Learning Declarative Bias

Will Bridewell[1] and Ljupčo Todorovski[1,2]

[1] Computational Learning Laboratory,
Center for the Study of Language and Information,
Stanford University, Stanford, CA, USA 94305
willb@csli.stanford.edu
[2] University of Ljubljana, Faculty of Administration
Gosarjeva 5, SI-1000 Ljubljana, Slovenia
ljupco.todorovski@fu.uni-lj.si

Abstract. In this paper, we introduce an inductive logic programming approach to learning declarative bias. The target learning task is inductive process modeling, which we briefly review. Next we discuss our approach to bias induction while emphasizing predicates that characterize the knowledge and models associated with the HIPM system. We then evaluate how the learned bias affects the space of model structures that HIPM considers and how well it generalizes to other search problems in the same domain. Results indicate that the bias reduces the size of the search space without removing the most accurate structures. In addition, our approach reconstructs known constraints in population dynamics. We conclude the paper by discussing a generalization of the technique to learning bias for inductive logic programming and by noting directions for future work.

Key words: inductive process modeling, meta-learning, transfer learning

1 Introduction

Research on inductive process modeling [1] emphasizes programs that build models of dynamic systems. As the name suggests, the models are sets of processes that relate groups of entities. For example, neighboring wolf and rabbit populations interact through a predation process, which may take one of many forms. As input, these programs take observations, which record system behavior over time, background knowledge, which consists of scientifically meaningful generic processes, and entities whose behavior should be explained. The output is a model that comprises processes instantiated with the available entities. A naive solution to the task would exhaustively search the space of models defined by the instantiated processes, but this approach produces several nonsensical models and the search space grows exponentially in the number of instantiations. To make inductive process modeling manageable in nontrivial domains, one must introduce bias.

Recently, researchers developed the notion of a process hierarchy to define the space of plausible model structures [2]. This solution defines which processes must always appear in a model, which ones depend on the presence of others, and which ones mutually exclude each other. Although one can use the hierarchy to substantially reduce the size of the search space, specifying relationships that both constrain the space and have validity in the modeled domain is difficult. Importantly, the introduction of this bias replaces the task of manually building a model with that of manually defining the space of plausible model structures. Ideally we would like to automatically discover this knowledge.

Ample literature exists on bias selection [3], which emphasizes search through the space of learning parameters, and constructive induction [4], which increases the size of the search space. In contrast, we wish to learn constraints that will reshape the search space and ensure that the program considers only plausible and accurate models. In the context of inductive process modeling, the learned constraints would imply the same restrictions as those encoded in the process hierarchy. For instance, in the case of the wolves and rabbits, we would like to discover that an accurate model of the dynamics must include a predation process. As this example hints, these constraints are generalizations drawn from the space of models.

To be more specific, we used inductive logic programming to find clauses that characterize accurate and inaccurate models. The examples are individual models from the search space that are classified as accurate or inaccurate according to their fit to training data. As further input, the background knowledge comprises descriptions of model organization such as predicates that indicate whether a model includes a particular process and which entities participate in a process. Given this information, we learn theories whose clauses describe accurate and inaccurate models, which we then turn into constraints on the model structures.

Before describing our approach in detail, we first provide a high level introduction of inductive process modeling with an emphasis on those aspects used to learn declarative bias. We then describe our learning algorithm, which relies on the combination of HIPM [2] and Aleph [5]. Next we describe promising results on a predator–prey domain and highlight the transfer of learned bias across multiple data sets. Following the experiments, we explain the general applicability of this approach to other artificial intelligence tasks and identify other work of a similar nature. Finally, we discuss limitations of our current approach and highlight its effect on inductive process modeling.

