Exploiting Problem Data to Enrich Models of Constraint Problems

Martin Michalowski[1], Craig A. Knoblock[1], and Berthe Y. Choueiry[1,2]

[1] University of Southern California, Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292 USA
{martinm,knoblock}@isi.edu
[2] Constraint Systems Laboratory, University of Nebraska-Lincoln
256 Avery Hall, Lincoln, NE 68588 USA
choueiry@cse.unl.edu

Abstract. In spite of the effectiveness of Constraint Programming languages and tools, modeling remains an art and requires significant involvement from a CP expert. Our goal is to alleviate the load of the human user, and this paper is a first step in this direction. We propose a framework that enriches a 'generic' constraint model of a domain area with a set of constraints that are applicable to a particular problem instance. The additional constraints used to enrich the model are selected from a library; and a set of rules determines their applicability given the input data from the instance at hand. We address application domains where problem instances slightly vary in terms of the applicable constraints, such as the Building Identification (BID) problem and we use Sudoku puzzles as a vehicle to illustrate the concepts and issues involved. To evaluate our approach, we apply it to these domains, using constraint propagation on the generic model to uncover additional information about the problem instance. Our initial results demonstrate our ability to create customized models whose accuracy is further improved with the use of constraint propagation. We also discuss results obtained by solving the newly inferred models, showing that the combination of rule-based constraint inference and constraint propagation is a step towards precise modeling. Finally, we discuss domains that can benefit from our approach (e.g., timetabling and machine translation) and present directions for future work.

1 Introduction

Constraint Programming (CP) has been shown to be an effective paradigm for modeling and solving combinatorial problems [1–4]. However, the modeling step remains an art, requiring a CP expert to specify the variables, their domains, and the set of constraints that govern a particular *Constraint Satisfaction Problem* (CSP). Further complicating the modeling process is the need to specialize a given constraint model for all variations found within the problem class. As a first step to automating the modeling process and alleviating the load placed on

the human user, we propose to enrich the generic constraint model[3] of a problem class by adding to it the constraints that apply to a given problem instance. The additional constraints are inferred from the input data of a problem instance.

The embedded information that we exploit is a set of instantiated variables (i.e., variable-value pairs) which we call *data points*. Our framework tests the features of these data points in order to select, from a library of constraints, those constraints that should be added to the generic constraint model of the problem. We reduce the load on a domain expert by limiting their involvement to defining the rules, constraints and features describing the data points. While such tasks still require human expertise, we contend that (1) this effort is leveraged over time and (2) storing the individual models for all problem variations is infeasible in practice. As such, we use this expert knowledge along with the information found in the problem description to generate a customized problem model.

When automatically generating constraint models, it is difficult to know when the 'right' model for a problem instance is reached. Hence, our proposed method to modeling attempts to *approach* the right model of a problem. Subsequently, it is possible that we may incorrectly add to the generic constraint model constraints that do not apply. To contend with incorrect inferences, we augment our inference method with a backtrack search over the set of possible models. Assuming that given problem instances are solvable allows us to detect inconsistencies in the model (e.g., annihilation of a variable's domain) to determine that a model is incorrect and that the search requires backtracking. The real-world nature of the problem domains we consider (Building Identification (BID) [1] and Sudoku) requires the existence of a solution for all instances, making our assumption valid. We believe this assumption holds for other application domains, and our intention is to apply this framework to such domains.

This paper is structured as follows. Section 2 motivates our approach. Section 3 provides a general definition of our framework with examples of the introduced concepts from the BID and Sudoku problem domains. Section 4 reports results of experiments on both the BID problem and Sudoku puzzles. Section 5 relates our contribution to existing approaches in modeling and CSPs. Section 6 demonstrates the generality of our approach by identifying various settings where it is desirable, and identifies directions for future research.

2 Motivating Example

Consider the problem of mapping postal addresses to buildings in satellite imagery using publicly available information, defined as the Building Identification (BID) problem [1]. The input here is a bounding box that defines the area of a satellite image, buildings identified in the image, vector information that specifies streets in the image, and a set of phone-book entries for the area. Using the geospatial characteristics of addressing in the world, the task is to return a set of possible address assignments for each building. When presented with a

[3] The basic set of constraints that represent the general characteristics of the problem.

user query, the satellite imagery provides the buildings that need to be assigned addresses. Not knowing which streets each building could lie on requires the use of vector data to determine this information. Finally, the phone book entries provide a set of addresses that are known for the area and must appear in the final solution. To assign address labels to the buildings in the image requires the combination of the gathered and induced information with the addressing characteristics for the area. Casting the BID problem as a CSP is an effective method for combining the various sources of information to find sets of assignments that adhere to the addressing constraints of the area in the satellite image [1], thus identifying a new application domain for the CP research community.

