Two Kinds of Knowledge in Scientific Discovery

Will Bridewell and Pat Langley

Computational Learning Laboratory

Center for the Study of Language and Information

Stanford University, Stanford, CA 94305 USA

June 15, 2008

Abstract

Research on computational models of scientific discovery investigates both the induction of descriptive laws and the construction of explanatory models. Although the work in law discovery centers on knowledge-lean approaches to searching a problem space, research on deeper modeling tasks emphasizes the pivotal role of domain knowledge. As an example, our own research on inductive process modeling uses information about candidate processes to explain why variables change over time. However, our experience with IPM, an AI system that implements this approach, suggests that process knowledge is insufficient to avoid the consideration of implausible models. To this end, the discovery system needs additional knowledge that constrains the model structures. We report on an extended system, SC-IPM, that uses such information to reduce its search through the space of candidates and to produce models that human scientists find more plausible. We also argue that, although people carry out less extensive search than SC-IPM, they rely on the same forms of knowledge—processes and constraints—when constructing explanatory models.

1 Introduction

As a field, computational scientific discovery seeks to understand the products and processes of science by studying artifacts that engage and assist in knowledge construction. Along these lines, researchers have investigated activities like taxonomy formation, law discovery, and theory development. Their findings have demystified these activities and suggest a strong link between the disciplined work of scientists and the everyday reasoning skills shared by everyone. Moreover, this work has revealed previously implicit types of knowledge crucial to scientific reasoning.

The field's earliest and most fundamental result was that programs could actually make discoveries. Up to the late twentieth century, the belief that scientific discovery requires a "divine spark" peculiar to humanity dominated academic thought on the matter. However, a continual stream of programs that recapitulate historical discoveries and make new ones of their own has served to reshape scientific activity as an understandable, computational process based on problem-space search. Some of the earliest support for this view came from the BACON series of programs which rediscovered Kepler's third law, the ideal gas law, and other numerical relationships from experimental data (Langley et al., 1987). Later systems expanded rediscovery efforts to new problems that involved biochemical reactions (Kulkarni & Simon, 1990), genetic networks (Zupan et al., 2007), and other scientific knowledge. Many systems targeted other tasks such as the autonomous design, execution, and analysis of experiments (Żytkow et al., 1990); the discovery of equations from observational data (Todorovski & Džeroski, 1997); and the assistance of scientists in their own discoveries (Mahidadia & Compton, 2001). Additionally, some researchers investigated scientific reasoning by exploring its links with everyday activities (Pazzani & Flowers, 1990; Klahr, 2002) bringing a new perspective to the field.

An unexpected benefit of research on computational scientific discovery has been the recognition of new types of scientific knowledge that are often not discussed in the literature on the history, philosophy, and psychology of science. This knowledge arose from the practical nature of system building as researchers found it necessary for a program to function. One of the earliest such findings was the chemical fracturing heuristics in DENDRAL (Feigenbaum et al., 1971),

which interpreted the results from a mass spectrometer. More recently, Żytkow (1999) introduced models as distinct mediators between scientific theory and data. Our own work expanded on this by identifying the role of processes in explanatory scientific models and the importance of their modularity (Bridewell et al., 2006; Bridewell et al., 2008).

In this paper, we report an extension to this work on process modeling that introduces another type of scientific knowledge. We first review the task of *inductive process modeling* and describe a computational approach that carries out search for a reasonable explanation. This search employs generic processes, which are a form of background knowledge that defines the space of candidate models. We then introduce modeling constraints, which are another type of scientific knowledge that often remains implicit. Following this, we describe an extended system that includes and respects these types of constraints. Afterwards, we recount experimental results that analyze the effects modeling constraints and conclude with a summary of our findings, a discussion of some closely related work, and directions for future research.

2 Inductive Process Modeling

In many fields, scientists use *mechanisms*, which illustrate the relationships among entities and processes, to express causal explanations. In these models the *entities* are physical objects with spatial extension and have properties of interest such as weight, color, and concentration. Although a particular entity may refer to a single object, such as a bird or a table, it can also represent a group of objects like a flock or a population. The degree of aggregation depends primarily on the driving scientific question and the methods for addressing it. The *processes* in a mechanism connect entities to each other and are the manifestations of change in the world. Unlike entities, which have measurable properties, processes are unobservable, so scientists infer their existence by observing changes in entities. The analog to aggregation in a process is the level of detail. For instance, a model could represent plant growth as a high-level process that increases a population exponentially, or it could explicitly include the chemical processes that compose photosynthesis.

