitti is supported by the IWT (Agentschap voor Innovatie door Wetenschap en Technologie). Department of Computer Science, KU Leuven, Belgium. davide.nitti@cs.kuleuven.be

[2] Tinne De Laet is a Postdoctoral Fellow of the Fund for Scientific Research–Flanders (F.W.O.) in Belgium. Department of Mechanical Engineering, KU Leuven, Belgium. tinne.delaet@mech.kuleuven.be

[3] Department of Computer Science KU Leuven, Belgium. luc.deraedt@cs.kuleuven.be

samples (particles). Given N weighted samples $(x_t^{(i)}, w_t^{(i)})$ distributed as $bel(x_t)$, a new observation z_{t+1}, and a new action u_{t+1}, the particle filter generates a new weighted sample set that approximates $bel(x_{t+1})$. The algorithm proceeds in three steps:

- Sample a new set of particles $x_{t+1}^{(i)}$, $i = 1, \ldots, N$, from a proposal distribution $q(x_{t+1}|x_t^{(i)}, z_{t+1}, u_{t+1})$.
- Assign to each particle $x_{t+1}^{(i)}$ the weight: $w_{t+1}^{(i)} = \frac{p(z_{t+1}|x_{t+1}^{(i)})p(x_{t+1}^{(i)}|x_t^{(i)},u_{t+1})}{q(x_{t+1}^{(i)}|x_t^{(i)},z_{t+1},u_{t+1})}$
- If the variance of the sample weights exceeds a certain threshold, resample with replacement, from the particle set, with probability proportional to $w_{t+1}^{(i)}$.

A common simplification is the bootstrap filter where $q(x_{t+1}|x_t^{(i)}, z_{t+1}, u_{t+1}) = p(x_{t+1}|x_t^{(i)}, u_{t+1})$ and $w_{t+1}^{(i)} = p(z_{t+1}|x_{t+1}^{(i)})$.

B. Distributional Clauses

Our statistical relational representation for specifying the distributions of the particle filter is based on distributional clauses [3], an extension of the distribution semantics [4]. We assume some familiarity with standard terminology of statistical relational learning and logic programming [1]. We now introduce the key notions: A *clause*, in logic programming, is a first-order formula with a head, an atomic formula, and a body, a list of atomic formulas or their negation. For example, the clause

$$\texttt{inside}(\texttt{A},\texttt{B}) \leftarrow \texttt{inside}(\texttt{A},\texttt{C}), \texttt{inside}(\texttt{C},\texttt{B})$$

states that for all $\texttt{A},\texttt{B}$ and $\texttt{C}$, $\texttt{A}$ is inside $\texttt{B}$ if $\texttt{A}$ is inside $\texttt{C}$ and $\texttt{C}$ is inside $\texttt{B}$ (transitivity property). $\texttt{A},\texttt{B}$ and $\texttt{C}$ are logical variables. A ground atomic formula is a predicate applied to a list of terms that represents objects. For example, $\texttt{inside}(1,2)$ is a ground atomic formula, where $\texttt{inside}$ is a predicate, sometimes called relation, and $1, 2$ are symbols that refer to objects. A literal is an atomic formula or a negated atomic formula. A clause usually contains non-ground literals, that is, literals with logical variables (e.g., $\texttt{inside}(\texttt{A},\texttt{B})$). A substitution θ, applied to a clause or a formula, replaces the variables with other terms. For example, for $\theta = \{\texttt{A} = 1, \texttt{B} = 2, \texttt{C} = 3\}$ the above clause becomes:

$$\texttt{inside}(1,2) \leftarrow \texttt{inside}(1,3), \texttt{inside}(3,1)$$

and states that if $\texttt{inside}(1,3)$ and $\texttt{inside}(3,1)$ are true, then $\texttt{inside}(1,2)$ is true.

Most probabilistic relational languages consider only boolean random variables. In contrast, distributional clauses allow one to define random variables with any distribution, continuous or discrete. Moreover, a ground atom (i.e., tuple of a relation) represents a random variable. Therefore, throughout the paper, ground atoms and random variables are considered interchangeable. Formally, a *distributional clause* is a formula of the form $\texttt{h} \sim \mathcal{D} \leftarrow \texttt{b}_1, \ldots, \texttt{b}_\texttt{n}$, where the $\texttt{b}_\texttt{i}$ are literals and $\sim$ is a binary predicate written in infix notation. The intended meaning of a distributional clause is that each ground instance of the clause $(\texttt{h} \sim \mathcal{D} \leftarrow \texttt{b}_1, \ldots, \texttt{b}_\texttt{n})\theta$ defines the random variable $\texttt{h}\theta$ as being distributed according to $\mathcal{D}\theta$ whenever all the $\texttt{b}_\texttt{i}\theta$ hold, where θ is a substitution.

The term $\mathcal{D}$, that represents the distribution, can be non-ground, i.e., values, probabilities, or distribution parameters can be related to conditions in the body. A distributional clause is a powerful template to define conditional probabilities: the random variable h has a distribution $\mathcal{D}$ given the conditions in the body $\texttt{b}_1, \ldots, \texttt{b}_\texttt{n}$ (referred to also as $\texttt{body}$).

Furthermore, a term $\simeq(d)$ constructed from the reserved functor $\simeq/1$ represents the value of the random variable d. Consider the following clauses:

$$\texttt{n} \sim \texttt{poisson}(6). \quad (1)$$

$$\texttt{pos}(\texttt{P}) \sim \texttt{uniform}(1,10) \leftarrow \texttt{between}(1, \simeq(\texttt{n}), \texttt{P}). \quad (2)$$

Clause (1) states that the number of people $\texttt{n}$ is governed by a Poisson distribution with mean 6; clause (2) models the position $\texttt{pos}(\texttt{P})$ as a random variable uniformly distributed from 1 to 10, for each person $\texttt{P}$ such that $\texttt{between}(1, \simeq(\texttt{n}), \texttt{P})$ succeeds. Thus if the outcome of $\texttt{n}$ is 2, there will be 2 independent random variables $\texttt{pos}(1)$ and $\texttt{pos}(2)$.