To be useful as a web application accessed online, this application needs to contend with the slight addressing variations found in cities throughout the world. For example, some cities adhere to a block-numbering scheme where addresses increment by a fixed factor (i.e., 100 or 1000) across street blocks while other cities do not. The direction in which addresses increase also varies, in some cities this occurs to the east while in others it is to the west. Additionally, the globalization of addressing across continents ensures that some general guidelines are followed, but this standardization is typically met with regional/cultural customization such as the red/black numbering in Europe or the block numbering in the US. Therefore, to expand this application to support unseen addressing characteristics requires the addition of new constraints. Finally, some cities such as El Segundo California (used in the experiments reported in [1]) are characterized by non-monotonic addressing where building numbers increase to the west in one part of town and to the east in another. Therefore, a model representing cities such as El Segundo needs to contain both the constraints representing its specific addressing characteristics and the *context* in which they apply.

The creation, storage, and maintenance of individual constraint models, for all cities, that account for all of the applicable addressing constraints is an unrealistic and formidable endeavor. However, the work required of the expert to define constraints that capture all of the characteristics of addressing seen to date is relatively simpler and more manageable. Moreover, combining this expert knowledge with known building addresses provided by public sources such as gazetteers[4] allows the web application to dynamically build a constraint model of the area of interest and makes this application a more realistic proposition.

Another example where generic models gain to be enriched to adapt to a specific context is Sudoku. Sudoku is a logic-based placement puzzle similar to a Latin Square. It has been shown that solving basic Sudoku puzzles is NP-complete [5] and can be done using CP [4]. Interestingly, slight variations of the basic Sudoku puzzle are played throughout the world (see Figure 1), where each variation adds to the constraints that define the basic puzzle. For example, the diagonal Sudoku adds an 'all-different' constraint for each of the diagonals; and the Samurai Sudoku combines five different puzzle variations in a quincunx arrangement. A generic CSP model represents only weakly all these variations.

[4] A geographical dictionary of building addresses and their spatial coordinates.

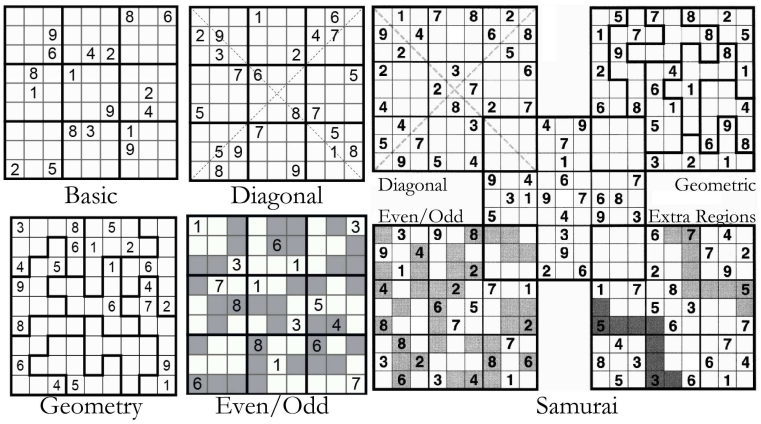

Fig. 1. Variations of sudoku puzzles.

Hence, to solve a given Sudoku instance, an automated system must identify the constraints and their context for the given instance.

We realize that it is highly unlikely that a user would be given a set of pre-filled cells and the vague description 'this is some Sudoku variant' and be left to work out what puzzle they are supposed to solve. However, Sudoku puzzles serve *as a vehicle to illustrate the ideas and concepts behind our proposed framework.* The BID problem is more complex in nature and harder to illustrate in the scope of a single paper. Yet the issues encountered in this problem can be quite naturally mapped to Sudoku puzzles, which are more widely recognized and in our view easier to understand. As such, we analyze both domains in this paper to help facilitate the reader's understanding of our ideas.

3 Constraint-Inference Framework

In this section, we discuss the problem of inferring applicable constraints from input data. We describe our constraint inference framework and present general techniques for constraint selection. We introduce the use of constraint propagation in an iterative way to improve the inference capabilities of this framework. Throughout this section, we use the Sudoku puzzle and the BID problem to illustrate the introduced concepts. As mentioned in Section 1, a prevailing assumption being made is that all problem instances are solvable. This assumption holds in the BID problem because all buildings in the world must be assigned a unique address and in the Sudoku domain because all of the problem instances we consider are well-formed (i.e., each instance has one unique solution).

3.1 General framework

Figure 2 informally defines our general inference framework. C_G is the set of constraints on the generic model, and apply to *all* instances for a problem class. Examples in the BID problem are all known addresses have to be assigned to a building, corner buildings are only assigned to one street, and in Sudoku the AllDiff constraints on the rows and columns.