2 Inductive Process Modeling

Our program learns bias by analyzing the model structures that HIPM generates during search. For this reason, we provide a rough description of the knowledge representations for generic process libraries and quantitative process models. In the context of this paper, details about the internals of the processes (e.g., conditions and equations), the structural and parametric search techniques, and

Table 1. Part of a library for population dynamics, which includes both generic entities and processes. A generic process's type appears in braces after its name. For simplicity, we suppress each process's equations, conditions, and numeric parameters.

generic entity predator: variables concentration{sum};	generic entity prey: variables concentration{sum};
generic process exp_growth{growth}: entity E{prey, predator};	generic process log_growth{growth}: entity E{prey, predator};
generic process exp_loss{loss}: entity E{prey, predator};	generic process holling_type_1{predation}: entity P{prey}, R{predator};
generic process holling_type_2{predation}: entity P{prey}, R{predator};	generic process holling_type_3{predation}: entity P{prey}, R{predator};

the simulation routine are of less importance. We will briefly mention these aspects, which are described in greater detail elsewhere [1, 2].

At a high level, the background knowledge used by HIPM takes the form of the process library shown in Table 1. This library defines two generic entities which have properties (i.e., variables and constants). In this case, both declarations specify a single variable, *concentration*, with an additive combining scheme.[3] The generic entities fill labeled roles in the generic processes, which are defined below each process's name. For instance, *exp_growth* requires an instantiation of the generic entity *prey* or *predator* and in its complete form contains a constant that defines the growth rate and an equation that alters the prey's concentration. Notice that this generic process also has type *growth*, which is specified in braces after the name. In HIPM, process types define groups of generic processes that help delimit the search space. For instance, one could require that all population dynamics models include one of the many predation processes, letting the program select from these based on the model's fit to data.

In contrast to libraries, models contain instantiated entities and processes. For the entities, one must define the properties by specifying the values of constants and by either associating variables with a trajectory or stating their initial values. As an example, the entity *Wolf* could instantiate *predator* and its variable could refer to weekly recordings of a localized wolf population's size. Instantiating a process involves specifying the values of local constants and filling each labeled role with an entity that has the appropriate type. To a degree, and for the purposes of this paper, one can view generic processes and entities as predicates and the instantiated versions as ground facts.

Expanding on this logical view, a model is a conjunctive clause that can predict the quantitative behavior of a dynamic system. To explain a data set,

[3] When multiple processes influence a variable, one must aggregate the effects. To this end, HIPM supports combining schemes that select the minimal or maximal effect, add the effects, or multiply them.

Table 2. The logical representation of the population dynamics library from Table 1.

entity(predator).	entity(prey).
process(exp_growth, growth).	process(log_growth, growth).
process(exp_loss, loss).	process(holling_type_1, predation).
process(holling_type_2, predation).	process(holling_type_3, predation).

HIPM constructs model structures that satisfy constraints, which are part of the bias. The program then estimates the numeric parameters of each model using a gradient search algorithm [6] coupled with random restarts. HIPM relies on CVODE [7] to simulate the system of equations entailed by the model and compares the trajectories to measured time-series. To learn bias, our program employs a more general logical view of the model and couples it with a predicate that associates the structure with a particular level of accuracy.

3 Learning Bias by Inductive Logic Programming

The primary contribution of this paper is a logical representation that lets one learn declarative bias by analyzing the space of possible models. We developed such a formalism for describing both libraries and models associated with inductive process modeling and we present a strategy for applying inductive logic programming tools. In this section, we describe the formalism, the input to the learning system, the performance element that uses the learned rules, and the learning software that we employed.

Model structures contain a number of processes, each of which relates a number of entities, and each of those may be shared among multiple processes. These three properties indicate the inherent relational structure of the models and suggest that a first-order representation would best capture this characteristic. Indeed, in Section 2 we suggested that the inductive process modeling representation resembles first-order logic. That is, one could view the names of generic processes and entities as predicates and instantiations of these as ground facts. However, this mapping creates a domain-specific language that one must tailor to other modeling problems. For instance, we would like to use the same predicates for describing knowledge and models from the population dynamics domain as for those in a physiological one. Therefore, we designed domain-independent predicates that explicitly characterize the structure of processes and models.

