Exploiting Automatically Inferred Constraint-Models for Building Identification in Satellite Imagery

Martin Michalowski[1], Craig A. Knoblock[1],
[1]Information Sciences Institute
Dept. of Computer Science
University of Southern California
{martinm,knoblock}@isi.edu

Ken Bayer[2], and Berthe Y. Choueiry[1,2]
[2]Constraint Systems Laboratory
Dept. of Computer Science and Engineering
University of Nebraska-Lincoln
{kbayer,choueiry}@cse.unl.edu

ABSTRACT

The *building identification* (BID) problem is based on a process that uses publicly available information to automatically assign addresses to buildings in satellite imagery. In previous work, we have shown the advantages of casting the BID problem as a *Constraint Satisfaction Problem* (CSP) using the same generic constraint-model to represent all problem instances. However, a generic model is unable to represent with the necessary precision the addressing variations throughout the world, limiting the applicability of our previous approach. In this paper, we describe the end-to-end process used to solve the BID with a new model-generation technique that uses instance-specific information to automatically infer a representative constraint model of the BID. This inferred model is used by our custom constraint solver to identify buildings in satellite imagery more efficiently and with higher precision than using a single model. We evaluate our approach on El Segundo California, and empirically demonstrate its effectiveness for geographic areas larger than previously tested. We conclude with a discussion of the generality of our approach, and present directions for future work.

Keywords

Geospatial data integration, Knowledge discovery, Modeling

1. INTRODUCTION

The amount of geospatial information on the Web has rapidly increased thanks in large part to the creation of online services such as Google Maps[1] and MSN Virtual Earth.[2] This recent surge in publicly available information has spurred the development of new and innovative data-integration applications. One such application is the building identification framework introduced by Michalowski and Knoblock [15]. This framework enables a system to integrate traditional geospatial data (satellite imagery, vector data, etc.) with non-traditional data (phone books, property tax information, etc.) to identify buildings in a given satellite image. The goal of this framework is to provide a user with the ability to delimit a geographic area of interest and identify the addresses of the buildings found in that area.. The identification process exploits global addressing rules to assign potential addresses to buildings. However, to expand the coverage of our system, the building identification process must not only use global addressing rules but also identify and exploit the local addressing customizations identified around the world, such as block numbering in the US and red/black numbering in Europe where one building is assigned two addresses of different colors.

The work by Michalowski and Knoblock [15] showed that the non-trivial task of integrating publicly available data can be accomplished by casting the BID problem as a *Constraint Satisfaction Problem* (CSP). A CSP is defined by a set of variables $\mathcal{V}$, a set of variable domains $\mathcal{D}$, and a set of constraints $\mathcal{C}$. These constraints are relations defined over subsets of the variables and restrict the allowable combinations of values to variables. An assignment of a value to a variable is defined as a variable-value pair. A solution to a CSP is an assignment of one value for each variable such that all constraints are satisfied. Finally, the task may be to find one solution, all solutions, etc.

Therefore, solving a BID problem instance cast as a CSP entails two key steps: building a constraint model of the problem instance and solving the constraint model. The constraint model plays a vital role in determining the precision of the returned solutions. Without a representative model of a problem instance, it is unrealistic to expect precise results regardless of the effectiveness of the constraint solver. For the BID problem, a good constraint model must account for the various addressing variations encountered throughout the world without relying on a domain expert to manually construct models for all areas of the world. Towards this end, we present a new model-generation technique that exploits the data found within a problem instance to automatically specialize a 'generic' model, thus yielding a representative model of the instance.

Furthermore, the benefits of an accurate model are only fully realized when the solving mechanism takes advantage of the structure and characteristics of a problem instance. To generate a precise solution, the solving component must be flexible in supporting varying problem models. To improve the

[1] maps.google.com
[2] maps.live.com

ACM GIS '07 Seattle, Washington, USA

performance of problem solving, the solver should exploit the structure of the problem and incorporate appropriate heuristics to reduce the explored search space. In this paper we present a customized solver that improves the scalability of previous approaches by allowing constraints to be turned on and off based on the problem instance, exploiting the inherent structure found in the BID problem, and using applicable heuristics to speed up search.