(a)	(b)	
$\dot{p} = \alpha p - \beta pn$	$\dot{p} = \underbrace{\alpha p}_{growth} - \underbrace{\beta pn}_{predation}$	α =growth rate γ =death rate
$\dot{n} = \epsilon\beta pn - \gamma n$	$\dot{n} = \underbrace{\epsilon\beta pn}_{predation} - \underbrace{\gamma n}_{death}$	β =attack rate ϵ =efficiency

Table 1: The Lotka–Volterra equations for population dynamics (a) unlabeled and (b) labeled with the theoretical interpretation of the components.

As with entity aggregation, the level of detail for processes depends on the scientific question currently under investigation.

When building such causal explanations scientists appeal to a mixture of theory and data. Imagining a spectrum of modeling practices, we place theory-driven or "first principles" approaches at one endpoint and data-driven or "inductive" approaches at the other. Although one can build models according to first principles, in practice, empirical observations play a role in estimating numerical parameters and validating explanations. Likewise, scientists can build purely descriptive models from data, but theoretical concerns still influence variable selection and data collection. Moreover, descriptive models lack explanatory import unless they make contact with theoretical concepts. For example, the Lotka–Volterra equations in Table 1(a) describe a generic interaction between two variables until we designate their meaning as in Table 1(b). Although illustrated in quantitative terms, these ideas also apply to qualitative models, which must in principle connect to data and domain theory to have relevance.

We claim that building mechanisms is a form of scientific discovery that fits the mold of search through a problem space. Toward this characterization, we have developed a formal representation for an important class of mechanisms, called *quantitative process models*, and the domain knowledge from which they are built. In these models, an entity is a named collection of the variables and constants that define its state. A process refers to entities and contains algebraic and differential equations that specify its effects. Optionally, each process may contain conditions that signal its applicability (e.g., if a variable's value exceeds a threshold). Table 2 contains an example model

Table 2: A quantitative process model from population dynamics. The entities $rabbit$ with type $prey$ and fox with type $predator$ each have a variable p that stores their respective population size. The notation $d[X, t, 1]$ indicates the first derivative of X with respect to time t.

model predator_prey
entities $rabbit\{prey\}, fox\{predator\}$
process exponential_growth
equations $d[rabbit.p, t, 1] = 2.5 * rabbit.p$
process exponential_loss
equations $d[fox.p, t, 1] = -1.2 * fox.p$
process predation_holling_1
equations $d[rabbit.p, t, 1] = -0.1 * rabbit.p * fox.p$
$d[fox.p, t, 1] = 0.3 * 0.1 * rabbit.p * fox.p$

that includes processes and entities related to population dynamics. Given initial conditions for the variables determined by differential equations, one can simulate the model using a numeric solver such as CVODE (Cohen & Hindmarsh, 1996). Importantly, removing a process completely removes that interaction from the model, so one can replace the exponential growth process in Table 2 with logistic growth without disturbing the rest of the model's structure.

This interchangeability suggests a method for exploring a space of causal explanations. That is, given a collection of potential processes, a program could combine them into candidate models and evaluate their ability to account for dynamic phenomena. As defined, processes like those in Table 2 are specific to a particular situation; however, their general forms are not. For instance, the *generic process* for exponential growth applies to a variety of scenarios with wildly different entities and growth rates. Distinct from their instantiated counterparts, generic processes have entity roles instead of entities and parameter ranges instead of fixed values. The entity roles are placeholders for entities that must themselves instantiate a particular *generic entity* that has the properties altered by the process. Table 3 shows an example library of generic entities and processes for population dynamics that supports a large space of causal models.

Together the generic processes and generic entities form the domain theory for a modeling task, but to evaluate a causal explanation one needs data. In particular, a program for building quantitative process models can guide its search through the problem space by evaluating each

Table 3: A generic library for population dynamics. The variable type constraints are denoted in braces following the variable's name, while parameter bounds are specified within brackets. The notation $d[S, t, 1]$ indicates the first derivative of S with respect to t.