Table 2 contains an encoding of the population dynamics library from Table 1. There are only two predicates in this representation: *entity* and *process*. For our program, the only relevant information about a generic entity is its name. Generic processes differ slightly in that they have both a name, the first argument, and a type, the second one. Currently we leave the entity roles of a generic process unspecified because the instantiated processes in each model

Table 3. The logical representation of a population dynamics model.

model(m1).	r2(m1, 0.85).
process_instance(m1, p1, exp_growth).	process_instance(m1, p2, exp_loss).
parameter(m1, p1, aurelia).	parameter(m1, p2, nasutum).
process_instance(m1, p3, holling_type_1).	
parameter(m1, p3, aurelia).	entity_instance(nasutum, predator).
parameter(m1, p3, nasutum).	entity_instance(aurelia, prey).

relate entities according to the type specification in the library. A key feature of this formalization is that it limits the domain-specific information to the process library and lets us build high-level predicates that generalize to other tasks.

The logical representation for process models resembles that for the library with the key differences that it ties the components of individual models together and that it includes information about the process parameters. To illustrate, consider the model in Table 3. Since the examples are a collection of ground facts, we need a way to associate processes, parameters, and entities with particular models. Here, we introduce the predicate *model* that declares a ground term to be a model identifier. This term appears in all predicates associated with that particular model except for *entity_instance*. Since all models explain the behavior of the same entities, a direct tie between them is unnecessary.

Similarly, we associate parameters with particular processes. For instance, the model in Table 3 contains a *process_instance* that belongs to model *m1*, has the unique identifier *p1*, and instantiates the generic process *exp_growth*. The use of the unique identifier lets one relate multiple entities to a single process.

The collection of ground facts from Table 3 defines an organized model structure that has a particular accuracy. Here we encode this value with the *r2* predicate, which records the coefficient of determination calculated over the training data. This value falls in the range $[0, 1]$ and indicates how well the shape of the simulated trajectory matches that of the observed values.

Although the predicates for the generic process library and for instantiated models define the structures, they would lead to clauses that are difficult to interpret. To address this problem, we introduce a set of higher level predicates that *describe* the structures in terms of properties. These predicates combine those from Tables 2 and 3 into forms like those in Table 4. To illustrate, the predicate `includes_processtype_entity(m1, p1, aurelia)` compactly represents the conjunction `process_instance(m1, p1, exp_growth), process(exp_growth, growth), parameter(m1, p1, aurelia), entity_instance(aurelia, prey)`. We use these higher-level predicates *in addition* to the lower-level ones so that one can identify comprehensible rules without losing the ability to learn unanticipated relationships among model structures.

Table 4. Examples of high-level predicates that describe the model structure.

includes_processtype(M, T) :– model(M), process(P, T), process_instance(M, _, P).	includes_processtype_entity(M, T, E) :– model(M), entity(E), process_instance(M, PI, P), process(P, T), parameter(M, PI, EI), entity_instance(EI, E).

Whereas the library definition and high level predicates compose the background knowledge, the models serve as the examples. To assign the models to a target class, we use the predicates *accurate_model* and *inaccurate_model*. We define the rules for these predicates as `accurate_model(M) :- r2(M, R), R` $\geq$ `threshold.` and `inaccurate_model(M) :- r2(M, R), R` $<$ `threshold.`[4] We describe the method for selecting a threshold in Section 4.

Given background knowledge and examples, our approach produces theories whose clauses characterize the structure of accurate and inaccurate models. We can use each such clause to bias the search space of candidate model structures. To this end, the clauses that predict accurate models specify which structures should remain in the search space, while the clauses for inaccurate models state ones the automated modeler should prune. To learn these rules, we apply the inductive logic programming system Aleph, which produces separate theories for accurate and inaccurate models. For example, Aleph could learn the clause, `accurate_model(M) :- includes_processtype(M, predation).`, which we would turn into background knowledge for HIPM that forces all models to contain a process with type *predation*.

As we have described the task, learning declarative bias seems like a natural problem for inductive logic programming; however, a propositional approach would work as well since we are working in a finite domain without recursive predicates. For instance, given a fixed, finite domain of processes, we could construct a Boolean feature vector for each model that contains all the properties captured by the background knowledge. This method would effectively involve a reimplementation of the LINUS system [8], which, as a result, reinforces the appropriateness of a relational learner for inducing declarative bias. In the next section, we evaluate our described approach in an ecological domain.