This paper is structured as follows. Section 2 motivates our approach and provides an overview of the BID problem solving-process with an example problem scenario. Section 3 describes our constraint-inference framework, and Section 4 shows how the model of an instance is generated using the inferred constraints. Section 5 describes our specialized solver and Section 6 shows empirically the improvements introduced by our approach. Section 7 relates our contribution to problem modeling and solving and to previously explored methods for building identification. Finally, Section 8 demonstrates the generality of our approach by identifying various settings where it is desirable and identifies directions for future research.

2. THE BID PROBLEM

Our work is motivated by the problem of mapping postal addresses to buildings in satellite imagery using publicly available information, which we introduced as the Building Identification (BID) problem. This problem takes as input 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. The task is to find the set of possible address assignments for each building. In the context of a web application, a typical BID problem scenario is as follows. A user, presented with a map such as a Google map, either selects a specific building in an area of interest and requests the address of the building, or provides an address and requests the buildings that could have this address.

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. In other cities, addresses along East-West running streets increase to the West in one part of town but to the East in another part. Finally, 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.

The creation of individual *models* that account for all of the addressing constraints for each city in the world is an overwhelming and unrealistic task. However, the work required of the expert to define *constraints* that capture all of the characteristics of addressing seen to date is relatively small and manageable. Therefore, we propose a framework in which the constraint model of the area of interest is dynamically built by augmenting the set of *basic* constraints, which form the *generic* constraint model, with those constraints that specify the addressing schema that governs the area of interest. This customized model is then provided to a solver that leverages the model's features to efficiently assign addresses to buildings for the given area. This approach improves the precision of the returned solution when compared to previous work that used the same, generic model to solve all instances of the BID problem.

Our proposed end-to-end process for solving a BID problem instance is shown in Figure 1. Generally speaking, the process of solving a BID problem begins by taking instance-specific data and inferring a representative constraint model of the instance. This inferred constraint model is passed to a model generation component, which creates a full model of the problem. Finally, a CSP solver solves the constraint model by assigning address labels to all of the buildings in the image and returns this set of labels to the user. In Sections 3, 4, and 5 we describe each component in more detail, outlining the challenges present for the corresponding component. Our proposed approach improves upon the process used in our previous work [15] in two ways.

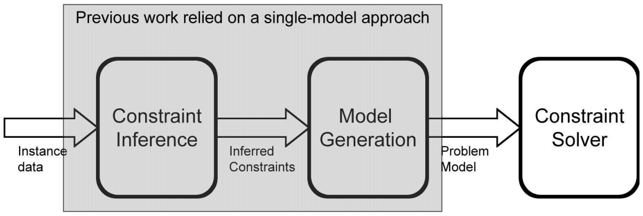

Figure 1: The end-to-end process for solving a BID problem instance

First, the previous work on solving a BID problem assumed that the same generic model represented all problem instances. As we have shown, this assumption is not realistic. Studying various areas throughout the world reveals the existence of different addressing schemes, many of which apply only regionally. Some examples of these schemes are: block numbering seen in certain cities in the US, districting in Venice Italy, red/black numbering seen in some parts of Europe, and others. To deal with these regional variations, we replace the static modeling approach with a dynamic one, represented by the two components within the gray box in Figure 1.

Second, we had previously used a generic constraint solver CPlan [19]. While that solver was useful for quickly establishing the feasibility of the approach and was able to handle small problems, it did not exploit any characteristics specific to the BID problem. Indeed, it was unable to solve problem instances with more than 34 buildings and did not have the flexibility to turn constraints on/off as needed. The new customized constraint solver we have developed for the process exploits the inherent structure found in BID problem instances and, as our results show in Section 6, is able to efficiently solve much larger problem instances then was previously possible. The flexibility of turning constraints on/off allows the overall solving process to handle the addressing variations mentioned previously.