---

library population_dynamics
generic entity predator
  variables c
generic entity prey
  variables c
generic process logistic_growth
  relates $P\{prey\}$
  parameters $gr[0, 3], ic[0, 0.1]$
  equations $d[P.c, t, 1] = gr * P.c * (1 - ic * P.c)$
generic process exponential_growth
  relates $P\{prey\}$
  parameters $gr[0, 3]$
  equations $d[P.c, t, 1] = gr * P.c$
generic process exponential_loss
  relates $R\{predator\}$
  parameters $dr[0, 2]$
  equations $d[R.c, t, 1] = -1 * dr * R.c$
generic process predation_holling_1;
  relates $P\{prey\}, R\{predator\}$;
  parameters $ar[0.01, 10], ef[0.001, 0.8]$;
  equations $d[P.c, t, 1] = -1 * ar * P.c * R.c$;
        $d[R.c, t, 1] = ef * ar * P.c * R.c$;
generic process predation_holling_2;
  relates $P\{prey\}, R\{predator\}$;
  parameters $ar[0.01, 10], ef[0.001, 0.8], ht[1, 5]$;
  equations $d[P.c, t, 1] = -1 * ar * P.c * R.c/(1 + ht * ar * P.c)$;
        $d[R.c, t, 1] = ef * ar * P.c * R.c/(1 + ht * ar * P.c)$;

---

candidate's fit to observed trajectories. Ideally these data would include values for each variable that determines the system's state (e.g., the sizes of the fox and rabbit populations) to support comprehensive appraisal, but this is not always practical. More importantly, one should record a system's state at intervals relevant to the central dynamics—predation between foxes and rabbits operates on a different timescale from RNA transcription. So modeling a dynamic system requires not just any data associated with the problem, but data appropriate to the question at hand.

We claim that, when scientists build explanatory models in many domains, they have in mind knowledge similar to what we describe. The development and research of programs that carry out a similar task requires an explicit statement of the activity's input and output. To this end, we define *inductive process modeling* as:

- *Given*: Observations for a set of continuous variables as they change over time;
- *Given*: Generic entities that have properties relevant to the observed dynamics;
- *Given*: Generic processes that specify causal relations among entities using generalized functional forms;
- *Given*: A set of entities present in the modeled system;
- *Find*: A specific process model that, when given initial values for the modeled variables and values for any exogenous (i.e., forcing or unmodeled) variables, explains the observed data and predicts unseen data accurately.

A system that carries out this task would produce a model that links domain knowledge to scientific data. Importantly, that model would explain the measured phenomena in a formalism much like a scientist's own.

We developed IPM to carry out this task as a program that searches the space of process models. This search operates in two stages, first building a candidate structure and second estimating its numerical parameters.[1] Initially, IPM binds the generic processes to the problem specific entities. These *bound processes* are more concrete than their generic counterparts, but still lack numerical parameters. Processes in this form serve as the atomic elements of a model's structure, and a set of them specifies the conjectured interactions among entities. In fact, a candidate solution consists of the entities, a set of bound processes, and the numerical parameters.

The space of candidate models grows exponentially with the number of bound processes, so IPM uses beam search to explore the space of candidate solutions.[2] Our implementation of beam search incorporates a simplicity bias, which means that IPM considers models with fewer processes

[1] We omit details of the parameter estimation routine, noting that it is based on a local search algorithm like gradient descent that minimizes the error between predicted and observed values.

[2] The program also supports exhaustive search, which is often impractical due to the number of candidate models.

first. In general, this strategy takes a set of candidate solutions, keeps a few that perform the best, and uses those to guide the next round of exploration. Search halts when an exploration round fails to produce any solutions better than those already considered. In many cases, beam search quickly finds high quality solutions even though its simplicity bias is naive for IPM's modeling task.

Although structural search plays a key role in IPM, we acknowledge that scientists are not as systematic or as thorough when they build models. Instead, they focus on one or at most a few structures and are quite conservative when suggesting revisions or new approaches. Nevertheless, scientists do not construct sets of equations from scratch but build models by considering a similar space of model fragments and adding those to a partially complete structure. IPM differs from scientists in another important respect: it sometimes suggests implausible explanations. Earlier work by Pazzani et al. (2001) suggests that scientists will reject implausible models produced by discovery systems, urging us to identify further constraints on the space of model structures. We hold that scientists call on theory-level constraints to quickly rule out implausible models.