4 Empirical Evaluation

To determine the utility of the approach described in Section 3, we applied it in the domain of population dynamics. After describing the experimental method, we detail a strategy for selecting an appropriate threshold value to separate accurate models from inaccurate ones. In the rest of the section, we evaluate three conjectures about the learned bias. First, we expect the bias will substantially

[4] Note that *r2* is not a background knowledge predicate for learning bias.

reduce the search space of candidate model structures. Second, we expect the reduced search space will retain the most accurate models. And third, we anticipate that the learned model constraints will be consistent with the existing knowledge in the domain of population dynamics modeling and have a potential to contribute new findings to it.

4.1 Method

We performed the experiments on three modeling tasks. Each includes modeling population dynamics from measured time-series of concentrations of two species, aurelia and nasutum [9], involved in a predator–prey interaction. We ran HIPM on the three tasks with an initial library that lack constraints on process combinations in models. HIPM performed exhaustive search through the space of 9402 model structures that have up to five processes, fit the constant parameters of each candidate structure against the measured time-series, and reported the obtained model and its performance (i.e., the coefficient of determination on the training time-series). We reformulated the traces of HIPM runs in first-order logic to create three data sets for learning bias: PP1, PP2, and PP3.

For each data set, we must first select the value of the performance threshold that distinguishes accurate models from the inaccurate ones first. We evaluate the effect of the threshold value on the recall of the best models in the next subsection, and based on this analysis, we propose a method for selecting an optimal threshold for a given data set. Having selected the threshold value for the training set, we run Aleph to induce bias, and then evaluate its performance on the remaining two data sets which were unseen during learning.[5] We use Aleph's default parameter settings, except that we set its noise parameter to 10, which lets a clause cover up to 10 negative examples, the minpos parameter to 2, which prevents Aleph from adding ground facts to the theory, and the nodes parameter to 100,000, which increases the upper bound on the program's search space complexity.

To validate our conjectures about the induced bias, we use two quantitative evaluation metrics and qualitative analysis. First, we measure the size of the search space before and after applying the learned bias. Formally, we calculate the *reduction factor* as $|SS_0|/|SS|$, where $|SS_0|$ denotes the number of models in the unconstrained search space and $|SS|$ is the number of model structures in the biased space. Second, *best model recall* measures the percentage of the n best models of a particular data set in the entire search space that also appear in the reduced space. Ideally, the reduced space would retain 100% of the best models. In the experiments, we measure recall of the ten and fifty best models. Next, we compare the distribution of the model performance in the reduced search space to the distribution of the model performance in the entire space. We expect the overall performance of models in the reduced search space to compare favorably

[5] Since HIPM performs exhaustive search, we do not run HIPM again to search the constrained space of models. Instead, we remove the model structures that violate the induced constraints.

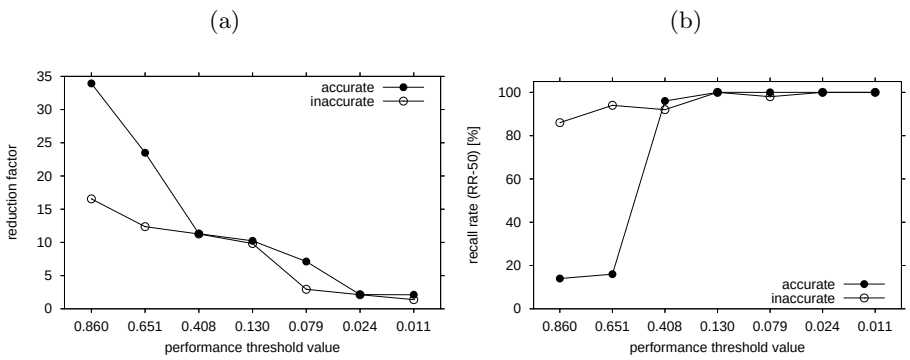

Fig. 1. Sensitivity analysis of the bias to the performance threshold value as calculated on training data. The graph on the left-hand side (a) shows the reduction of the search space, while the one on the right-hand side (b) shows the recall of the best fifty models. Full and empty circle symbols indicate the performance of the model constraints for the accurate and inaccurate class, respectively.

to the overall performance of all candidate models. Finally, we present the model constraints that were learned from all three data sets and analyze them in terms of their consistency with existing knowledge in the population dynamics domain.