This formalism allows to specify continuous distributions (under reasonable conditions) and naturally copes with an unknown number of objects (as BLOG [8] does). In addition to distributional clauses, we shall also employ definite clauses as in Prolog. We shall often talk about clauses when the context is clear. A distributional program, that is, a set of distributional clauses, defines a distribution over possible worlds. The procedure used to generate possible worlds defines the semantics and a basic inference algorithm. A possible world is generated starting from the empty set $S = \emptyset$; then for each distributional clause $\texttt{h} \sim \mathcal{D} \leftarrow \texttt{b}_1, ..., \texttt{b}_\texttt{n}$, whenever the body $\{\texttt{b}_1\theta, ..., \texttt{b}_\texttt{n}\theta\}$ is true in the set S for the substitution θ, a value v for the random variable $\texttt{h}\theta$ is sampled from the distribution $\mathcal{D}\theta$ and added to the set S. This is repeated until a fixpoint is reached, that is, no more variables can be sampled. The probability of a query given evidence $p(q|e)$ is estimated as the number of worlds sampled that satisfy the query and evidence divided by the number of worlds that satisfy the evidence. The described inference algorithm is generally inefficient, though several improvements are possible. The inference proposed by Gutmann et al. [3] uses magic sets [9] to generate only facts relevant to the query. In this paper we propose a more efficient inference algorithm based on backward reasoning (section VI) and focus on filtering in dynamic models (section III and IV).

III. Dynamic Distributional Clauses

We now extend distributional clauses to Dynamic Distributional Clauses (DDC) for modeling dynamic domains.

In DDCs, each predicate/variable is classified as state x, action u, or observation z, with a subscript that refers to time 0, for the initial step; time t for the current step, and $t + 1$ for the next step. The definition of a discrete-time stochastic process follows the same idea of a Dynamic Bayesian Network. We need clauses that define: 1) the prior distribution: $\texttt{h}_0 \sim \mathcal{D} \leftarrow \texttt{body}_0$, 2) the state transition model: $\texttt{h}_{\texttt{t}+1} \sim \mathcal{D} \leftarrow \texttt{body}_\texttt{t}$, 3) the measurement probability:

$\mathtt{h}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_{t+1}$, and finally, 4) clauses that define a random variable at time t from other variables at the same time: $\mathtt{h}_t \sim \mathcal{D} \leftarrow \mathtt{body}_t$. As these are all essentially distributional clauses, the semantics is retained from [3]. Details and formal definitions can be found in [3].

IV. A Particle Filter For DDC

We assume that the (dynamic) distributional clauses define the required distributions for a particle filter, i.e., the prior distribution, the proposal distribution and the weighting function. Furthermore, the particles $x_t^{(i)}$ will be interpretations, i.e., sets of ground facts for the predicates and the values of random variables that hold at timepoint t.

The basic relational particle filter applies the same steps as the classical particle filter sketched in Section II-A and employs the forward reasoning procedure for distributional clauses sketched in Section II-B. An optimized inference algorithm is presented in Section VI. In addition, multiple observations are supported, where the total weight is the product of the weight of each observation, assuming conditional independence of the observations given the state. This is not a real restriction on the expressiveness of the language as multiple observations could be taken together as a single observation. For resampling we use a minimum variance sampling algorithm [10]. The resulting framework is called Distributional Clauses Particle Filter (DCPF).

V. Magnetism experiment

To illustrate DDCs, we consider the following scenario. There is a table with several objects that can be either permanent magnets, ferromagnetic, or non-magnetic objects. The objects are marked so that their position and orientation can be easily recognized. In the experiments, pairs of objects are placed close to one another by a human or a robot, while keeping one object fixed in the hand. Then, the resulting interactions are observed. From such observations people are able to infer which objects are magnets, ferromagnetic or possibly non-magnetic. For instance, if the free object attracts the one in the hand, we know that at least one of them must be a magnet, and the other a magnet or a ferromagnetic object. The goal is now to realize this type of reasoning with our particle filter.

We modelled the following physical principles: 1) non-magnetic objects do not interact with any other objects, 2) a magnet and a ferromagnetic object attract each other with a force that depends on the distance, 3) two magnets can attract or repulse each other 4) the magnetic force produces a movement if it is greater than the static friction of the table, 5) an object moves if it is pushed by another object or it is grasped by a robot or human; 6) if an object is held by a gripper or hand, its movement is independent of the eventual magnetic force.