3 Constraints on Process Models

The previous section emphasized the critical role of knowledge in IPM. By adding generic processes and generic entities, we increase the knowledge available to the program and widen the scenarios that it can address. A long trend of thought in cognitive science and artificial intelligence suggests that adding knowledge (e.g., heuristics) reduces search. However, as we add generic processes to IPM, the program not only carries out more search but also considers more implausible models. This observation brings to light the distinction between model fragments and constraints on their combination. In practice, scientists work with both types of knowledge although the components tend to be more visible or explicit than the constraints on their applicability.

So far, we have viewed generic processes and entities as atomic model fragments. This perspective downplays their own role in reducing search. To illustrate, imagine applying an equation discovery system to generate the population dynamics model in Table 2. The system would have to

search through a space of equations bounded by length, for a combination of constants, variables, and operators that describe observations. To reduce the difficulty of that task, one could introduce larger chunks of knowledge and develop a library of terms to swap in and out of the equations. Possessed with enough knowledge, one could even limit the equations to assemblies of a small set of terms, which would reduce the branching factor of the search. Among their other benefits, the generic processes in IPM play this role.

In practice, the generic processes severely restrict the form that a system of equations can take. If we consider the final model to be a set of differential and algebraic equations,[3] then the processes constrain possible solutions by grouping the equation elements that must appear together. Recalling the population dynamics domain, a predation process that leads to more predators without diminishing the prey is parasitism, which is conceptually different from predation and is an implausible process in fox–rabbit interactions. The parameter ranges that appear in both the generic entities and processes further restrict the models by limiting the quantitative search to theoretically promising regions within that space.

The above constraints exist entirely within the entity and process definitions, but other aspects of the process modeling language limit relationships among components. Specifically, a process's entity roles have type requirements for the objects that fill them. This knowledge prevents models where rabbits eat foxes or elephants photosynthesize. For IPM, the type constraints reduce the number of bound processes which has direct influence on the size of the search space and the maximum complexity of a model. We distinguish this type of knowledge as compositional constraints between entities and processes.

Our experience has shown that the above constraints insufficiently rule out implausible models. Often IPM would confront us with models that lacked crucial components such as light dependency for plant growth or that included incompatible processes such as irreversible and reversible reactions between the same two chemical species. These aberrations suggested the introduction

[3]To avoid misinterpretation, we note that the explanatory content of the model stems from its relationship to scientific concepts and not from the equations themselves. Equations without a theoretical interpretation provide a description of system dynamics, but one should avoid calling them explanations.

of constraints among processes. Through these, we could declare generic processes as necessary, mutually exclusive, or contingent and further limit the candidates considered by an inductive process modeler. A program called HIPM (Todorovski et al., 2005), which uses a library of generic processes organized in a hierarchy, captures our original approach to this problem. We report some results with this system in Section 5. Although this system reduced the amount of search, scientists found its library formalism unwieldy for knowledge representation and search control.

4 Constraints in SC-IPM

The advantages of constraints and the problems with HIPM's representation led us to a new approach. We knew that the modularity of the generic processes lends flexibility to library design and that imposing a rigid structure would compromise that advantage. Therefore we designed equally modular constraints that would complement the process formalism. We implemented these constraints in a system called SC-IPM as an acronym for "Satisfying Constraints to Induce Process Models" that adjusts its candidate structures to fit with this new knowledge.

Our initial implementation of SC-IPM supports four types of constraints each of which encodes a basic relationship among generic processes. One of these is the *necessary* constraint which asserts that at least one instantiation of the included generic processes must appear in a model. For example, this type of constraint could ensure that a predation model includes an exponential loss process or an instantiation of a particular predation process (e.g., Holling type I). The *always-together* constraint extends *necessary* and declares that if a model instantiates one of the generic processes within a group then it must instantiate all of the others. This relationship is useful in ecosystem models that include nutrient-limited primary production where the nutrient-limited growth process should appear if and only if there is a corresponding nutrient absorbtion process.