To illustrate our solving process, consider the BID problem example shown in Figure 2. We are presented with a satellite image of an area in El Segundo, California and we would like to identify the buildings in this image. We use the coordinates that define the area's bounding box to retrieve NGA vector data and phone-book entries from Yahoo specific to this area. The vector data allows us to determine the set of potential street assignments for any given building and the phone book provides a list of known addresses. We would like to use this information in conjunction with knowledge of the addressing characteristics for El Segundo to assign address labels to all of the buildings in the area (note that one building may receive multiple labels). We have additional information, obtained from a data source such as a gazetteer, that provides a small set of data points identified by both an address and its latitude and longitude coordinates. We can use this additional information to infer which of the known addressing constraints apply to El Segundo, eliminating the need to provide this information a priori. Using a representative model generated from the above mentioned data, we use a constraint solver to assign address labels to the buildings. At the end of the process, we have provided information not available in any one data source (address labels for buildings given a satellite image of El Segundo) by integrating publicly available data and extracting new knowledge using this integration.

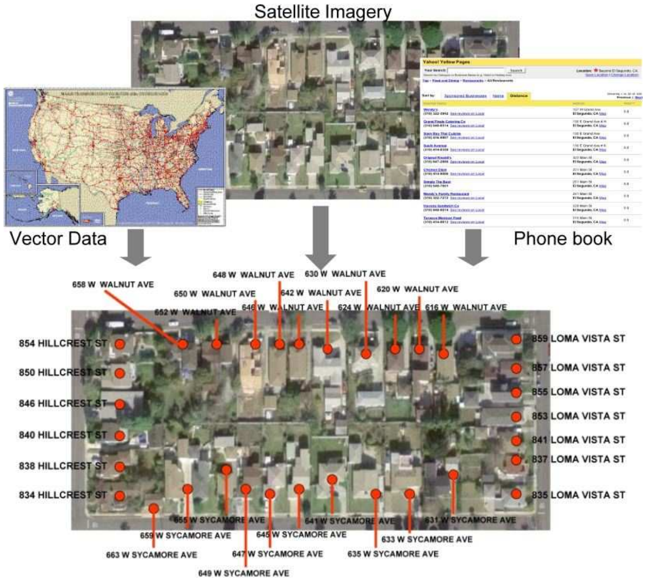

Figure 2: A simple instance of the BID problem

3. CONSTRAINT INFERENCE

As stated in Section 2, one of our contributions is a dynamic approach to model generation that uses instance-specific data to infer applicable constraints. In this section, we define our inference framework and provide examples of all introduced concepts. Section 3.1 details the procedure used by this framework to infer the applicable constraints. Figure 3 informally defines our general inference framework. The generic model has the set C_G of constraints that hold for *all* problems of a problem class. For example, in the BID problem all known addresses have to be assigned to a building, and corner buildings are only assigned to one street.

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,
- 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. The governing constraints, union with C_G, define the constraint model for the given instance.

Figure 3: 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 BID problem, data points are landmark buildings defined by the following set of features: ID, Address Number, Street Name and Orientation, Side of Street, Latitude, Longitude, Block Number, and Street Ordering. The British Prime Minister's residence at 10 Downing Street or the White House at 1600 Penn Avenue are two examples of a data point.

Table 1: Examples of constraints in the constraint library

Name	Description
Parity (odd/even)	Addresses on the same side of a street have the same parity
Continuous	Addresses increment continuously by a fixed number n
Block Numbering (Grid)	Addresses increment by a factor of k across grid lines
Ordering	Addresses increase monotonically along a given street
. . .	

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. Example constraints in the BID constraint library are shown in Table 1. This library serves as the repository from which our framework selects applicable constraints and adds them to the generic constraint model.