4.2 Selecting a Performance Threshold

Before we select an appropriate performance threshold for classifying a model as accurate or inaccurate, we analyze its effect on bias performance in terms of search space reduction and best models recall. To identify a list of plausible threshold values, we rank the models according to their accuracy and divide them into 10 bins. Then we select the point of maximal performance change between two consecutive models in each bin. This leads to an initial list of ten candidate thresholds, which we revise by removing consecutive points that are close to each other.

The graph in Figure 1(a) shows that the relation between the performance threshold and the search space reduction is strictly monotonic. As one would expect, high threshold values render most of the models inaccurate, which leads to highly specific model constraints. Using these specific model constraints considerably narrows the search space, which is reflected in the high values of the reduction factor (over 30). On the other hand, the graph in Figure 1(b) shows that the bias corresponding to high threshold values is too restrictive and removes most of the best models from the search space. Model constraints induced with threshold values of 0.860 and 0.651 filter out most of the fifty best models, while the others include most of the best models. The model constraints for the class of inaccurate models follow roughly the same pattern.

Note that the analysis performed here is limited to the training data set, to emphasize the fact that we select the performance threshold on the basis of

Table 5. Evaluating the utility of the model constraints learned on a train data set TrainDS (for accurate and inaccurate target predicates) on test data sets TestDS. For each bias, the table reports the reduction of the search space (RSS) and the recall of the top ten (BMR-10) and top fifty (BMR-50) models for the test set.

Model Constraints (Bias)			Bias Evaluation		
TrainDS	Class	RSS	TestDS	BMR-10[%]	BMR-50[%]
PP1	accurate	11×	PP2	100	96
			PP3	100	94
PP1	inaccurate	11×	PP2	90	88
			PP3	90	92
PP2	accurate	16×	PP1	100	98
			PP3	60	36
PP2	inaccurate	11×	PP1	100	98
			PP3	90	82
PP3	accurate	10×	PP1	100	100
			PP2	100	100
PP3	inaccurate	9×	PP1	100	100
			PP2	100	98

training data only. The bias performance change on the test data sets correlates highly with the results on the training data and is virtually identical to that shown in Figure 1(b). This indicates the good generalization performance of the induced bias, which we further analyze in the next subsection.

In summary, the graphs in Figure 1 clearly render 0.130 and 0.408 as optimal threshold values, since they both lead to a substantially reduced search spaces that retain most of the best models. Based on this analysis, we use 0.408 as threshold for performing further experiments on the PP1 data set. The analysis of threshold influence on the bias performance on the other two data sets, PP2 and PP3, shows a similar effect.

4.3 Evaluating the Generalization Performance

Once we identify the optimal threshold value for learning bias on a particular data set, we induce the bias using that threshold, and we analyze its performance on the other two data sets. Table 5 summarizes the results of the evaluation. All model constraints, induced on different data sets and for different target predicates, lead to a reduction factor ranging from 9 to 16. The highest reduction rate is observed for the bias induced on the PP2 data set for the inaccurate class.

All induced model constraints recall most of the top ten models for all test data sets, except the ones induced on the PP2 data (inaccurate class) that recall 6 of the top ten models for PP3. The recall of the top fifty models is lower, but still over 90% for most of the cases, with the exception for the bias induced from the PP2 data set (82% and 63% recall of the top fifty models for PP3 using inac-

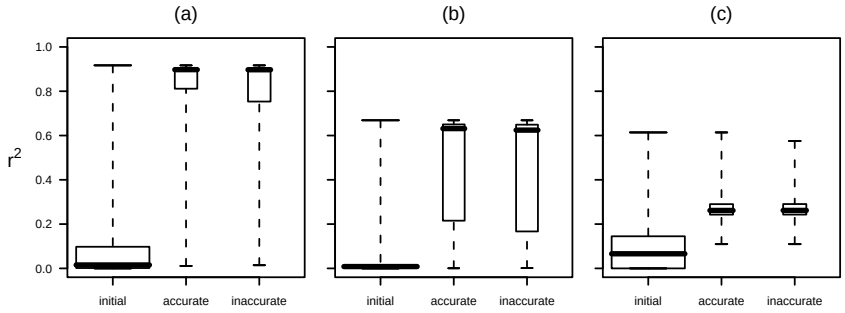

Fig. 2. Comparing the distributions of the model performance in the initial (unconstrained) search space to the performance distributions in the biased spaces induced from the PP1 data set (accurate and inaccurate class). Graphs (a), (b), and (c) compare the distributions on the PP1, PP2, and PP3 data sets respectively.