SC-IPM's other two constraints are closely related to each other. The first of these, *at-most-one*, ensures a mutually exclusive relationship among a set of generic processes. This addition makes alternative functional forms more practical. For example, one can preclude ecosystems

models where the same two species engage in both symbiotic and parasitic interactions. The final constraint, which we call *exactly-one*, combines *at-most-one* and *necessary* to define a mutually exclusive group of generic processes where one of the alternatives must appear in the model. Table 4 gives an example process library for ecosystem models, omitting the details of the generic processes to focus on the constraints.

As described SC-IPM's four constraints apply to models as a whole, so the constraint "growth_-alternatives" in Table 4 limits models to a single instantiation of one of its three members. This limitation is overly restrictive if there are multiple producers. In response, we include an entity-based syntax that broadens the expressiveness of each constraint. For example, the modification:

constraint growth_alternatives

type *exactly-one*

processes $exponential\_growth(P)$, $logistic\_growth(P)$, $limited\_growth(P)$

alters growth_alternatives from Table 4 to signify that a model must have exactly one of the alternatives *for each* producer entity. Likewise, the nutrient_limited_growth constraint ensures that limited_growth is always accompanied by an instantiation of the nutrient_limitation process for each available nutrient.

To develop a program that respects these constraints, we modified IPM's structure generator. Specifically, we turned to approaches for solving constraint satisfaction problems. Methods from this area of research typically begin by translating statements in higher order logics to a propositional representation. For SC-IPM, this takes place during the creation of the bound processes, each of which serves as a variable in a Boolean sentence. Given a sentence, the constraint satisfier attempts to find values for all the variables that will make the sentence true. If a variable is *false*, then it must not appear in the candidate model, if *true*, then it must appear, otherwise it is considered *free* and its inclusion does not affect the plausibility of the model structure. For example, given the bound processes $p_1, \ldots, p_5$, a set of constraints could lead to the Boolean sentence $p_1 \wedge \neg p_2 \wedge (p_4 \vee p_5)$. In this case, p_1 must appear in all plausible models, p_2 can never appear, at least one of p_4 and p_5 must appear, and p_3 is a free variable, which leads to six valid structures.

Table 4: A SC-IPM library for ecosystem models with some of the details removed.

generic entity producer

generic entity grazer

generic entity nutrient

generic entity light

generic process exponential_growth
 relates $P\{producer\}$

generic process logistic_growth
 relates $P\{producer\}$

generic process limited_growth
 relates $P\{producer\}$, $N\{nutrient\}$

generic process ivlev
 relates $P\{producer\}$, $G\{grazer\}$

generic process ratio_dependent
 relates $P\{producer\}$, $G\{grazer\}$

generic process nutrient_limitation
 relates $P\{producer\}$, $N\{nutrient\}$

generic process light_limitation
 relates $P\{producer\}$, $L\{light\}$

generic process exponential_loss
 relates $P\{producer\}$

generic process degradation
 relates $G\{grazer\}$

constraint growth_alternatives
 type *exactly-one*
 processes $exponential\_growth$, $logistic\_growth$, $limited\_growth$

constraint predation_alternatives
 type *exactly-one*
 processes $ivlev$, $ratio\_dependent$

constraint nutrient_limited_growth
 type *always-together*
 processes $limited\_growth$, $nutrient\_limitation(N)$

constraint optional_light_limitation
 type *at-most-one*
 processes $exponential\_growth$, $logistic\_growth$, $light\_limitation$

constraint mandatory_loss
 type *necessary*
 processes $exponential\_loss$, $degradation$

After SC-IPM transforms the constraints into a propositional sentence, it can use any of the available methods for constraint satisfaction. These approaches employ either traditional heuristic search (Davis et al., 1962) or local search (Selman et al., 1996). To be efficient in time, both technologies expect the logical sentence to be in conjunctive normal form (CNF). [4] Unfortunately, transforming an arbitrary logical sentence into CNF can result in an exponential increase in clauses particularly in the case of mutually exclusive relationships. Nevertheless, this strategy lets us generate only those model structures that satisfy the given constraints. An alternative worth mentioning is the translation of the sentence into disjunctive normal form (DNF). [4] In this case, external algorithms are unnecessary because each conjunctive clause defines a space of valid models. As with the previous case, DNF conversion can also result in an exponential increase in clauses.