We perform two types of actions: 1) putting an object near another one (while keeping the first object in the hand), 2) pushing an object until it makes contact with another one, and then pulling it away. After performing an action the robot or human pulls the objects apart again (if they attracted one another) and then takes another object and repeats the experiment. The goal is to estimate interactions and therefore objects' types from observed objects' positions during the experiments. The objects' types are correlated to one another, therefore whenever we perform an experiment on two objects this may affect the belief of objects tested previously. The magnetism theory has been encoded in the state transition model, while the measurement model consists of the product of Gaussian distributions around each object's position $\mathtt{pos}_t(\mathtt{i})$ (assuming that the measurements are i.i.d.):

$$\mathtt{obsPos}_{t+1}(\mathtt{ID}) \sim \mathtt{gaussian}(\simeq(\mathtt{pos}(\mathtt{ID})_{t+1}),\, cov). \quad (3)$$

The hidden state consists of the objects' type, position, static friction and relations amongst them such as $\mathtt{near}(\mathtt{A},\mathtt{B})_t$, $\mathtt{interaction}(\mathtt{A},\mathtt{B})_t$ and $\mathtt{contact}(\mathtt{A},\mathtt{B})_t$. Other properties are included, such as the objects' dimensions and colors. The number of objects is not defined in advance, on the contrary, the state grows every time an object is added. Relation $\mathtt{near}(\mathtt{A},\mathtt{B})_t$ is true if the distance of objects A and B is lower than a threshold (only then the objects A and B can interact):

$$\mathtt{near}(\mathtt{A},\mathtt{B})_t \leftarrow \mathtt{dist}(\simeq(\mathtt{pos}(\mathtt{A})_t), \simeq(\mathtt{pos}(\mathtt{B})_t)) < 0.11. \quad (4)$$

The relation $\mathtt{interaction}(\mathtt{A},\mathtt{B})_t$ is 1 if there is an attraction between the objects, and -1 if there is a repulsion. For example, for two magnets, interaction (with equal probability 0.5 for attraction/repulsion) is defined as follows:

$$\mathtt{interaction}(\mathtt{A},\mathtt{B})_t \sim \mathtt{finite}([0.5{:}1, 0.5{:}-1]) \leftarrow \mathtt{A} \neq \mathtt{B}, \simeq(\mathtt{type}(\mathtt{A})_t) = \mathtt{magnet}, \simeq(\mathtt{type}(\mathtt{B})_t) = \mathtt{magnet}. \quad (5)$$

For simplicity, we also assume that the interaction is constant over time: if two magnets attract each other, they will maintain that interaction, ignoring the possible repulsion by rotation of the magnet:

$$\mathtt{interaction}(\mathtt{A},\mathtt{B})_{t+1} \sim (\simeq(\mathtt{interaction}(\mathtt{A},\mathtt{B})_t)) \quad (6)$$

that is, $\mathtt{interaction}(\mathtt{A},\mathtt{B})_{t+1} = \mathtt{interaction}(\mathtt{A},\mathtt{B})_t$. However, orientation could be integrated to distinguish attraction and repulsion from objects' orientation. In addition, the relation $\mathtt{contact}(\mathtt{A},\mathtt{B})_t$ is true if A and B are in contact or overlapping:

$$\mathtt{contact}(\mathtt{A},\mathtt{B})_t \leftarrow \mathtt{dist}(\simeq(\mathtt{pos}(\mathtt{A})_t), \simeq(\mathtt{pos}(\mathtt{B})_t)) < (\mathtt{dim}(\mathtt{A}) + \mathtt{dim}(\mathtt{B}))/2 + \theta$$

i.e., $\mathtt{contact}(\mathtt{A},\mathtt{B})_t$ is true if the center's distance is smaller than half of the sum of their dimensions plus a threshold θ. This qualitative definition assumes that the objects are spherical, but it is sufficient for our purposes. To correct objects overlapping, the distance between them is increased in the next step if below the limit. Since the objects' type distributions do not change over time we can write:

$$\mathtt{type}(\mathtt{A})_t \sim \mathtt{finite}([1/3{:}\mathtt{magnet}, 1/3{:}\mathtt{ferromagnetic}, 1/3{:}\mathtt{nonmagnetic}]) \leftarrow \mathtt{object}(\mathtt{A}). \quad (7)$$

This special clause (7) directly specifies the type belief of those variables $\mathtt{type}(\mathtt{A})_t$ that do not yet occur in the particle

and hence, have not yet been sampled. Same considerations hold for clause (5). Whenever an object type needs to be evaluated and is not in the particle, it will be sampled from that distribution. Sampled variables $\texttt{type(A)}_\texttt{t}$ remain constant over time, as described for $\texttt{interaction(A,B)}_\texttt{t}$ in clause (6). Although we can define clauses that specify a belief distribution that changes over time.

The state transition model encodes the principles listed above. For example, the next object's position is the current position plus Gaussian noise if there are no objects nearby. If two objects are close enough, and if they are magnet-ferromagnetic or magnet-magnet they will attract one another till contact, whenever the magnetic force overcomes the static friction with the table. In addition, we assume that the attraction brings the objects in contact in one step.

$$\begin{aligned}&\texttt{pos(A)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{middlepoint(A,B)}, cov) \leftarrow \\ &\texttt{near(A,B)}_\texttt{t}, \texttt{not}(\texttt{action}(\texttt{move}(\texttt{A},\_)), \\ &\texttt{not}(\texttt{action}(\texttt{move}(\texttt{B},\_)), \simeq(\texttt{interaction(A,B)}_\texttt{t}) = \texttt{1}, \\ &c{\cdot}\texttt{dist}(\simeq(\texttt{pos(A)}_\texttt{t}), \simeq(\texttt{pos(B)}_\texttt{t}))^{-2} > \simeq(\texttt{friction(A)}_\texttt{t}) \quad (8)\end{aligned}$$

This clause defines the next position of an object A when it is not moved or held by a gripper or human, that is $\texttt{not}(\texttt{action}(\texttt{move}(\texttt{A},\_))$, and if it is near to another object B with which there is an attraction force that overcomes the static friction with the table. The next position is determined by a Gaussian around the point in the middle of A and B considering the object's size to avoid overlap, computed by the function $\texttt{middlepoint(A,B)}$. The magnetic force is $c \cdot \texttt{dist}(\simeq(\texttt{pos(A)}_\texttt{t}), \simeq(\texttt{pos(B)}_\texttt{t}))^{-2}$, where c is a constant. Moreover, the static friction is randomly sampled from a Gaussian independently every step. The definition of middlepoint and friction are omitted for brevity.