curate and accurate constraints, respectively). The worse overall performance is observed when applying the bias induced from the accurate models in PP2 data set, which is the most restrictive bias in terms of the search space reduction.

The graphs in Figure 2 compare the performance distribution for the models in the initial (unconstrained) search space to the performance distribution in the search spaces constrained by the bias induced from the PP1 data set. Figure 2(a) shows that the median r^2 in the entire search space is close to 0 denoting that most of the models perform poorly on the training time-series (recall that the range of r^2 is [0,1]). The learned bias focuses the search to the space of candidate model structures that perform much better, their median value being 0.89. Note however, that the bias does not filter out all inaccurate models from the search space, since the minimal observed performance remains close to 0. The graphs in Figures 2(b) and 2(c) show that the constraints induced on the PP1 data set also narrows the search focus on the test data sets, PP2 and PP3. Bias still shifts the distribution towards models with better performance: from median r^2 close to 0 to medians of 0.63 and 0.26, for PP2 and PP3 respectively. However, the performance distribution for PP2 is blurred towards worse performing models indicating that the bias induced on the PP1 data set allows a considerable number of sub-optimal models. Nevertheless, the comparison of the distributions confirm that the bias learned on PP1 generalizes well to PP2 and PP3. The effect of the constraints induced from PP2 and PP3 are comparable.

4.4 Semantic Analysis of the Induced Constraints

For each data set Aleph induced a different collection of model constraints. Yet, eight of these, presented in Table 6, appear in every theory. Since these constraints provide generalizations that we can match against existing domain knowledge, we analyze their potential to enrich it.

Table 6. The eight rules that appear in each of the theories induced from the PP1, PP2, and PP3 data sets.

```
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, exp_growth, prey),
  includes_process(M, hassell_varley_2).
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, exp_growth, prey),
  includes_process(M, holling_type_3).

accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, log_growth, prey),
  includes_process(M, hassell_varley_2).
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, log_growth, prey),
  includes_process(M, holling_type_2).
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, log_growth, prey),
  includes_process(M, holling_type_3).

inaccurate_model(M) :-
  doesnotinclude_processtype_entity(M, growth, prey).
inaccurate_model(M) :-
  doesnotinclude_processtype_entity(M, loss, predator).
inaccurate_model(M) :-
  doesnotinclude_processtype(M, interaction).
```

Before describing the clauses produced by Aleph, we acknowledge their propositional nature and stress its superficiality. To illustrate, we point out that the higher-level predicates mask the relational structure of the rules. For example, one can rewrite the first rule in Table 6 in terms of the lower-level predicates as

```
accurate_model(M) :-
 model(M), entity(predator), process_instance(M, PI, P),
 process(P, loss), parameter(M, PI, EI),
 entity_instance(EI, predator), process(exp_growth), entity(prey),
 process_instance(M, PI2, exp_growth), parameter(M, PI2, EI2),
 entity_instance(EI2, prey), process(hassell_varley_2),
 process_instance(M, PI3, hassell_varley_2).
```

In addition, had we relaxed the limitations on which entity types could bind to particular processes (e.g., by letting entities having type *predator* bind to *growth* processes and those having type *prey* bind to *loss* processes), rules such as

```
inaccurate_model(M) :- includes_processtype_entity(M, growth, ET),
 includes_processtype_entity(M, loss, ET).
```

would likely appear.