Applicability Rules: The rules in our framework are predefined by a domain expert, similar to the use of expert modules in PROVERB [14]. They are separated from the constraints in the library because they act as an 'intermediary' between the constraints and the features defining the data points (variables in the CSP model). 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. 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, i.e. added to the constraint model. The generic constraints for the given problem class are always included in the model. An additional benefit to using rules is that multiple rules can map to a single constraint, allowing for a stronger support for the level of inference of the constraint. Our rule language supports any programmable predicate expressions and rules are defined using the following format:

```
IF ⟨test points' features for rule applicability⟩
    IF ⟨test points' features for constraint applicability⟩
        THEN ⟨add positive support to constraint⟩
        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. A sample rule for the BID problem is shown in Figure 4. This rule checks for the applicability of the Parity constraint, which states that if two points are on the same street and on the same side, then the parity of their address number must be the same.

Parity Rule

```
IF Street($P_1$)=Street($P_2$) then
  IF [SamePar(Num($P_1$),Num($P_2$)) $\wedge$ SameSide($P_1$,$P_2$)] $\vee$
      [OppPar(Num($P_1$),Num($P_2$)) $\wedge$ OppSide($P_1$,$P_2$)]
      THEN Add positive support for Parity constraint
      ELSE Add negative support for Parity constraint
```

Figure 4: Sample BID problem applicability rule

3.1 Inference process

The selection of constraints, based on the information found in the problem (data points), is the key contribution of our framework. Previous work in the Constraint Programming (CP) community on selecting constraints from a library has shown that such an approach is an effective method to modeling CSPs [5, 8]. 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 place the data points into buckets, similarly to the buckets approach of [13]. A single bucket is defined by an attribute value (i.e., Street name A) and represents the first IF condition in the applicability rules. This mechanism allows our framework to efficiently select the two data points to compare, allowing the framework to scale to large sets of data points. Second, after comparing all data points within each bucket, the inference engine has a set of supports, both positive and negative, for all inferred constraints. Constraints with no supports may also exist. Third, 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 add to the generic 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 are constraints in the generic model and $C_a \subseteq C_L$ are constraints classified as *applicable* and selected from the constraint library.

For the purpose of our evaluations, a support level of 1 (the constraint has at least one positive support and no negative support) classifies a constraint as *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. Studying the impact of support levels is our next course of action and is discussed in Section 8. Note that it is possible that a constraint can be incorrectly inferred. This incorrect inference may be caused by noisy data points or by a lack of information in the initial problem definition. For the evaluations carried out in this paper, we assume that all constraints are correctly inferred as long as the inferred model has at least one solution. However, we are currently exploring the use of support levels and constraint propagation techniques as two ways to deal with noisy or uninformative data [16].

4. CONSTRAINT MODEL GENERATION

Once a set of applicable constraints has been inferred, the problem model must be generated. The 'Model Generation' component shown in Figure 1 defines the variables and their domains and uses the inferred constraints to generate constraints over the problem variables. Our model-generation process also provides an additional improvement over our previous approach [15]. As we show in our experimental evaluation, our new model formulation improves the performance of the constraint solver. Our new model has three types of variables: *orientation* variables (called global variables by Michalowski and Knoblock [15]), *building* variables, and *corner* variables. The *orientation* variables describe the overall layout of the map, such as the direction in which the numbers appear. The *building* variables store the numeric address assignments to a specific building and the *corner* variables store the street assignments to corner buildings.