VI. Optimized Inference

The classical particle filter samples all variables and generates *complete* interpretations. For the magnetism scenario this requires computing all relationships that hold for each pair of objects in a particle. This is often unrealistic and undesirable; it may also lead to wasted resources and bad performance. Thus, we work with particles that are partial interpretations. For instance, in the magnetism scenario, if two objects are far apart, the state in the particles will consist of the positions of both objects. Any other state variable is marginalized, e.g., type, interaction, contact, and friction. Indeed if the objects are far apart the relation near is false and there are no interactions, therefore we do not need to sample the object type or other relations. Whenever such a variable is needed during evaluation it is sampled from the relevant distribution. For example, consider the clause (8) for objects 1 and 2, if $\texttt{near(1,2)}_\texttt{t}$ is true and there are no action moves for 1 and 2, then $\texttt{interaction(1,2)}_\texttt{t}$ needs to be evaluated. If that random variable is not already in the particle, clauses such as (5) are evaluated to sample it. Those clauses check the object's type, if not already in the particle it will be sampled from (7). Hence, distributional clauses allow to define a template of context-specific independence assumptions with a relational representation.

A. Mathematical Formalization

More formally, we assume that the complete interpretation at time t can be written as $x_t = x_t^{PI} \cup x_t^a$. Here x_t^{PI} denotes the variables in the partial interpretation and x_t^a the remaining variables, i.e., the marginalized variables. The partial interpretation needs to remain finite, however the number of marginalized variables can be countably infinite. To define a proper distribution, the distributional program has to satisfy Sato's finite support condition [4], that is, for each random variable there is finite number of explanations, where each explanation is a finite conjunction. In other words, each random variable can be derived from a finite number of other random variables and there is a finite number of ways to derive such a variable, i.e., there is a finite number of proofs (explanations). Therefore, even though an infinite number of random variables can be derived/sampled from x_t^{PI}, each variable of interest can be derived from a finite set of other variables. The marginalized variables x_t^a will only be non-empty if the belief of the entire state can be derived from the partial interpretation x_t^{PI}:

$$\begin{aligned}&bel(x_t) = p(x_t^a, x_t^{PI} | z_{1:t}, u_{1:t}) = \\ &= p(x_t^a | x_t^{PI}, z_{1:t}, u_{1:t}) p(x_t^{PI} | z_{1:t}, u_{1:t}) = p(x_t^a | x_t^{PI}) bel(x_t^{PI}).\end{aligned}$$

This assumption allows to apply the Bayes Filter to x_t^{PI}. The variable x_t^a represents what we can derive from x_t^{PI} together with the probabilistic rules defined in the model. For example, if $\texttt{near(a,b)}_\texttt{t}$ is marginalized, clause (4) can be used to sample it given x_t^{PI}, that is $\texttt{pos(a)}_\texttt{t}$ and $\texttt{pos(b)}_\texttt{t}$. Clause (7) is a special case with $p(x_t^a | x_t^{PI}) = p(x_t^a) = p(x_{t-1}^a) = \ldots = p(x_0^a)$ for variables $\texttt{type(A)}_\texttt{t}$ not yet sampled.

In a particle filter we have to compute, for each particle, the **smallest** partial interpretation needed to sample the next state and evaluate the weight of the particle. This means that for each step t and each particle i the set $x_{t+1}^{PI(i)}$ and $x_t^{PI(i)}$ must satisfy the following conditions:

1) the variables in the partial interpretation $x_{t+1}^{PI(i)}$ do not depend on the marginalized variables: $p(x_{t+1}^{PI(i)} | x_t^{PI(i)}, x_t^a, x_{t+1}^a, u_{t+1}) = p(x_{t+1}^{PI(i)} | x_t^{PI(i)}, u_{t+1})$;
2) the weighting function does not depend on the marginalized variables: $p(z_{t+1} | x_{t+1}^{(i)}) = p(z_{t+1} | x_{t+1}^{PI(i)})$
3) the marginalized variables x_{t+1}^a depend only on the partial state at time $t+1$: $p(x_{t+1}^a | x_{t+1}^{PI(i)}, x_t^{PI(i)}, x_t^a, x_{t+1}^a, u_{t+1}) = p(x_{t+1}^a | x_{t+1}^{PI(i)})$ unequivocally described by clauses

Under these assumptions, we can apply the particle filter steps without sampling the marginalized variables. Initially, a particle i at time t consists of a partial state $x_t^{PI(i)}$ together with marginalized x_t^a; the next x_{t+1} is not sampled yet, therefore $x_{t+1}^a = x_{t+1}$ and $x_{t+1}^{PI} = \emptyset$ (Figure 1 left). Given a new observation z_{t+1}, the algorithm performs the particle filtering steps sampling marginalized variables in x_t^a and x_{t+1}^a (thus expanding $x_t^{PI(i)}$ and $x_{t+1}^{PI(i)}$) until the above

conditions are satisfied, as shown on the right of Figure 1. Note that the set of variables in $x_{t+1}^{PI(i)}$ can be different from the set in $x_t^{PI(i)}$, and can differ between particles.