Given:

- a generic CSP model of a particular class of problems containing a set of generic constraints C_G,
- a library with a set of constraints C_L applicable to this class, and
- a set of data points $\{D_i\}$, where each data point D_i has a set of features,

a *Constraint-Inference Framework* is a set of rules $\{R_m\}$ along with an algorithm (i.e., inference engine) that operates on these rules. The rules map the features of D_i to the constraints $\{C_L\}$ of the library, indicating which constraints govern a problem instance.

Fig. 2. Constraint-inference framework.

Data Points: Generally speaking, data points $\{D_i\}$ can be any elements of the input data, such as information that instantiates some of the CSP variables of the generic CSP model. These data points are described using a set of domain-specific features defined by a domain expert. In the Sudoku domain, the data points are the cells filled with numbers defined by the features: Filled-in Number, Row, Column, Region, and Cell Color. In the BID problem, data points are landmark buildings defined by: ID, Address Number, Street Name and Orientation, Side of Street, Latitude, Longitude, Block Number, and Street Ordering.
Constraint Library: The constraint library consists of a set of constraints C_L that represent the additional characteristics introduced by variations of problem instances within a problem class. An individual constraint captures a certain characteristic that is only applicable to *some* problem instances. The block numbering in the BID problem and the AllDiff on the diagonals in Sudoku are two examples of constraints found in the library. This library serves as the repository from which our framework selects applicable constraints.
Applicability Rules: The rules in our framework are predefined by a domain expert, similar to the use of expert modules in PROVERB [3]. They are separated from the constraints in the library and act as a link between the constraints and the features defining the data points. The differences between rules and constraints are as follows: A constraint's scope is over a subset of the variables in the model. Therefore, a constraint is satisfied given a particular set of variable-value pairs (i.e., assignments of values to variables). The rules are triggered by a predicate function over the *features* of the variables in the problem instance (the head of the rule). When this function is true, the constraint (whose scope is the variables in the head of the rule) is asserted. The generic constraints for the given problem class are always included in the model. An additional benefit of using rules is that multiple rules can map to a single constraint, allowing higher levels of inference for a constraint. Our rule language supports any programmable predicate expressions and rules are defined using the following format:

1. IF ⟨test points' features for rule applicability⟩
2. IF ⟨test points' features for constraint applicability⟩
3. THEN ⟨add positive support to constraint⟩
4. ELSE ⟨add negative support to constraint⟩

The first test checks the applicability of the rule, the second that of a constraint from the constraint library. If the second test succeeds, the *positive support* of the constraint is increased, otherwise the *negative support* of the constraint is increased. Finally, as in a classical Expert System architecture, the rules are separated from the inference engine making the inference framework applicable across problem domains. Two sample rules are shown in Figure 3. Applying rules is linear in the number of rules and we use bucketing (see Section 3.2) to improve scalability of testing rules applicability. Furthermore, it is possible to use various modeling or rule languages to represent the applicability rules. However, our contribution lies in using the input data to select the constraints, *independent* of the representation language used for the rules.

BID Applicability Rule: Parity
IF Street(P_1)=Street(P_2) then
  IF [SameParity(Num(P_1),Num(P_2)) $\wedge$ SameSide(P_1,P_2)] $\vee$
    [OppositeParity(Num(P_1),Num(P_2)) $\wedge$ OppositeSide(P_1,P_2)]
      THEN Add positive support for Parity constraint
      ELSE Add negative support for Parity constraint
Sudoku Applicability Rule: AllDiff Row
IF Row(P_1)=Row(P_2) then
  IF Number(P_1)$\neq$Number(P_2)
    THEN Add positive support for All-Diff Row constraint
    ELSE Add negative support for All-Diff Row constraint

Fig. 3. Two examples of applicability rules.

3.2 Selecting constraints

The selection of constraints based on the information found in the problem (data points) is the key contribution of our framework. Previous work in selecting constraints from a library has shown that such an approach is an effective method to modeling CSPs [6, 7]. As such, our framework uses the constraint library as a 'knowledge base' from which we can enrich the generic model. The selection of constraints is a three-step process. First, in order to enhance the performance of testing rule applicability and the scalability of the inference engine, we perform a bucketing [8] of the data points based on feature values. Our evaluation (see Section 4.1) illustrates the benefits of bucketing.

After comparing all data points within each bucket, we obtain a set of supports (positive and negative) for inferred constraints, with some constraints in the library receiving no support. Before selecting which constraints from the constraint library to add to the generic constraint model, all constraints are categorized into one of three categories, *applicable*, *non-applicable*, or *unknown*, based on their respective set of supports. The categorization of constraints is an important step towards determining which constraints to select and add to the CSP model. If a given constraint receives no support, it is classified as *unknown*. For all other constraints, they are categorized as *applicable* or *non-applicable* depending on their *support level*, a function of the positive and negative supports of the constraint that allows the framework to express confidence in its inference.