There are four orientation variables, exactly like the ones described by Michalowski and Knoblock [15]. Normally, the model has exactly one of each of the four orientation variables. This fact reflects that street numbering schemas are homogeneous over the considered geographical area. However, there are real-world situations where orientations differ between streets, as is the case for the city of Belgrade. Our model can easily accommodate such non-homogeneous numbering schemas by generating additional orientation variables for those streets that do not follow the regular pattern.

The model includes a building variable for every building on the map. The domain of each such variable is the set of all addresses (a combination of a street and a number) that the building can take. To populate the domain for a building on a given street, Michalowski and Knoblock [15] enumerated all addresses from 1 to the largest number that appears in the phone book for that street. However, choosing the largest phone-book address is an arbitrary limit, and leads to incorrect results when the correct solution contains addresses larger than the largest phone-book address. To address this issue, in our model the range of possible addresses is specified as $(0, \infty)$, unless the largest value is known. Note that the domains contain values that are not in the phone book, and thus the solutions that we find contain values for the missing addresses. Domains are represented as a set of intervals, and in Bayer et al. [4], we discuss in detail this representation and introduce reformulation techniques to reduce the size of these intervals.

We include a corner variable in the constraint model for every corner building on the map. The domain of a corner variable is the set of streets to which the building is adjacent. We use separate variables for the assignment of an address and a street, where an address is a combination of a street and a number, for the following reason. Determining the streets on which corner buildings lie before we assign numbers to them decomposes the constraint network into chains, as discussed by Bayer et al. [4]. Note that the corner variables are *hidden* variables, in the sense that they are not part of the solution reported to the user. They are variables that only exist to facilitate the decomposition of the problem. Finally, we reduced the arity of the generic constraints defined by Michalowski and Knoblock [15] but due to a lack of space, we omit the details from this paper.

In this paper, we claim that a specialized model is likely to produce more precise solutions than a generic model. This statement is supported by the following points: (1) A generic constraint model represents the *least* constrained model for a BID problem instance, and (2) All constraints in the constraint library are monotonic (the addition of a new constraint cannot increase the set of consistent solutions). Therefore, any constraint model not equivalent to the generic model is guaranteed to generate a solution space whose size is less than or equal to the size of the solution space of the generic model. It is possible that the specialized model, for lack of sufficient information in the data, is still not able to narrow down the set of consistent solutions to a single one, but returns a subset of the solutions of the generic model. This subset still represents a reduction in the solution space and, conversely, an increase in the precision of the returned solutions. Figure 5, where P_G is the problem with generic constraints and P_{new} is the problem with inferred constraints, illustrates the reduction of the solution space by the addition of C_a. The results presented in Section 6 show the increased precision.

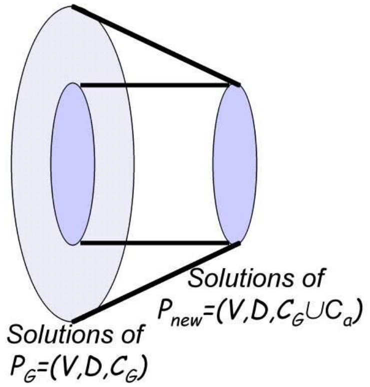

Figure 5: Solution space reduction by the addition of inferred constraints

5. SOLVING A BID PROBLEM

The intuition behind solving a BID problem instance is as follows. When presented with a user query, the satellite imagery provides us with the buildings that need to be assigned addresses. Because we do not know which streets each building could lie on, we use the vector data to determine this information. Finally, the phone-book entries provide us with a set of addresses that are known for the area and must appear in the final solution. We combine all of the gathered and induced information with the addressing characteristics for the area to assign address labels to the buildings in the image. Section 3 showed how we determine which addressing constraints characterize the given area. Section 4 detailed how the gathered and induced information is combined with the inferred constraints to generate a customized model of the problem instance. Below, we give a brief overview of our specialized solver used to solve the customized constraint model.