A third strategy for developing a structure generator employs a backtracking algorithm as used in Prolog. The advantage of this approach is that the logical sentence may take an arbitrary form, so one can use this as a final resort when CNF and DNF conversion are infeasible. This or any of the other strategies are viable candidates for SC-IPM's structure generator and are all well supported by long histories of research. Currently the system the backtracking approach by default.

Each of these strategies differs from those in IPM. Although SC-IPM can still carry out beam search, we have added a new stage to the structure generator. In the language of constraint satisfaction, the system first finds a valid variable assignment to the logical formula and then explores the space of models defined by that assignment and the free variables. The first step yields a plausible model structure with a minimal set of processes, and the second step adds processes to that structure by including subsets of the unconstrained components.

In this section, we introduced four constraints among processes that let one restrict an inductive process modeler to plausible explanations. We also discussed how to alter IPM's solution generator to take those constraints into account. Next we describe experiments that show the effects of further constraints on the problem space.

[4] Conjunctive normal form is a sequence of disjunctive (i.e., logical or) clauses combined with the logical and operator. Disjunctive normal form inverts the relationship and is a disjunction of conjuncts.

5 Experiments in Inductive Process Modeling

Constraints on model structure have played a role in inductive process modeling since its first appearance (Langley et al., 2002), but they have not historically taken the form of declarative, scientific knowledge. The earliest programs supported mutually exclusive generic processes, limits on the size of the model, and limits on the number of instantions of a generic process. An initial focus on exhaustive search and the need to control the amount of search drove the development of these capabilities. The clearest example of this need appears in a recently published article (Bridewell et al., 2008), and we recount it here.

The modeling task involves predicting the water level in Denmark's Ringkøbing Fjord. To guide model development, we had the trajectory for the observed water level recorded hourly for roughly one year, which the model should predict, and trajectories for the water level of the open sea, the influx of fresh water, wind velocity and direction, and the number of gate segments that were opened to allow water exchange between the estuary and the open sea. IPM began with a partial solution and had to provide explanations for the gate and wind influences from a selection of 16 generic processes. The size of the library led to over 65,000 model structures, which amounted to several days of computation.

To reduce search, we forced IPM to include two of the processes and introduced mutually exclusive groups for some of the wind-related processes. These constraints were conceptually reasonable and ultimately reduced the number of candidate structures from 2^{16} to 1,280. While we certainly welcomed this result, we were concerned that the reduced space may have affected model accuracy. To test this, we compared training accuracy with results reported by Todorovski (2003) for the LAGRAMGE equation discovery system, which searched a larger model space. We reported a root mean squared error (RMSE) of 0.052 and a coefficient of determination (r^2) of 0.421 for a model trained over the first half of the data set. In comparison, LAGRAMGE's model, which was trained over twice as much data had an RMSE of 0.059 and an r^2 of 0.434.[5] The difference in

[5]The root mean squared error is the average absolute error of each prediction. The coefficient of determination for IPM is the square of the correlation coefficient, so values range between 0 and 1 with larger values indicating that the model explains more of the observed variance.

scores was minimal when compared to the savings in search and reflects much of our experience with introducing constraints to IPM.

To better understand how constraints affect model search, we compared IPM to a program with constraints similar to those in SC-IPM (Todorovski et al., 2005). These experiments used beam search and focused on both the r^2 scores of the best models and the number of models considered by each system. The first set of experiments were on synthetic data generated from an ecosystem-style model that included four entities and nine processes. In all cases, the structural constraints led to at least a seven-fold reduction in search without any loss in quantitative accuracy. Moreover, the constraint-aware program accurately reconstructed the original model structure in a majority of the trials.

To further test the effectiveness of a constrained library, we applied both programs to two collections of scientific data. The first of these was from predator–prey experiments between microscopic species (Veilleux, 1979), and the second was of phytoplankton dynamics in the Ross Sea (Arrigo et al., 2003). For these domains, the overall reduction in search varied between seven-fold and nineteen-fold much as we had expected, but the error measures told different stories. In the Ross Sea domain, the constraint-aware system substantially outperformed IPM in accuracy with r^2 values ranging from 0.06 to 0.16 points higher. In contrast, the constraints reduced r^2 in the predator–prey domain by more than 0.2 points in each case. Curious about this result, we looked more closely at the models that each system produced. Doing so revealed that none of the models produced by IPM were plausible because they either included mutually exclusive processes or excluded necessary ones. Our tentative conclusion was that the library of generic processes either lacked the appropriate functional forms or that the parameter ranges were too restrictive. Subsequent studies with the same or similar data sets suggest the latter.