The enriched constraint model C_{new} is defined as $C_G \bigcup C_a$ where C_G is the set of constraints of the generic model and $C_a \subseteq C_L$ is the set of constraints from the constraint library classified as *applicable*.

In our evaluations (see Section 4), a support level of 1 (the constraint has at least one positive support and no negative support) classified a constraint as *applicable* and constraints with both negative and positive supports were classified as *non-applicable*. *Non-applicable* and *unknown* constraints are not added to the CSP model. This setup enforces a binary classification of constraints and uses a minimum support level, allowing us to test the major ideas present in this framework. Studying the impact of support levels is our next course of action and is discussed in Section 6. Furthermore, it is possible that an incorrect inference can be made and a constraint is incorrectly added to the generic constraint model (due to noisy data or a lack of information from the data points). The elimination of erroneous inferences is important for maintaining high solution precision and techniques presented in Section 3.3 are a means towards this end.

Finally, the *context* in which certain constraints apply must be determined. To begin, we require the user to define a set of conflicting constraints. Because automatically identifying conflicting constraints is NP-hard and may require considering 2^c combinations of constraints (c is the number of constraints in the library), we reduce the initial complexity of the framework by using this predefined set. Subsequently, if two or more inferred constraints appear in the set of conflicting constraints, we exploit semantic information to generate contexts that spatially separate the conflicting constraints. As such, the scope of a conflicting constraint will be limited to variables within one context, while all other constraints will be defined across all contexts. Once a contextualized set of constraints has been determined, the customized model can be generated. This context-inference approach is specific to a given problem domain and we discuss the generalization of the context-inference process in Section 6.

3.3 Improving constraint selection using constraint propagation

A lack of information in the data points of a problem instance may lead to incorrect constraint inferences or not inferring constraints that apply. Given that the constraints in the generic model hold for the problem instance at hand, we propose to apply constraint propagation algorithms on the *generic* constraints in order to make explicit more features of the particular instance. Starting with Arc-Consistency (AC) [9] we gradually move to more discriminating propagation algorithms and inject the newly inferred data points back into the generic model at each step. For the Sudoku domain, we considered AC, Generalized Arc-Consistency (GAC) [10], and Shaving or Singleton Arc-Consistency (SAC) [11], applying these algorithms in sequence for better performance.

The addition of new data points by constraint propagation benefits the constraint-selection process described in Section 3.2. We developed an iterative algorithm where we use constraint propagation to infer new data points and then use constraint inference to augment the generic model. This algorithm takes as input the initial model for the given instance, made up of the generic

constraints C_G and the initial data points of the problem D_i. It then loops, propagating the current set of constraints to infer new data points. When new data points are inferred, the constraint-selection process described in Section 3.2 is re-applied using the new set of data points to select applicable constraints. The newly inferred constraints are then added to the constraint model. At the next step, constraint propagation is applied to both inferred and generic constraints. Our proposed algorithm ends when the process reaches quiescence, yielding the inference of one possible model. While it is possible that alternative models exist, our algorithm returns the first model found.

It should be noted that this algorithm may require backtracking. If a new constraint inference leads to the annihilation of a variable's domain, then backtracking is necessary to eliminate the erroneously inferred constraint. Backtracking would require this algorithm to keep track of which constraints were added at each time step. In this paper, we remove the need for backtracking by only applying constraint propagation to the generic constraints C_G and increasing the strength of propagation for each constraint-inference iteration using AC, GAC, and SAC. Additionally, an augmented model may not exist, at which point our algorithm simply returns the generic constraint model.

4 Experimental Evaluation

We carried out experiments on two application domains: the BID problem [1] and Sudoku puzzles. We present our initial findings and the performance of solving the inferred models for both domains, and we evaluate our constraint propagation techniques (see Section 3.3) in the Sudoku domain.

4.1 Sudoku puzzles

Our evaluation of Sudoku puzzles included the basic puzzle along with four variations: geometry (regions are of irregular shape), diagonal (additional AllDiff constraints apply to the diagonals), odd/even (numbers in colored cells have same parity), and magic (each number inside a polyomino must be no larger then the number of blue cells in the region, along with the two diagonal constraints). The Samurai puzzle was not included because we were unable to freely obtain instances of this puzzle type. The constraint library consisted of the 'defining' Sudoku constraints along with all the additional constraints introduced by each puzzle variation. Data points correspond to filled-in cells as defined in Section 3.1.

