Identifying Buildings in Aerial Images Using Constraint Relaxation and Variable Elimination

Thomas H. Kolbe, Lutz Plümer, and Armin B. Cremers, University of Bonn

THE AUTHORS SHOW HOW CONSTRAINT LOGIC PROGRAMMING CAN HELP DETECT BUILDINGS FROM THE AIR. THE RECOGNITION SYSTEM DESCRIBED IN THIS ARTICLE USES CONSTRAINT RELAXATION AND VARIABLE ELIMINATION TO HANDLE UNCERTAINTY AND UNOBSERVABILITY OF BUILDING PARTS.

THIS ARTICLE DESCRIBES AN application of constraint logic programming that comes from the research field of object recognition: the identification of buildings in aerial images. 3D building extraction is needed for an increasing number of tasks related to measurement, planning, construction, environment, transportation, energy, and property management. Semiautomatic photogrammetric tools are well established,[1] but they reveal inefficiencies due to the extensive amount of data that must be acquired. The integration of automatic, or at least semiautomatic, image-understanding tools to photogrammetry provides a way to achieve efficiency in 3D data acquisition.

Researchers in photogrammetry and remote sensing have studied this topic extensively, because of its high practical relevance. In fact, a German National Research Council program called "Semantic Modeling" has been running since 1993, bringing together photogrammeters, cartographers, and computer scientists. Its main goal is to incorporate *semantics* and *explicit models* into the object-recognition process. Here, we show how we incorporate explicit models in the subproject "Building Extraction" by using constraint techniques and logic programming embedded in the Eclipse System.[2–4]

Modeling buildings

We employ a generic, hierarchical building model with a coherent 2D and 3D representation.[5] Figure 1 shows the four aggregation levels: buildings, building parts, feature aggregates, and features. A volumetric 3D building model contains building parts subdivided into terminals and connectors. These terminals and connectors build some form of construction kit, where only certain building parts fit to others. This allows generic modeling of complex buildings such as L-shaped, S-shaped, or U-shaped houses. We can even model rows of buildings this way. Building parts themselves contain special groups of features called *feature aggregates*. Features build the lowest level of the hierarchy and are subdivided into three classes: point, line, and region.

The coherent 2D and 3D modeling allows different interpretation strategies. For example, the transition from 2D image space to the 3D object space can occur at different levels. One or more 2D raster images give the interpretation its starting point. A digital elevation model (DEM), if available, provides additional 3D information about the height of observable objects in the given scene's context. The final result is a 3D instance of the building model.

The L-shaped saddleback-roof house in Figure 1 is an example of a special building

1094-7167/00/$10.00 © 2000 IEEE

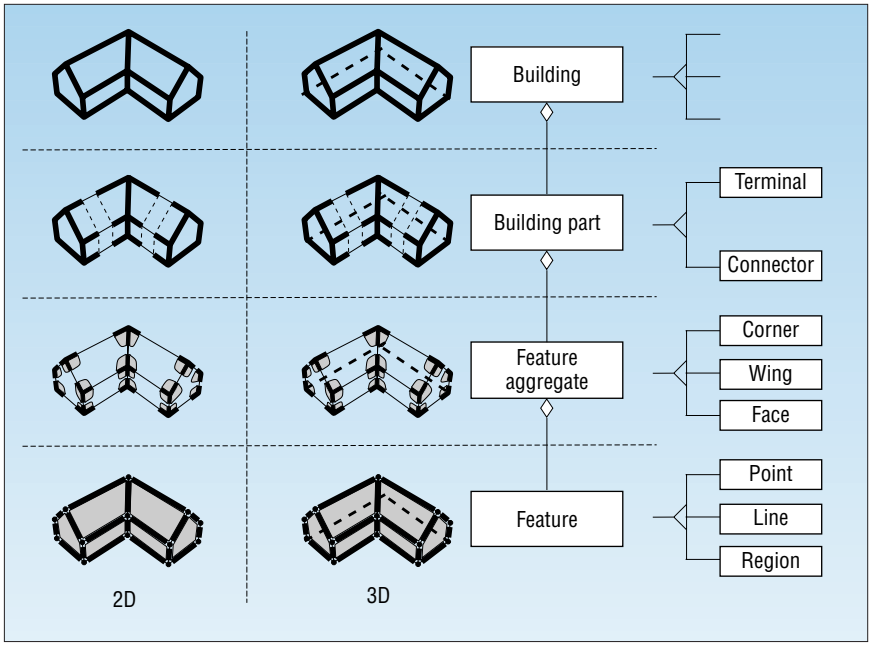

Figure 1. Hierarchical, coherent 2D and 3D model for building reconstruction.