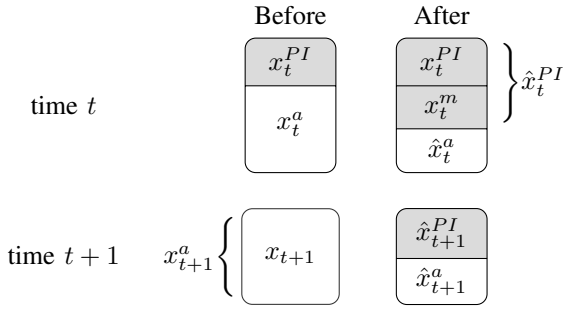


Fig. 1: Particle partition, before (left) and after (right) the filtering algorithm. Initially x_{t+1} is not sampled, therefore $x_{t+1}^a = x_{t+1}$ and $x_{t+1}^{PI} = \emptyset$. The inference algorithm samples variables $x_t^m \subseteq x_t^a$, $x_{t+1}^m \subseteq x_{t+1}^a$ and adds them respectively to x_t^{PI} and x_{t+1}^{PI}. Indeed, $\hat{x}_t^{PI} = x_t^{PI} \cup x_t^m$, $\hat{x}_t^a = x_t^a \setminus x_t^m$, $\hat{x}_{t+1}^{PI} = x_{t+1}^{PI} \cup x_{t+1}^m = x_{t+1}^m$, $\hat{x}_{t+1}^a = x_{t+1}^a \setminus x_{t+1}^m$.

This method is a special case of Rao-Blackwellisation [11], where the state space is partitioned into one set of variables that is sampled and another one that is calculated analytically, as shown in Figure 1. The decision of which variables to sample and which ones to marginalize is made automatically according to the model and the specific interpretation, as explained in the following section.

B. Inference Algorithm

The inference algorithm is based on tabled backward reasoning (SLD-resolution) extended with a Monte-Carlo mechanism similar to that employed in ProbLog [12]. This differs from the algorithm used for non-temporal distributional clauses [3]. Tabling [13], a well-known logic programming technique, is applied to memorize computed goals, including variables and their distribution $\mathtt{h} \sim \mathcal{D}$. The tables are separate for each particle and are reset every step, since the variables/predicates are different over time and between particles. Tabling avoids that the same goal is computed multiple times and avoids infinite loops in recursive calls.

1) Query Evaluation: We can estimate the belief of a query q at time t as:

$$bel_t(q) \approx \sum_{i=1}^{N} p(q|x_t^{PI(i)}) w_t^{(i)},$$

where $x_t^{PI(i)}$ is a particle and $w_t^{(i)}$ its weight. If the query involves variables V_q in the particle, i.e., $V_q \subset x_t^{PI(i)}$, then $p(q|x_t^{PI(i)})$ is 1 if q complies with the values of $x_t^{PI(i)}$ and 0 otherwise. If $V_q \not\subset x_t^{PI(i)}$, the algorithm looks for distributional clauses that define the distribution of each $v_i \in V_q$ to compute $p(q|x_t^{PI(i)})$. The variables v_i are forced to take values that make the query true, following a principle similar to likelihood weighting. For each variable v_i the algorithm looks for a clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{body}$ such that $\mathtt{h}\theta = v_i$ for a substitution θ, and evaluates the body recursively; if $\mathtt{body}\theta$ holds, we can compute the likelihood of v_i (that complies with the query q) from v_i's distribution $\mathcal{D}\theta$. Then $p(q|x_t^{PI(i)})$ is the product of the likelihood of each variable v_i. The evaluation of distributional clauses may require sampling **new** random variables expanding the particles.

2) Filtering Algorithm: The weighting and prediction steps query the following logical goals in each particle i:

Step (1): the weight of the given observation $z_{t+1} = val$
Step (2): the next state x_{t+1}.

To explain the algorithm let us consider the bootstrap filter.

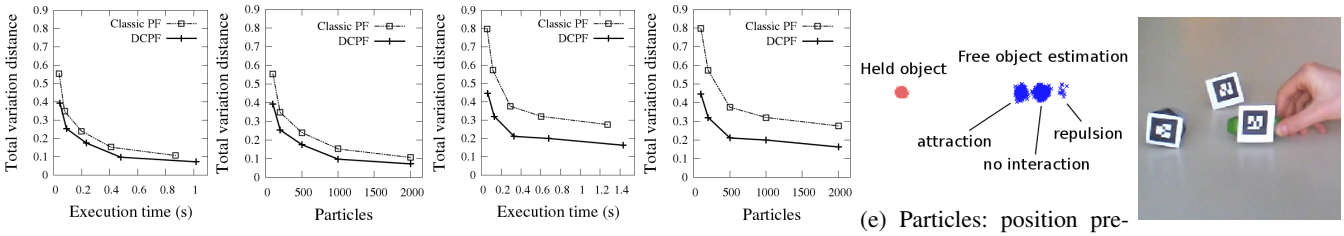
Step (1) performs the weighting step and implicitly (part of) the prediction step: it computes the weight $w_{t+1}^{(i)} = p(z_{t+1}|x_{t+1}^{PI(i)})$ using the same algorithm described to evaluate $p(q|x_t^{PI(i)})$, where $q = (z_{t+1} = val)$. This process will automatically sample those variables at time $t+1$ and t that are needed to evaluate the weight starting from the particle $x_t^{PI(i)}$ (but this process never goes back to time $t-1$).

Step (2) performs the prediction step for variables that have not yet been sampled because they are not involved in the weighting step. The algorithm looks for any DC clause whose head is related to the next state x_{t+1} (state transition model), and then evaluates the body recursively. Whenever the body is true the variable/relation in the head will be sampled and added to the particle.