We conducted three sets of experiments for each puzzle type, testing our framework on 100 puzzle instances for each puzzle difficulty level.[5] Our initial results are summarized in Table 1, and report recall and precision. Recall is defined as the ratio of the number of correct constraints inferred to the total number of constraints representing the puzzle type, and precision is defined as

[5] Randomly sampled from www.menneske.no/sudoku/eng and www.printsudoku.com/index-en.html

the ratio of the number of correct constraints to the total number of constraints inferred. $|C_{new}|$ corresponds to the number of constraint types that define each puzzle type.

Table 1. Sudoku: precision and recall of inferred constraints.

		Easy		**Medium**		**Hard**	
	$\|C_{new}\|$	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
Basic	3	1.0	0.88	1.0	0.87	1.0	0.87
Geometry	3	1.0	0.86	1.0	0.88	1.0	0.88
Diagonal	4	0.86	1.0	0.86	1.0	0.85	1.0
Even/Odd	4	1.0	0.93	1.0	0.94	1.0	0.95
Magic	5	(not categorized): Rec.: 0.81, Prec.: 1.0					

In general, we were able to correctly infer all of the constraints. Specifically, the basic and geometry puzzles are nearly the same puzzle except that geometry puzzles have irregular regions, hence the results are very similar for both puzzle types. We were able to infer the three core constraints (as reflected by the perfect recall), but we incorrectly inferred the existence of a diagonal constraint in some puzzle instances, causing a drop in overall precision. This same incorrect inference caused the drop in precision for the even/odd puzzle. The diagonal and magic puzzles were cases where all inferred constraints were correct (perfect precision), but the filled-in cell distribution was such that we were unable to infer a diagonal constraint (an applicable constraint) for all the puzzle instances (a drop in recall). In fact, only 10% of all magic puzzle instances contained enough data points to infer this constraint.

Table 2. Iterative Propagation: precision and recall of inferred constraints.

			Easy		**Medium**		**Hard**	
	$\|C_G\|$	$\|C_{new}\|$	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
Basic	2	3	1.0	0.99	1.0	1.0	1.0	0.99
Geometry	2	3	1.0	1.0	1.0	1.0	1.0	0.99
Diagonal	2	4	0.89	1.0	0.89	1.0	0.88	1.0
Even/Odd	2	4	1.0	0.93	1.0	0.94	1.0	0.94
Magic	2	5	(not categorized): Rec.: 0.81, Prec.: 1.0					

Those less-than optimal results necessitate a more robust approach to selecting constraints. We present in Table 2 results of combining the iterative algorithm described in Section 3.3 with our inference framework. In these experiments we only propagate the row and column AllDiff constraints ($|C_G|$), changing the propagation levels at each iteration. Due to a lack of space we omit the precision and recall measures seen at each iteration of the algorithm. As expected, the precision of the constraint model was improved for puzzle variations where a significant number of new data points were inferred. Specifically, the basic and geometry puzzles saw a significant jump in precision values because the new data points eliminated most of the erroneously inferred diagonal constraints. Interestingly, a small number of new data points in the diagonal puzzle helped improve the recall by providing enough information to allow the inference

of the diagonal constraint in additional instances. Not surprisingly, the even/odd and magic puzzle variations saw no significant change in their respective models because the propagation of constraints led to very few new data points. A drop in precision for the hard even/odd puzzles is caused by the incorrect inference of the diagonal constraint for some problem instances, caused by the newly inferred data points. In general, puzzle variations that are highly constrained (i.e. even/odd, magic) require propagation of more than just generic constraints. A solution is thus to propagate the set of currently inferred constraints at each iteration of the algorithm and we discuss this approach in Section 6.

Table 3. Sudoku problem results with an inferred constraint model.

	Easy		**Medium**		**Hard**	
	% solved	% one sol.	% solved	% one sol.	% solved	% one sol.
Basic	99%	100%	100%	100%	99%	100%
Geometry	100%	100%	100%	100%	99%	100%
Diagonal	100%	57%	100%	56%	100%	53%
Even/Odd	69%	100%	74%	100%	76%	100%
Magic	(¬ categorized): % solved: 100% % one sol.: 10%					

Finally, we evaluated the performance of solving the inferred model and present our results in Table 3. The results report the percentage of problem instances for each puzzle type and difficulty level that could be solved using the inferred constraint model, and the percentage of the solved puzzle instances that returned a single solution. As these results show, the basic and geometry puzzles are almost all solved, except for two puzzle instances that were represented with an over-constrained model (containing an incorrect inference of the diagonal constraint). For the diagonal and magic puzzles, we were able to solve all puzzle instances but a significant number of inferred models were under-constrained, leading to a lower percentage of instances with only one solution. Finally, the incorrect inference of the diagonal constraint in the even/odd puzzles led to a lower percentage of solved instances. As these results show, as a first step the inferred models are quite representative of the puzzle instances. However, we need to further explore support levels as a means to eliminate erroneous inference and adjust our iterative algorithm to provide additional data points for instances where few data points are inferred (see Section 6).