More recently, we reported on another system that operated in a population dynamics domain (Bridewell & Todorovksi, 2007). For these experiments, the program induced models from three predator–prey data sets. One group of models came from an unconstrained search, while three other groups came from searches limited by different sets of constraints. In the nine cases that

used constraints, the knowledge reduced the size of the search space by a factor of ten without sacrificing overall model accuracy. We also noted that the models in these smaller search spaces had higher average accuracy than those from the unconstrained space. The corresponding shifts in the error distributions of the models also suggested that the constraints removed several poorly performing structures.

We presented these results to communicate the value of structural constraints. Although we left out many of the details, each case shows that the addition of constraints among processes reduces search substantially without harming model accuracy. We expect that the final version of SC-IPM will reflect these findings, and we have since turned our attention from the value of structural constraints to the role of different types of constraints and their corresponding representations.

6 Concluding Remarks

Inductive process modeling is a recent contribution to the literature on computational scientific discovery. Inspired by previous work, we investigated this task with a program that searches through a problem space. Guided by time-series data, that system composed candidate explanations from a library of generic processes that delineated its search space. Unfortunately this approach entertained scientifically implausible candidates which may fit the data quite well. Regardless of their quantitative accuracy, these models poorly explain the phenomena of interest and should be excluded from consideration.

This need to further restrict the search space led to the discovery of constraints on model structures. This type of knowledge forms a key part of scientific theories that has not been acknowledged in the literature and that helps focus a system's attention on plausible models. To take advantage of these constraints, we developed a language for their representation and incorporated it into a new program called SC-IPM.

Research on inductive process modeling builds on a long line of work in fields as varied as equation discovery and qualitative physics. We were particularly influenced by the research on LA-

GRAMGE (Todorovski, 2003) which discovered systems of equations from time-series data much like IPM. Whereas our approach uses scientifically plausible processes to direct a search for explanatory models, LAGRAMGE uses a context-free grammar to define a space of differential and algebraic equations. Our focus on processes as an important type of scientific knowledge stems from Forbus's characterization of qualitative physics (Forbus, 1984) and other work from that area has certainly influenced our own.

Although the value of search constraints is well known, their role in model formation has been ambiguous at best. MECHEM (Valdés-Pérez, 1995), which takes knowledge of chemistry in the form of solution constraints as input and produces chemical reaction mechanisms as output, is closest in spirit to the reported research. As in SC-IPM, that system's constraint language was modular, but the input was problem specific in that a scientist would enter constraints relevant to a particular chemical reaction. In contrast, SC-IPM focuses on theory-level constraints and not problem-level ones. This distinction is important in the same sense that scientific theory differs from data description—the knowledge is general enough to transfer across modeling tasks.

The importance of theory-level constraints and the little attention that they have receieved leaves the door open for an abundance of future research. We certainly intend to follow MECHEM's example and include problem-level constraints into SC-IPM so that we can refine the space of candidate models based on scenario specific considerations. Additionally, we are dedicated to new approaches for learning modeling constraints. We have already published some results in this direction (Bridewell & Todorovksi, 2007; Bridewell et al., 2007) and are currently investigating instances when constraints generalize across problems and across task domains.

In this paper, we highlighted the role of model fragments and model constraints in scientific discovery. We believe that the distinction between these two types of knowledge opens new avenues of research for the development and investigation of discovery systems and the analysis and acquisition of knowledge. We eagerly anticipate further revelations of new types of scientific knowledge that play a key role in the discovery process.

Acknowledgments

This research was supported by Grant Number IIS-0326059 from the National Science Foundation. We thank Matt Bravo and Ljupčo Todorovski for their contributions to the IPM and SC-IPM systems and their role in developing the constraint language.