Turning now to the semantic analysis of the rules, we see that five of the frequent clauses shown in Table 6 characterize accurate models. The first two specify that the structure of an accurate predator–prey model includes three processes: loss of the predator, exponential growth of the prey, and one interaction process, which may be one of the two specific formulations. The three other clauses that characterize accurate models are similar, since they also claim that an accurate models include three processes of loss, growth, and interaction. The difference from the first two rules is that they specify alternative form of the growth process (logistic instead of exponential) and a larger set of interactions.

Finally, the three model constraints for the inaccurate_model predicate paraphrase the rules for the class of accurate models, but they are more general. They identify the three main properties of an inaccurate model's structure: the lack of prey species growth, the lack of predator loss, and the lack of interaction between species. In other words, an accurate model structure should include at least one process of each type, which is a rediscovery of the well known fact established in early work by Lotka and Volterra [10]. On the other hand, the five rules for the accurate models establish novel hypotheses about predator–prey interaction between the observed species, which ecologists may further evaluate.

We consider the reconstruction of well-known facts from the domain of population dynamics as important evidence about our program's potential to learn useful and meaningful bias constraints. This result also improves the credibility of the hypotheses established by the other model constraints.

5 General Discussion

Although initial results suggest the feasibility of our approach to inducing bias, many questions remain. In this section, we describe related work and explore the generality and limitations of our method.

In the introduction, we differentiated our work from bias selection and constructive induction, but there are other similar approaches that we should discuss. In particular, our research falls within the general category of metalearning [11], but much of this work emphasizes the prediction of algorithm performance, whereas we use the output of learning to reshape the search space for an algorithm's future applications. Instead, our work more closely matches that of McCreath and Sharma [12] who used inductive logic programming to learn mode and type declarations, which could constrain the space of candidate clauses. Notably, their program produced syntactic constraints unrelated to the specific domain, whereas our approach induces semantic ones that are interpretable within

the domain's context. Additionally, we note similarities with learning control rules for planning [13] since the algorithms analyze the output of the planner to improve future performance. However, such systems generally view only the operators and the context of their application as opposed to an entire plan.

The predicates with which we characterize model structures resemble the relational clichés introduced by Silverstein and Pazzani [14]. Relational clichés are conjunctions of predicates that are useful for building classification rules in a particular domain. As such they relate to a combination of processes that must appear in an accurate model. While our approach learns these combinations (i.e., clichés) from examples of inaccurate and accurate models, the work presented in [14] does not deal with learning clichés but rather demonstrates the benefit of using them as declarative bias for learning classification rules. More recently, Morin and Matwin [15] proposed an approach to inducing relational clichés, but instead of using the meta-learning approach presented here, they learn clichés directly from the examples in one domain and then transfer them into another domain. Their work focuses on learning and transfer of bias between domains and not on learning constraint rules that would contribute to the theory in the domain of interest. On the other hand, transferring induced knowledge to other domains is an open challenge for our approach.

Even though we showed how to learn bias in the limited context of inductive process modeling, we expect that it will generalize to other domains. For instance, in the case of inductive logic programming, one would examine each evaluated clause as a separate entity and identify a bias that restricts the structure of the antecedents. This usage would require the learning program to report the performance of all considered clauses instead of just those in the final theory, but such an extension requires minimal effort with the potential for substantial gains in both the effectiveness of the search and the plausibility of the induced theories. Extensions to propositional and association rule learning are similar.

Apart from generalization to other artificial intelligence tasks, there are several open avenues for future work. First, in this paper, we assume that exhaustive search of the model space is possible. Such scenarios are uncommon, and we need to better understand the effects of model sampling on the induction of bias. Second, we would like to use a similar approach to analyze the best performing models in a domain. This task requires an inductive logic programming system that learns from positive examples only [16] and raises questions about what to include in the set of best models. Third, Reid [17] introduces the idea of learning an evaluation bias, which lets one infer the reliability of a logical rule from its past performance in related tasks. In the spirit of his research, we would like to estimate the quantitative fit of a model structure based upon its performance in similar domains. This step would let a program establish priorities over a set of candidate structures so that "better" ones would have earlier access to the computationally expensive parameter-estimation routine. Finally, Pazzani and Kibler [18] show that biasing the space of candidate models with domain-specific knowledge helps reduce overfitting and improves overall accuracy on a test set.