4.2 BID problem

For the BID domain, our evaluation included the cities: El Segundo California, San Francisco CA, Downtown Los Angeles CA, and Boulder Colorado, chosen for their unique characteristics. We varied the set of landmark data-points (defined in Section 3.1) by using different public data sources. Each source provided a different set of data points varying in number and distribution within the area of interest. Results are summarized in Tables 4 and 5.

We ran three sets of experiments in El Segundo (Table 4). First, we collected from a Property Tax website a set of 38 well-distributed data points, all of

Table 4. El Segundo: inferred constraints.

	Constraints		
Data Points	**Parity**	**Block** k=100	**Ascending**
38 data points west of Main St.	✓	✓	✓ East-West & North-South
1650 geocoded points	✓	✓	✓ Context 1: East-West & North-South ✓ Context 2: East-West & North-South
20 USGS gazetteer data points	✓	✓	✓ Context 1: East-West & North-South ✓ Context 2: North-South × Context 2: East-West

✓ applicable constraint × non-applicable constraint

which are located to the west of Main St. (referred to as Context 1). This trial served as a sanity check and the results shown in Table 4 conform to the ground truth. The second trial used roughly 1650 points generated by a geocoder web-service. This experiment tested the scalability of the framework along with its ability to determine contexts. From a scalability standpoint, this experiment required no more than 10 seconds to run (but well over an hour without the bucketing mentioned in Section 3.2). The framework also correctly identified the two contexts for the area, and correctly determined and contextualized the sets of constraints. Finally, we utilized all of the gazetteer points (20) from the USGS gazetteer that lay within El Segundo and had an address associated with them. The contexts and all but one constraint were correctly identified. This experiment illustrates the effect that the distribution of the data points has on the results. To further explore the effect of landmark-point distributions in different areas, we ran the inference engine on other parts of the US. The results of our trials are shown in Table 5.

Table 5. Other cities: inferred constraints.

	Constraints		
Area	**Parity**	**Block** k=100	**Ascending**
7 hotels found in Downtown LA	✓	✓	✓NS × EW
16 USGS gazetteer points in San Francisco	✓	✓	✓ NS & EW
16 USGS gazetteer points in Boulder	✓	N/A in this area	✓ EW × NS

✓ applicable constraint × non-applicable constraint

For downtown LA, seven landmark points where derived from an online source of hotels, each with an address and latitude and longitude coordinates. Our framework was able to correctly determine all but one of the applicable constraints. Indeed, there was not enough information in the data to determine in which direction addresses increased on East-West (EW) running streets. Next we considered a subarea of San Francisco CA and Boulder CO. Using 16 and 7 points respectively from the USGS gazetteer, we were able to correctly identify all but one of the applicable constraints for both regions. There wasn't enough information in the data to determine in which direction addresses increased on North-South (NS) running streets in Boulder. These experiments illustrate the importance of data point distribution over the problem instance. We estimate

that, with a perfect distribution of data points, the minimum number needed to correctly identify all of the constraints currently considered is $4 \times n$, where n is the number of contexts. Essentially, we require two points for each street type (North-South or East-West running). While this minimal set of points allows us to select a set of constraints, we remain vulnerable to noise.

Table 6. BID problem case studies.

Case study	**Phone-book completeness**	**Number of...** bldgs	blocks	building-address combinations
NSeg125-c	100.0%	125	4	4160
NSeg125-i	45.6%			1857
NSeg206-c	100.0%	206	7	10009
NSeg206-i	50.5%			4879
SSeg131-c	100.0%	131	8	3833
SSeg131-i	60.3%			2375
SSeg178-c	100.0%	178	12	4852
SSeg178-i	65.6%			2477

We evaluate the quality of the solutions generated when solving the constraint model inferred from 38 data points (see Table 4). We apply this inferred model to areas west of Main St. (defined in Table 6), and solve the model using a customized BID problem solver [12]. The largest region tested previously contained 34 buildings and a single city block [1] and all of our areas represent an increased problem size over that work. The completeness of the phone book indicates what percentage of the buildings on the map have a corresponding address in the phone book. We created the complete phone books using property-tax data, and the incomplete phone books using real-world phone books. The number of building-address combinations is the number of possible combinations of buildings and phone-book addresses. Note that this number is smaller when the phone book is incomplete.

Table 7. BID problem results with an inferred constraint model.