Our specialized solver is implemented as a backtrack-search solver in Java that uses backtrack search (BT) with nFC3, a look-ahead strategy for non-binary CSPs [6], and conflict-directed back-jumping [18]. It exploits the topology of the problem to improve the performance of search in the following way. After we assign streets to the corner buildings, any ordering constraint between a corner building and a building on a different street is deactivated. Thus, once the orientation and corner variables have been instantiated, the constraint network becomes a forest.

Freuder [10] showed that tree-structured constraint graphs can be solved in polynomial time. We first apply (generalized) arc-consistency directionally from the leaves of the tree to the root, chosen arbitrarily. Then, when no domain is wiped out by directional arc-consistency, we can instantiate the nodes of the tree in a backtrack-free manner (i.e., in linear time in the number of the nodes). Consequently, in our solver, as long as nFC3 does not cause a domain wipe-out, we can stop search and guarantee solvability without instantiating the remaining variables. In this sense, the corner variables are backdoor variables of the CSP[3].

By leveraging the structure of the BID problem and incorporating reformulation and abstraction techniques, we are able to support larger problem instances than were previously possible. Furthermore, our specialized solver allows constraints to be turned on/off as needed, providing support for varying problem models. For more implementation and algorithmic details, we refer the reader to our work presented in Bayer et al. [4].

6. EXPERIMENTAL EVALUATION

In this section, we present experimental results obtained when applying our framework to El Segundo California (see Figure 6). The applicable constraints that characterize the addressing found in this area are the following: (1) Parity constraints restrict addresses to the same parity for houses on the same side of a given street and to the opposite parity for houses on opposing sides of the street, (2) Orientation constraints specify the direction in which address numbers increase along a street, and (3) 100-Block Numbering constraints indicate that addresses go up by increments of 100 across blocks. To generate a specialized model for this area, we applied our constraint-inference framework using various data-point sources. We were able to correctly identify all of the applicable constraints using 26 data points in this area. Therefore, we use the specialized model generated using these data points throughout all of the experiments. For a thorough evaluation of the effects that data points have on inferring constraints for different areas in the US, see our work presented in Michalowski et al. [16].

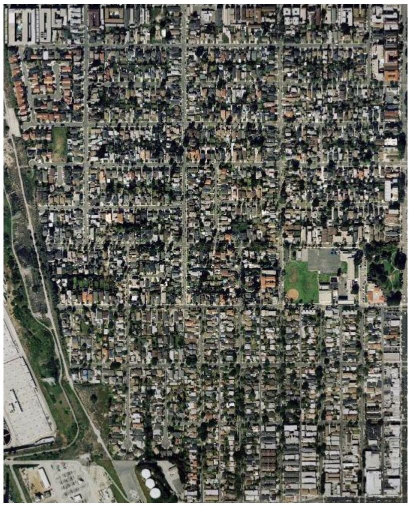

Figure 6: El Segundo area used in experiments

Having inferred the constraint model for El Segundo, we validate our claims of improved solution quality and solving efficiency by applying the model to several different regions in the city of El Segundo. These cases vary in size and which regions of the city they cover. Table 2 describes the properties of the regions on which we ran our experiments. The largest region tested by Michalowski and Knoblock [15] contained 34 buildings and a single city block. 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 than when it is complete.

Table 2: Case studies used in experiments

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

Our case study results are summarized in Tables 3 and 4, and divided into two categories: (1) The problem model *without* the block-numbering constraint, and (2) The problem model *with* the block-numbering constraint. As the work by Bayer et al. [4] showed, without the block-numbering constraint (called the grid constraint by Bayer et al. [4]) the BID problem can be solved in polynomial time using a matching algorithm. The existence of the block-numbering constraint forces the BID problem solver to use the backtrack search CSP algorithm. Therefore, to carry out a fair comparison with previous results, we demonstrate the improvement introduced by our inference framework for both the matching and search algorithms. To further demonstrate the effect inferred constraints have on the solving process, for each algorithm we present results obtained when the Parity and Ordering constraints (denoted by *orientation cons* in Tables 3 and 4) are included and when they are unknown. *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 for the matching algorithm presented in Table 3 show on average a factor of 17 improvement in runtime and a factor of two improvement in domain reduction. As Bayer et al. [4] noted, every building has the correct label in its domain (resulting in a perfect recall). Therefore a factor of two domain reduction results in a large increase in the solution's precision. For the CSP search algorithm results presented in Table 4, the domain reduction is less than that of the matching algorithm results, although there is still a reduction of an average factor of 1.68. This is due to the