References

Arrigo, K. R., Worthen, D. L., & Robinson, D. H. (2003). A coupled ocean–ecosystem model of the Ross Sea: 2. Iron regulation of phytoplankton taxonomic variability and primary production. *Journal of Geophysical Research-Oceans*, *108*, 24–1–24–17.

Bridewell, W., Borrett, S., & Todorovski, L. (2007). Extracting constraints for process modeling. *Proceedings of the Fourth International Conference on Knowledge Capture* (pp. 87–94). Whistler, BC: ACM Press.

Bridewell, W., Langley, P., Todorovski, L., & Džeroski, S. (2008). Inductive process modeling. *Machine Learning*, *71*, 1–32.

Bridewell, W., Sánchez, J. N., Langley, P., & Billman, D. (2006). An interactive environment for the modeling and discovery of scientific knowledge. *International Journal of Human–Computer Studies*, *64*, 1099–1114.

Bridewell, W., & Todorovksi, L. (2007). Learning declarative bias. *Proceedings of the Seventeenth Annual International Conference on Inductive Logic Programming* (pp. 63–77). Corvallis, OR: Springer.

Cohen, S., & Hindmarsh, A. (1996). CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics*, *10*, 138–143.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*, 394–397.

Feigenbaum, E. A., Buchanan, B. G., & Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. *Machine Intelligence*, *6*, 165–190.

Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Klahr, D. (2002). *Exploring science*. Cambridge, MA: MIT Press.

Kulkarni, D., & Simon, H. A. (1990). Experimentation in machine discovery. In J. Shrager and P. Langley (Eds.), *Computational models of scientific discovery and theory formation*, 255–273. San Mateo, CA: Morgan Kaufmann.

Langley, P., Sánchez, J. N., Todorovski, L., & Džeroski, S. (2002). Inducing process models from continuous data. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 347–354). Sydney.

Langley, P., Simon, H. A., Bradshaw, G. L., & Żytkow, J. M. (1987). *Scientific discovery*. Cambridge, MA: MIT Press.

Mahidadia, A., & Compton, P. (2001). Assisting model-discovery in neuroendocrinology. *Proceedings of the Fourth International Conference on Discovery Science* (pp. 214–227). Washington DC: Springer.

Pazzani, M., & Flowers, M. (1990). Scientific discovery in the layperson. In J. Shrager and P. Langley (Eds.), *Computational models of scientific discovery and theory formation*, 403–435. San Mateo, CA: Morgan Kaufmann.

Pazzani, M. J., Mani, S., & Shankle, W. R. (2001). Acceptance by medical experts of rules generated by machine learning. *Methods of Information in Medicine*, *40*, 380–385.

Selman, B., Kautz, H., & Cohen, B. (1996). Local search strategies for satisfiability testing. In *Cliques, coloring, and satisfiability*, 521–532. Providence, RI: American Mathematical Society.

Todorovski, L. (2003). *Using domain knowledge for automated modeling of dynamic systems with equation discovery*. Doctoral dissertation, Faculty of computer and information science, University of Ljubljana, Ljubljana, Slovenia.

Todorovski, L., Bridewell, W., Shiran, O., & Langley, P. (2005). Inducing hierarchical process models in dynamic domains. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 892–897). Pittsburgh, PA.

Todorovski, L., & Džeroski, S. (1997). Declarative bias in equation discovery. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 376–384). Nashville, TN: Morgan Kaufmann.

Valdés-Pérez, R. E. (1995). Machine discovery in chemistry: New results. *Artificial Intelligence*, *74*, 191–201.

Veilleux, B. G. (1979). An analysis of predatory interaction between *Paramecium* and *Didinium*. *The Journal of Animal Ecology*, *48*, 787–803.

Zupan, B., Bratko, I., Demšar, J., Juvan, P., Kuspa, A., Halter, J. A., & Shaulsky, G. (2007). Discovery of genetic networks through abduction and qualitative simulation. In *Computational discovery of scientific knowledge*, 228–247. Berlin: Springer.

Żytkow, J. M. (1999). Model construction: Elements of a computational mechanism. *AISB'99 Symposium on AI and Scientific Creativity*. Edinburgh, Scotland.

Żytkow, J. M., Zhu, J., & Hussam, A. (1990). Automated discovery in a chemistry laboratory. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 889–894). Boston: AAAI Press.