We need to evaluate whether this holds true when the bias is automatically induced.

6 Conclusion

In this paper, we developed a representation that lets one learn declarative bias for inductive process modeling using tool provided for inductive logic programming. Our primary contribution is that we showed how to construct the background knowledge, how to describe the examples, and how to select a threshold for the supervised learning task. We then evaluated our approach on a population dynamics domain and found that the learned bias substantially reduced the size of the candidate model structure space. We also found that the bias increased the proportion of accurate models in both the training data and test data taken from the same domain. Importantly many of the induced constraints verified known ecological theory. Finally, we described related work, proposed the generalization of this method to other learning algorithms, and highlighted future work that will lead to a better understanding of this research area. We believe that the reported approach opens a promising new avenue for scientists in artificial intelligence that is rich with open questions.

Acknowledgments. This research was supported by Grant No. IIS-0326059 from the National Science Foundation. We thank Pat Langley, Stuart Borrett, and Tolga Könik for discussions that influenced the ideas in this paper.

References

1. Langley, P., Sánchez, J., Todorovski, L., Džeroski, S.: Inducing process models from continuous data. In: Proceedings of the Nineteenth International Conference on Machine Learning, Sydney, Morgan Kaufmann (2002) 347–354
2. Todorovski, L., Bridewell, W., Shiran, O., Langley, P.: Inducing hierarchical process models in dynamic domains. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, Pittsburgh, PA, AAAI Press (2005) 892–897
3. Provost, F., Buchanan, B.: Inductive policy: The pragmatics of bias selection. Machine Learning **20** (1995) 35–61
4. Utgoff, P.E.: Machine Learning of Inductive Bias. Kluwer Academic Publishers, Boston, MA (1986)
5. Srinivasan, A.: The Aleph Manual. Computing Laboratory, Oxford University. (2000)
6. Bunch, D.S., Gay, D.M., Welsch, R.E.: Algorithm 717: Subroutines for maximum likelihood and quasi-likelihood estimation of parameters in nonlinear regression models. ACM Transactions on Mathematical Software **19** (1993) 109–130
7. Cohen, S., Hindmarsh, A.: CVODE, a stiff/nonstiff ODE solver in C. Computers in Physics **10** (1996) 138–143
8. Lavrac, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)

9. Jost, C., Ellner, S.: Testing for predator dependence in predator–prey dynamics: A non-parametric approach. Proceedings of the Royal Society of London B **267**(1453) (2000) 1611–1620
10. Kingsland, S.E.: Modeling Nature. Second edn. The University of Chicago Press, Chicago, IL (1995)
11. Giraud-Carrier, C., Vilalta, R., Brazdil, P.: Introduction to the special issue on meta-learning. Machine Learning **54** (2004) 187–193
12. McCreath, E., Sharma, A.: Extraction of meta-knowledge to restrict the hypothesis space for ILP systems. In: Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence, Canberra, Australia, World Scientific Publishers (1995) 75–82
13. Huang, Y., Selman, B., Kautz, H.A.: Learning declarative control rules for constraint-based planning. In: Proceedings of the Seventeenth International Conference on Machine Learning, Stanford, CA, Morgan Kaufmann (2000) 415–422
14. Silverstein, G., Pazzani, M.J.: Relational clichés: Constraining constructive induction during relational learning. In: Proceedings of the Eighth International Workshop on Machine Learning, Morgan Kaufmann (1991) 203–207
15. Morin, J., Matwin, S.: Relational learning with transfer of knowledge between domains. In: Proceedings of the Thirteenth Biennial Conference of the Canadian Society for Computational Studies of Intelligence, Springer (2000) 379–388
16. Muggleton, S.: Learning from positive data. In: Proceedings of the Sixth International Workshop on Inductive Logic Programming, Stockholm, Sweden, Springer (1996) 358–376
17. Reid, M.: DEFT guessing: Using inductive transfer to improve rule evaluation from limited data. Ph.D. thesis, University of New South Wales, Sydney, Australia (2007)
18. Pazzani, M.J., Kibler, D.F.: The utility of knowledge in inductive learning. Machine Learning **9** (1992) 57–94