[3] Backdoor variables are those variables whose instantiation reduces the CSP into a tractable problem [12].

Table 3: BID problem results for case studies without the block-numbering constraints

Matching-Based Solver

	W/o orientation cons		W/ orientation cons			
	Runtime (sec)	Domain size	Runtime (sec)	Domain size	Runtime reduction	Domain reduction
NSeg125-c	90.87	1.95	5.13	1.0	17.71x	1.95x
NSeg125-i	41.25	6.84	2.42	4.68	17.05x	1.46x
NSeg206-c	393.04	2.70	22.28	1.39	17.64x	1.94x
NSeg206-i	192.98	8.75	11.08	5.83	17.42x	1.50x
SSeg131-c	152.29	3.52	9.78	1.90	15.57x	1.85x
SSeg131-i	46.62	13.05	3.04	4.06	15.33x	3.21x
SSeg178-c	379.96	3.59	19.25	1.93	19.74x	1.86x
SSeg178-i	79.24	8.68	5.05	3.41	15.69x	2.55x
				Average	**17.02x**	**2.04x**

Table 4: BID problem results for case studies with the block-numbering constraints

CSP Search Solver

	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**

fact that the search algorithm includes the block-numbering constraint which further constrains the problem and produces small final domain sizes. However, we can still see a significant improvement in the runtime when the inferred constraints are included in the model. On average, we see a factor of 8.31 improvement in runtime, with some scenarios seeing a reduction by a factor as large as 18.37.

7. RELATED WORK

We divide the work related to the topics described in this paper into two main bodies of research: modeling for Constraint Programming (CP) and geospatial data integration. Recent work in CP modeling aims at automatically learning constraint networks from data. Coletta et al. [8] automatically learn constraint networks from full solutions (both consistent and inconsistent). Bessière et al. [7] use historical data (solutions previously seen) to learn constraint networks. Finally, Bessière et al. [5] propose a SAT-based version-space algorithm for acquiring constraint networks from examples. All of these approaches are a way to model a problem class without having to explicitly state the constraints. 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 set of data points). Furthermore, our work identifies small variations of similar problem classes while previous work focuses on finding constraint networks for a particular problem class. As future work, we propose to extend the techniques in this branch of research to support the types of constraints encountered in the BID problem (see Section 8), allowing us to automatically build-up the constraint library.

Specifically related to the BID problem, the work described by Bakshi et al. [3] presents methods to accurately geocode addresses using publicly available data sources. The end result of this work is accurate latitude and longitude coordinates for buildings in a given area. This work also uses online sources to improve the accuracy of building locations. The goal of this work is similar to ours in that their work attempts to identify buildings in satellite imagery. However, the authors are provided with an address and attempt to find the accurate position of the building in the image while we attempt to identify buildings in imagery that could be assigned the given address. Furthermore, this work assumes that the sources used to locate buildings in an image are complete, meaning they contain all of the buildings for a given area. This assumption does not hold for areas of the world where geospatial data is not readily available or where the requisite data is found across multiple sources.

There has also been work done on identifying buildings in satellite imagery and merging geospatial databases using computer vision approaches [1, 2, 9]. While some of the goals in this work are similar to ours (identifying objects in images), the work is primarily focused on the actual detection of buildings in the images. Thus, the goal of this work varies from our goal of labeling and reasoning over specific buildings in images. This work could serve as a source of information for the inference framework we developed.

8. DISCUSSION AND FUTURE WORK

In this paper, we describe improvements to the Building Identification (BID) problem-solving process. We present a framework for inferring constraint models specific to a given problem instance and a specialized solver that leverages the inherent structure of BID problems and exploits characteristics of the inferred model. We empirically show how the customized problem model improves the efficiency and precision of the solving process using real-world data, and we focus our evaluations on real-world BID problem instances. As shown by Michalowski and Knoblock [15], Constraint Programming is an effective paradigm for solving geospatial integration problems. With the techniques presented in this paper, the ability to cast geospatial integration problems as Constraint Satisfaction Problems (CSPs) is enhanced.