To understand the described algorithm let us consider the weight evaluation (Step (1)) for the observation $\mathtt{obsPos}(1)_{\mathtt{t+1}} = value$ related to object 1. The algorithm searches the definition of $\mathtt{obsPos}(1)_{\mathtt{t+1}}$, given by clause (3). This clause needs $\mathtt{pos}(1)_{\mathtt{t+1}}$, that is the next position of object 1; therefore the algorithm tests clause (8) that defines $\mathtt{pos}(1)_{\mathtt{t+1}}$ with substitution $\mathtt{A} = 1$. If $\mathtt{near}(1,\mathtt{B})_{\mathtt{t}}$ is false for any object B, the body of (8) is not true, and the inference algorithm will backtrack and search for another clause that defines $\mathtt{pos}(1)_{\mathtt{t+1}}$. In this situation, the remaining conditions in the body of (8) are not tested, such as $\mathtt{interaction}(1,\mathtt{B})_{\mathtt{t}}$, for any B. If the set of random variables $\mathtt{interaction}(1,\mathtt{B})_{\mathtt{t}}$ is never tested in the entire filtering step, then $\mathtt{pos}(1)_{\mathtt{t+1}}$ is independent of $\mathtt{interaction}(1,\mathtt{B})_{\mathtt{t}}$ given the particle, therefore the variables $\mathtt{interaction}(1,\mathtt{B})_{\mathtt{t}}$ are not sampled. On the other hand, if $\mathtt{near}(1,\mathtt{B})_{\mathtt{t}}, \mathtt{not}(\mathtt{action}(\mathtt{move}(1,\_))$ and $\mathtt{not}(\mathtt{action}(\mathtt{move}(\mathtt{B},\_))$ are true for a particular B, then $\mathtt{interaction}(1,\mathtt{B})_{\mathtt{t}}$ needs to be evaluated, hence the mentioned independence is context-specific.

Step (2) samples random variables $x_{t+1}^m \subseteq x_{t+1}^a$ that depend on the previous partial state $x_t^{PI(i)}$ to satisfy condition 3 in Section VI-A. For example, in the magnetism model we impose that the interaction type remains constant for each pair of objects (clause (6)). The algorithm will (deterministically) sample $\mathtt{interaction}(\mathtt{A},\mathtt{B})_{\mathtt{t+1}} = \mathtt{interaction}(\mathtt{A},\mathtt{B})_{\mathtt{t}}$ only for A, B such that $\mathtt{interaction}(\mathtt{A},\mathtt{B})_{\mathtt{t}} \in x_t^{PI(i)}$.

On the contrary, Step (2) does not sample variables x_{t+1}^a that depend only on the partial state $x_{t+1}^{PI(i)}$. For example, clause (4) is not evaluated for time $t+1$, therefore variables $\mathtt{near}(\mathtt{A},\mathtt{B})_{\mathtt{t+1}}$ are not sampled, at least in the current particle



(a) 3 objects (b) 3 objects (c) 4 objects (d) 4 objects (e) Particles: position prediction (f) Experiment

Fig. 2: (a-d) Total variation distance between the ideal type distribution and the estimation provided by the DCPF and classical PF for a predefined set of observations; the classical particle filter sample the entire state, while DCPF keeps partial particles. Performance improvement increases with the number of objects (e) Objects' position prediction when close to another object; (f) experimental setup

filter step $t \rightarrow t+1$. After Step (2) any marginalized random variable at time $t+1$ must be derivable from $x_{t+1}^{PI(i)}$ together with the distributional program. This condition avoids the need to go back in time during filtering or query evaluation, thus previous partial states $x_{0:t}^{PI(i)}$ can be forgotten. However, it may require to perform belief update for an unknown number of marginalized variables. This issue is solved using clauses that directly define the belief distribution over time for some of the marginalized random variables. For example, (5) and (7) describe the distribution of not yet sampled variables $\texttt{interaction(A,B)}_\texttt{t}$ and $\texttt{type(C)}_\texttt{t}$ respectively. Alternatively, under some constraints, lifted inference might be used to perform belief update of a set of variables together, but this solution is not investigated in this paper.

The described algorithm performs forward reasoning because it samples the next state starting from the current state even though it is based on backward reasoning to determine state variables relevant to belief update. If we apply a naive forward reasoning algorithm instead, it would generate a large or even infinite number of variables. Our algorithm generates partial interpretations and any marginalized variable can be derived from the partial interpretation using the same inference procedure. After the belief update, the particle will be partitioned as depicted in Figure 1, satisfying the conditions described in Section VI-A.

VII. Experiments

This section answers the following questions:

(Q1) Does the algorithm obtain the correct results?
(Q2) How do the DCPF and the classical particle filter compare?
(Q3) Is the DCPF suitable for real-world applications?

We compared two algorithms: the classical particle filter and our DCPF. The classical particle filter samples the entire state every step with a forward reasoning procedure. All algorithms were implemented in YAP Prolog and run on an Intel Core i5 3.3GHz for simulations and on a laptop Core i7 for the real-world experiment. For the evaluation of the algorithm we used the magnetism scenario.

In the first experiment we tested the correctness of DCPF (Q1) using a predefined sequence of actions and observations and compared the results with the expected outcome. In particular we simulated the observations of three objects a, b, c and the following three interactions in order: attraction of pairs (a,b) and (b,c), absence of interaction for (a,c). Assuming those interactions known, we can infer that the objects a, b, and c are respectively ferromagnetic, magnet and ferromagnetic; other combinations are not possible. The DCPF with the model described in section V converges to the expected solution for a predefined sequence of observations that simulates those interactions. In addition, we computed the three object type distribution given attraction (a,b) and (b,c). Then we computed the four objects' type distributions given attraction (a,b) and (b,c) and absence of interaction between (a,d), where d is another object. Therefore, we compared those exact distributions with the DCPF and classical particle filter estimations after a set of observations that simulate these interactions. It is clear that the type distributions are analytically computed in a simplified model where we assume the object interactions to be known. Therefore, the DCPF predicted distribution is expected to be affected by an error with respect to the ideal case. Indeed, the interactions are estimated only from the object positions. To measure the error between the predicted and the exact posteriors we use the total variation divergence (i.e., the sum of absolute differences). Figure 2 shows that both algorithms converge to the correct results (Q1), but our DCPF produces lower errors when compared to the classical particle filter (Q2) for the same number of particles (figures 2b, 2d) and for the same execution time (figures 2a, 2c). This is because DCPF samples only variables relevant for belief update, improving time performance. This speed improvement is more evident for a growing number of objects, as we can see comparing figures 2a with 2c and 2b with 2d. Indeed, the state space becomes larger, thus the computational difference between filtering the entire state and filtering a partial state increases. In addition, DCPF performs a kind of Rao-Blackwellization marginalizing a subset of variables, therefore we obtain a reduction of the total variation distance.