	W/o orientation cons		W/ orientation cons			
	Runtime (sec)	Domain size	Runtime (sec)	Domain size	Runtime reduction	Domain reduction
NSeg125-c	22397.08	1.22	1962.53	1.0	11.41x	1.22x
NSeg125-i	22929.49	6.11	3987.73	4.18	5.75x	1.46x
NSeg206-c	198169.43	1.21	10786.33	1.0	18.37x	1.21x
NSeg206-i	232035.89	7.91	12900.36	4.99	17.99x	1.59x
SSeg131-c	173565.78	1.56	125011.65	1.41	1.39x	1.11x
SSeg131-i	75332.35	12.56	17169.84	3.92	4.39x	3.20x
SSeg178-c	523100.80	1.41	284342.89	1.31	1.84x	1.08x
SSeg178-i	334240.61	8.24	62646.91	3.23	5.34x	2.55x
				Average	**8.31x**	**1.68x**

Our results are summarized in Table 7. We present results obtained when the Parity and Ascending constraints (denoted by *orientation cons*) are included and when they are unknown. When these constraints are unknown, the constraint model solved corresponds to the generic model. *Runtime* reports the runtime, in seconds, required to solve the problem, *Domain size* reports the geometric mean of the domain size for a building, *Runtime reduction* and *Domain reduction* report the factor by which the average domain size and runtime were reduced when using the customized model. As our results show, the use of a customized constraint model greatly improves the performance of the solver. The results show on average a factor of 8.31 improvement in runtime, with some scenarios seeing a reduction by a factor as large as 19. We also see an average factor of 1.68 improvement in domain reduction. As Bayer et al. [12] noted, every building has the correct label in its domain (resulting in a perfect recall), therefore a factor of 1.68 domain reduction results in a large increase in the solution's precision.

5 Related Work

An appealing new application domain for CSPs has been identified by Michalowski and Knoblock [1]. Incorporating our inference engine into this framework will improve the robustness of the BID system. Puzzles are also an interesting domain for constraint programming (CP). The PROVERB system [3] uses a probabilistic CSP approach to solving crossword puzzles. PROVERB uses clue-value pairs to infer themes in crossword puzzles by passing them onto sets of expert modules. These clue-value pairs are analogous to our data points, and the expert modules are similar to our applicability rules. Our work was partially inspired by PROVERB's inference mechanisms. Finally, Sudoku puzzles have also been modeled as a CSP and it has been shown how different known and ad-hoc propagation techniques affect the ability to solve basic Sudoku puzzles of varying difficulty [4]. Combining our inference framework with these propagation schemes would allow Simonis [4] to expand the set of Sudoku puzzles solved from only the 3x3 basic version to variations such as the ones discussed in this paper and others.

Recent work in CP modeling aims at automatically learning constraint networks from data. Colleta et al. [6] automatically learn constraint networks from full solutions (both consistent and inconsistent). Bessière et al. [13] use historical data (solutions previously seen) to learn constraint networks. Finally, Bessière et al. [7] propose a SAT-based version-space algorithm for picking applicable constraints from a library given a set of solutions. All of these approaches are a way to model a problem class without having to explicitly define the constraint model. However, each approach uses full problem solutions to learn the constraint networks. In our work, we do not require full solutions to a problem instance but only a small number of known values (a small partial-solution). Furthermore, our work identifies small variations of similar problem classes while previous work focuses on finding constraint networks for a particular problem class. However, we are investigating extending the work done by Bessière et al. [13] to support the type of constraints seen in our application domains. The work by Lallouet

et al. [14], which employs machine learning techniques to learn open constraints and their propagators, is also an interesting idea we will study in more detail.

Lastly, Colton et al. [15] find redundant constraints for quasi-groups and reformulate basic constraint models of these groups to improve search. Cheng et al. [16] show how ad-hoc global CASE constraints can be customized to construct various constraint models in the STILL-LIFE game. These papers provide insight into optimizing the inferred model by incorporating different types of constraints (i.e. redundant, CASE). Some of these techniques can be applied to the model generated by our framework and could lead to more efficient problem solving.

6 Discussion and Future Work

In this paper, we introduced a framework for enriching constraint models using data specific to a problem instance. Our framework selects applicable constraints and the context within which they apply and adds them to the constraint model of an instance, dynamically building a representative model. This approach reduces the load placed on a domain expert by requiring them to define constraints representing all known characteristics of a problem class (and the rules that map the information in the problem to these constraints) rather then generating individual models representing all possible variations of a problem class. Our initial results demonstrate the feasibility of this approach, and extending our framework to include iterative constraint propagation leads to the generation of more precise constraint models.