Furthermore, even though our constraint-inference framework is only applied to the BID problem, the techniques presented are general and can be applied across domains (see [16]). One such domain is syntactic machine translation [11, 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 which constraints apply. This information would allow a translation engine to be optimized at run-time based on the the deduced rule set and would open up the possibility of a more generalized translation engine for performing multiple bilingual translations.

Our future work aims to further improve the precision of the inferred model. As mentioned in Section 3.1, we may encounter situations where constraints are incorrectly classified as applicable. To handle these situations, we will study *support levels* and how they can be used to avoid incorrect inferences. Eliminating erroneous inferences will further improve the quality of the returned solutions. Similarly, we are working towards reducing the role played by a domain expert. In our current framework, this expert must define the constraints that make up the constraint library. We would like to eliminate this requirement by extending existing techniques in learning constraints and using them to build the constraint library automatically. Finally, we will apply our approach to areas other than El Segundo to evaluate its effectiveness when presented with different sets of addressing characteristics.

9. ACKNOWLEDGMENTS

This research is based upon work supported in part by the National Science Foundation under CAREER Award No. IIS-0324955, and in part by the Air Force Office of Scientific Research under grant numbers FA9550-04-1-0105 and FA9550-07-1-0416. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

The authors would also like to thank Maria Muslea for her contribution to the experimental evaluation.

10. REFERENCES

[1] P. Agouris, K. Beard, G. Mountrakis, and A. Stefanidis. Capturing and modeling geographic object change: A spatio-temporal gazeteer framework. *Photogrammetric Engineering and Remote Sensing*, 66(10):1224–1250, 2000.

[2] P. Agouris and A. Stefanidis. Integration of photogrammetric and geographic databases. *International Archives of Photogrammetry and Remote Sensing, ISPRS XVIIIth Congress*, 31:24–29, 1996.

[3] R. Bakshi, C. A. Knoblock, and S. Thakkar. Exploiting online sources to accurately geocode addresses. In *Proceedings of the 12th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'04)*, 2004.

[4] K. M. Bayer, M. Michalowski, B. Y. Choueiry, and C. A. Knoblock. Reformulating csps for scalability with application to geospatial reasoning. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-07).*

[5] C. Bessière, R. Coletta, F. Koriche, and B. O'Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. In *Proceedings of ECML'05*, pages 23–34., Porto, Portugal, 2005.

[6] C. Bessière, P. Meseguer, E. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP-99)*, pages 88–102, 1999.

[7] C. Bessière, J. Quinqueton, and G. Raymond. Mining historical data to build constraint viewpoints. In *Proceedings of CP-06 Workshop on Modelling and Reformulation*, pages 1–16, 2006.

[8] R. Coletta, C. Bessière, B. O'Sullivan, E. Freuder, S. O'Connell, and J. Quinqueton. Semi-automatic modeling by constraint acquisition. In *Proceedings of CP-03*, pages 111–124, 2003.

[9] P. Doucette, P. Agouris, M. Musavi, and A. Stefanidis. Automated extraction of linear features from aerial imagery using kohonen learning and gis data. *Lecture Notes in Computer Science*, 1737:20–33, 1999.

[10] E. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[11] M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of COLING/ACL2006*, pages 961–968, 2006.

[12] P. Kilby, J. Slaney, S. Thiébaux, and T. Walsh. Backbones and backdoors in satisfiability. In *Proceedings of AAAI 2005*, pages 1368–1373, 2005.

[13] A. Levy. Logic-based techniques in data integration. In *Logic Based Artificial Intelligence*, pages 575–595. Kluwer Publishers, 2000.

[14] M. L. Littman, G. A. Keim, and N. Shazeer. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1-2):23–55, 2002.

[15] M. Michalowski and C. A. Knoblock. A constraint satisfaction approach to geospatial reasoning. In *Proceedings of AAAI-05*, pages 423–429, 2005.

[16] M. Michalowski, C. A. Knoblock, and B. Y. Choueiry. Exploiting problem data to enrich models of constraint problems. In *Proceedings of the Sixth International Workshop on Constraint Modelling and Reformulation (ModRef'07)*, 2007.

[17] F. J. Och and H. Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449, 2004.

[18] P. Prosser. Mac-cbj: Maintaining arc consistency with conflict-directed backjumping. Technical Report 95/177, Univ. of Strathclyde, 1995.

[19] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *The Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590, 1999.