An optimized propositional particle filter can be faster than our DCPF when the state space is relatively small. However, the state space becomes larger and larger for an increasing number of objects and relations between them. In addition, general knowledge representation about the world

and objects is less straightforward in a propositional (non-relational) model. Furthermore, additional problems arise for a heterogeneous state space, where each possible world can contain different variables. In contrast, in distributional clauses and its dynamic extension we can easily integrate probabilistic as well as deterministic background knowledge exploiting the relational representation. For example we can integrate ontologies about objects, their properties and relations between them. The described inference algorithm will query that knowledge base whenever required, keeping acceptable performance.

We ran similar experiments with real-world data (Q3): we used magnets, iron-made and nonmagnetic objects with markers on them for an easy detection with a camera (figure 2f). We tested pairs of objects, as described above, putting an object near the other. Since the displacement can be small we pull one object after the attraction, hence the other object remains attached proving further evidence of attraction. The estimated distribution in this scenario converges to the ideal distribution up to a small but inevitable error. In the end, we continued the experiment adding new objects and testing them with previous objects.

As described before, a relational representation allows to perform complex queries exploiting a background knowledge. The queries can refer to static knowledge or dynamic variables updated in the particle filter. In the magnetism experiment, we added the definition of relations $\texttt{rightOf}(\texttt{A},\texttt{B})_\texttt{t}$, $\texttt{leftOf}(\texttt{A},\texttt{B})_\texttt{t}$ and $\texttt{color}(\texttt{A})$. Therefore, we can perform simple queries such as $\texttt{type}(1) = \texttt{magnet}$ to compute the probability that 1 is a magnet, or more complex queries such as $(\texttt{type}(\texttt{B}) = \texttt{magnet}, \texttt{rightOf}(\texttt{A},\texttt{B})_\texttt{t}, \texttt{color}(\texttt{A},\texttt{green}))$, to list all pairs $(\texttt{A},\texttt{B})$ such that B is a magnet with a green object A on the right; for each pair $(\texttt{A},\texttt{B})$ a probability that the query is true is provided. A video of the experiment is available at `https://dtai.cs.kuleuven.be/ml/systems/DC/dcpf.html`.

VIII. Related Work

A. Theoretical works

Our particle filter is related to probabilistic programming languages such as BLOG [8], Church [14], ProbLog [12], and the distributional clauses of [3]. While these languages are expressive enough to be used for modeling dynamic relational domains, these languages do not support filtering, which makes inference prohibitely slow for dynamic models.

There exist SRL approaches for temporal models. A variant of BLOG for dynamic domains and filtering has been proposed (but no experimental results are available in the literature). Logical HMMs [15] employ logical atoms as observations and states and hence, their expressivity is more limited. The lifted relational Kalman filter [16], performs efficient lifted exact inference for continuous dynamic domains but assumes Gaussian models. The relational particle filter of [17] and CPT-L [18] cannot handle partial particles. Finally, the approaches that are most similar to ours are those of [19] and [20]. The former employs first-order formulas to represent a set of states called hypothesis; these are similar to our partial interpretations in that they represent a potentially infinite number of states. The key difference is that our approach explicitly defines random variables, (in)dependence assumptions, and their conditional distributions in relationship to other random variables, which allows us to efficiently compute the distribution of a random variable that needs to be sampled and added to the particle. In [20] the filtering problem is converted into a deterministic first order logic one. While their framework is suited for planning domains defined in probabilistic STRIPS, it does not define a sensor model.

Furthermore, none of the frameworks of [19], [20], [21] supports dealing with continuous random variables (other than through discretization), therefore these techniques cannot deal with real-world applications in robotics. Discretization is not always a good solution, and it can dramatically increase the number of states, therefore it is unclear whether these algorithms would maintain good performance in such cases. Finally, their FOL representation allows discrete and fixed probabilities (for the state transition model and measurement model), instead DCPF provides a flexible language to represent continuous or discrete distributions that can be parameterized by other random variables or logical variables used in the body. This allows a compact model and faster inference.

Several improvements to classical particle filtering have been proposed, such as Rao-Blackwellization [22] and Factored Particle Filtering [23], which support continuous and discrete variables and cluster the state space reducing the variance and improving the accuracy. These methods are effective, but they consider a fixed state vector and do not exploit a relational representation and partial states. They are also complementary to our work and could be adapted in future work.