Our framework's effectiveness in two domains leads us to believe that it can be applied to others as well. One such domain is syntactic machine translation [17]. In this domain, syntactic transfer rules are derived from bilingual corpa and used to translate documents from a base to a target language. The text in the document being translated could determine a context (what type of document it is) and what constraints apply. This information would allow a translation engine to be optimized at run-time based on the context and the deduced rule set and would enhance the possibility of a more generalized translation engine for performing multiple bilingual translations.

Timetabling [18] is another relevant application domain. Previous timetable assignments (i.e., historical data) can be used to determine the library of constraints used to infer constraint models. When a timetabling system is being applied in a new environment, such as scheduling exams at a new university, we could look at data points from previous schedules and infer the scheduling rules that are in use in the new environment. This inferred set of constraints could then be applied to new timetabling scenarios without a domain expert having to define the applicable constraints. Furthermore, the timetabling application could support scenarios where some schedule assignments are required in the final set of assignments. These required assignments, in conjunction with the library of scheduling constraints, would be used to automatically generate and solve an applicable scheduling (constraint) model. This approach would be analogous to the use of our constraint-inference framework as defined in this paper.

Our future work includes several directions. Firstly, we will evaluate the effectiveness of the constraint propagation methods presented in Section 3.3 on the BID domain. Secondly, we will study strategies for determining the applicability of the constraints given their support levels. We will also generalize the methodology for inferring contexts by studying ways to determine boundaries within the *problem* space other than spatially. A machine learning concept worth pursing with regards to this problem is Support Vector Machines (SVM) [19]. Finally, we envision an offline process that uses a constraint learning technique to populate the constraint library. Towards this end, we will study approaches to learning constraints [13, 14] and determine if they can be applied to our framework to further lessen the role of the domain expert.

Acknowledgments. This research is supported by the Air Force Office of Scientific Research under grant numbers FA9550-04-1-0105 and FA9550-07-1-0416.

References

1. Michalowski, M., Knoblock, C.A.: A Constraint Satisfaction Approach to Geospatial Reasoning. In: AAAI. (2005) 423–429
2. Nadel, B.A.: Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens. IEEE Expert **5**(3) (1990) 16–23
3. Littman, M.L., Keim, G.A., Shazeer, N.: A Probabilistic Approach to Solving Crossword Puzzles. AI Journal **134**(1-2) (2002) 23–55
4. Simonis, H.: Sudoku as a Constraint Problem. In: CP: Workshop on Modelling and Reformulating Constraint Satisfaction Problems. (2005) 13–27
5. Yato, T.: Complexity and Completeness of Finding Another Solutions and its Application to Puzzles. Master's thesis, The University of Tokyo (January 2003)
6. Coletta, R., Bessière, C., O'Sullivan, B., Freuder, E., O'Connell, S., Quinqueton, J.: Semi-automatic Modeling by Constraint Acquisition. In: CP. (2003) 111–124
7. Bessière, C., Coletta, R., Koriche, F., O'Sullivan, B.: A SAT-Based Version Space Algorithm for Acquiring CSPs. In: ECML. (2005) 23–34.
8. Levy, A.: Logic-Based Techniques in Data Integration. In: Logic Based Artificial Intelligence. Kluwer Publishers (2000) 575–595
9. Mackworth, A.: Consistency in Networks of Relations. AI Journal (1977) 99–118
10. Mohr, R., Masini, G.: Good Old Discrete Relaxation. In: ECAI. (1988) 651–656
11. Debruyne, R., Bessière, C.: Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In: IJCAI. (1997) 412–417
12. Bayer, K.M., Michalowski, M., Choueiry, B.Y., Knoblock, C.A.: Reformulating CSPs for Scalability with Application to Geospatial Reasoning. In: CP. (2007)
13. Bessière, C., Quinqueton, J., Raymond, G.: Mining Historical Data to Build Constraint Viewpoints. In: CP: Workshop on Modelling and Reformul. (2006) 1–16
14. Lallouet, A., Legtchenko, A.: Consistency for Partially Defined Constraints. In: ICTAI. (2005) 118–125
15. Colton, S., Miguel, I.: Constraint Generation via Automated Theory Formation. Lecture Notes in Computer Science **2239** (2001) 575–579
16. Cheng, K.C.K., Yap, R.H.C.: Applying Ad-hoc Global Constraints with the CASE Constraint to Still-Life. Constraints **11**(2-3) (2006) 91–114
17. Och, F.J., Ney, H.: The Alignment Template Approach to Statistical Machine Translation. Computational Linguistics **30**(4) (2004) 417–449
18. Lim, A., Ang, J.C., Ho, W.K., Oon, W.C.: UTTSExam: A Campus-Wide University Exam-Timetabling System. In: IAAI. (2002) 838–844
19. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag (1995)