B. Applications

Some state estimation applications with a relational representation have been proposed. The relational particle filter of [24] uses relations such as 'walking together' in people tracking to improve prediction and the tracking process. They divide the state in two sets: object attributes and relations, making some assumptions to speed up inference. In their approach a relation can be true or false. In contrast, our approach does not make a real distinction between attributes and relations, indeed, each random variable has a relational representation, regardless of the distribution (binary, discrete or continuous). This allows parametrization and template definition for any kind of random variable. Furthermore, our language and inference algorithm are more general, keeping inference relatively fast. In addition, it is not clear if they can support partial states and integrate background knowledge while keeping good performance. A relational representation has been used in [25] for situation characterization over time. However, this work is based on HMMs and uses only binary relations (true or false). Interesting works have

been proposed [26], [27] for manipulation tasks exploiting a relational representation. Those works integrate relational knowledge about the world to reason about the objects and perform complex tasks. For probabilistic inference and belief update they use MLNs (Markov Logic Networks). However, MLNs are generally slow and not suited for belief update over time, indeed they use an update rate around 0.1 Hz.

IX. CONCLUSIONS

We proposed a flexible representation for hybrid relational domains in temporal models and provided an efficient inference algorithm for filtering. This framework exploits the relational representation and the (in)dependence assumptions to reduce the particle size (through partial interpretations) and the inference cost. DCPF is particularly suited for (probabilistic) relational models that involve objects and relations between them. It performs fast inference about static variables/relations (e.g., from an ontology) and dynamic variables/relations in the same framework. It was empirically evaluated and applied in the magnetism scenario. The results show that DCPF outperforms the classical particle filter in those models, and is promising for robotics applications.

ACKNOWLEDGMENTS

We thank McElory Hoffmann, Guy Van den Broeck and Ingo Thon for the fruitful discussions and collaborations during the early stages of this work.

REFERENCES

[1] L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, Eds., *Probabilistic Inductive Logic Programming, Theory and Applications*, ser. Lecture Notes in Artificial Intelligence. Springer, 2008.

[2] L. Getoor and B. Taskar, *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[3] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt, "The magic of logical inference in probabilistic programming," *Theory and Practice of Logic Programming*, 2011.

[4] T. Sato, "A statistical learning method for logic programs with distribution semantics," in *Proceedings of the Twelth International Conference on Logic Programming*. MIT Press, 1995, pp. 715–729.

[5] A. Doucet, S. Godsill, and C. Andrieu, "On sequential monte carlo sampling methods for bayesian filtering," *STATISTICS AND COMPUTING*, vol. 10, no. 3, pp. 197–208, 2000.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[7] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, 1960.

[8] B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov, "BLOG: Probabilistic models with unknown objects," in *IJCAI*, 2005.

[9] F. Bancilhon and R. Ramakrishnan, "An amateur's introduction to recursive query processing strategies," *SIGMOD Record*, vol. 15, pp. 16–51, 1986.

[10] G. Kitagawa, "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models," *Journal of Computational and Graphical Statistics*, vol. 5, no. 1, pp. 1–25, 1996.

[11] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Proceedings of UAI 2000*. Morgan Kaufmann, 2000, pp. 176–183.

[12] A. Kimmig, V. Santos Costa, R. Rocha, B. Demoen, and L. De Raedt, "On the efficient execution of ProbLog programs," in *Logic Programming*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pp. 175–189.

[13] T. Swift and D. S. Warren, "Xsb: Extending prolog with tabled logic programming," *CoRR*, vol. abs/1012.5123, 2010.

[14] N. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: A language for generative models," in *UAI*, 2008, pp. 220–229.

[15] K. Kersting, L. De Raedt, and T. Raiko, "Logical hidden markov models," *Journal of Artificial Intelligence Research*, vol. 25, 2006.

[16] J. Choi, A. Guzman-Rivera, and E. Amir, "Lifted relational kalman filtering," in *IJCAI*, 2011, pp. 2092–2099.

[17] C. E. Manfredotti, D. J. Fleet, H. J. Hamilton, and S. Zilles, "Relational particle filtering," Monte Carlo Methods for Modern Applications, 2010 NIPS Workshop, Whistler, B.C., December 2010.

[18] I. Thon, N. Landwehr, and L. De Raedt, "Stochastic relational processes: Efficient inference and applications." *Machine Learning*, vol. 82, 2011.

[19] L. S. Zettlemoyer, H. M. Pasula, and L. P. Kaelbling, "Logical particle filtering," in *In Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*, 2007.

[20] H. Hajishirzi and E. Amir, "Sampling first order logical particles," in *UAI*, 2008.

[21] S. Natarajan, H. H. Bui, P. Tadepalli, K. Kersting, and W. keen Wong, "Logical hierarchical hidden markov models for modeling user activities," in *In Proc. of ILP-08*, 2008.

[22] G. Casella and C. P. Robert, "Rao-Blackwellisation of Sampling Schemes," *Biometrika*, vol. 83, no. 1, pp. 81–94, 1996.

[23] A. Pfeffer, S. Das, D. Lawless, and B. Ng, "Factored reasoning for monitoring dynamic team and goal formation," *Inf. Fusion*, vol. 10, no. 1, pp. 99–106, Jan. 2009.

[24] L. Cattelani, C. Manfredotti, and E. Messina, "A particle filtering approach for tracking an unknown number of objects with dynamic relations," *Journal of Mathematical Modelling and Algorithms in Operations Research*, pp. 1–19, 2012.

[25] D. Meyer-Delius, C. Plagemann, G. Wichert, W. Feiten, G. Lawitzky, and W. Burgard, "A probabilistic relational model for characterizing situations in dynamic multi-agent systems," in *Data Analysis, Machine Learning and Applications*. Springer Berlin Heidelberg, 2008.

[26] M. Tenorth and M. Beetz, "KnowRob - Knowledge Processing for Autonomous Personal Robots," in *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, 2009, pp. 4261–4266.

[27] M. Beetz, D. Jain, L. Mosenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic, "Cognition-enabled autonomous robot control for the realization of home chore task intelligence," *Proceedings of the IEEE*, vol. 100, no. 8, pp. 2454–2471, Aug.