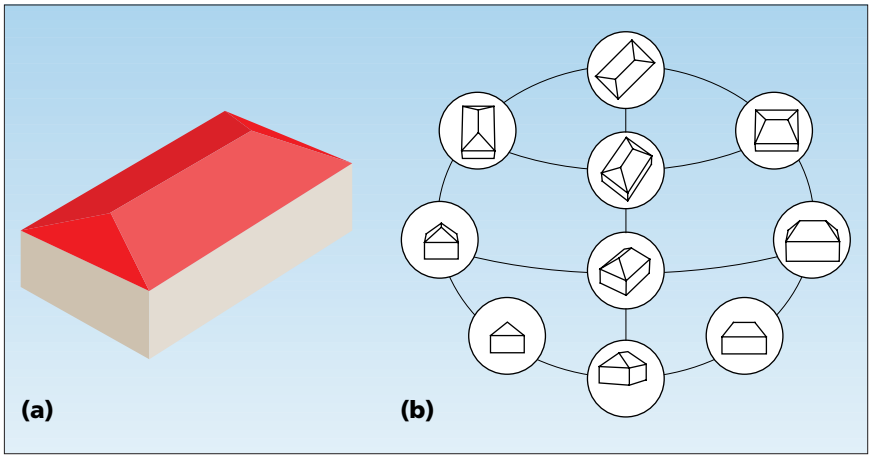

Figure 2. (a) 3D model of a hip-roof house; (b) the corresponding aspect graph. Every node of the aspect graph represents a class of topologically invariant object views.

hypothesis. To identify houses in aerial images, we need such a hypothesis. In this article, we focus on identifying buildings for which the hypotheses are already given.[6]

The formal methods applied to describe our models are constructive solid-geometry models augmented by constraints, aspect graphs, and structures for image features and their interrelationships. On the 3D object-level volumes, CSG models represent buildings' geometry and physical properties. (CSG is a formalism well established for man-made artifacts in computer-aided design.)

Although CSG models use Boolean operators to construct complex objects from primitives, most building semantics must be represented by additional constraints. For example, CSG models might say that a simple hip-roof house has a block and a prism, and additional constraints might state that the prism is fitted to the upper part of the block so that it takes into account the roof gutters. Other constraints restrict form parameters such as building height, width, and depth to reasonable sizes and ratios.

On the 2D image level, relations such as parallelism, collinearity, and adjacency associate observable features such as points, lines, and regions. These observations, extracted by low-level, syntactic operators build the feature relation graph. This graph's representation is mainly intentional, because only certain relations such as feature adjacency are explicitly enumerated by the feature extraction process.

A third model—the aspect graph—relates objects represented by constrained CSG models to observations aggregated in FRGs.[7] One aspect graph is derived for each polyhedral CSG house model, specifying which part of that model is observable from a certain viewpoint. An aspect graph's nodes represent the topologically different views (aspects) of 3D objects, and the edges specify view changes. Figure 2 shows a building's CSG model and its aspect graph. Once again, every aspect is composed of points, lines, and regions.

Image model and data. Aerial images have a typical size of about $10{,}000 \times 10{,}000$ pixels. Grey-level images have one channel per pixel; color or radar images have more. To interpret the image contents, we segment and aggregate it into three feature classes: points, lines, and homogeneous regions (blobs).[8] Figure 3 shows part of an aerial image and the result of its segmentation. The extracted features form a level of symbolic description, which is transformed to Prolog terms. Features are stored as Prolog facts, such as

```
line(id,y1,x1,y2,x2,angle,
  length,scale,type)
point(id,y,x,scale,strength,
  type)
blob(id,bbox_y1,bbox_x1,bbox_
  y2,bbox_x2,stripes,size,
  circum,formfactor,numholes,
  interiorpoint_y,interiorpoint
  _x,diameter,gravitypoint_y,
  gra svitypoint_x,mainangle,
  intensity,intensityvariance)
```

Another result of the segmentation is the feature-adjacency relation, enumerating the neighbors of each feature. An extensional subgraph of the FRG, the feature-adjacency graph is represented by Prolog facts of the form

```
fag_edge(edge_id,feature_id1,
  type1,feature_id2,type2)
```

To get an idea of the combinatorial complexity of the search task, note that even the segmentation of small portions of an aerial image results in several thousand points, lines, and regions. The number of edges of the corresponding feature-adjacency graph is one order of magnitude higher (see Figure 3).

Object models and their transformations. From constrained CSG models, we derive the corresponding aspect graphs represented by Prolog facts similar to the facts representing

the image features. Implicit in this transformation is a mapping of 3D object models to 2D representations. Our system transforms each aspect to a CLP clause such as the one in Figure 4.

As the figure suggests, the number of variables corresponds to the number of features associated with one aspect, and the number of constraints to the number of relations specific for that aspect. The domain variables F_1, …, F_3 represent expected observable regions; L_1, …, L_{10} represent expected observable lines. Their domains are the corresponding class's features extracted from the whole image. Therefore, the number of extracted lines (respectively, regions) determines the domain sizes. In the listed CLP clause house_aspect, the first two lines construct and assign the domains to the variables. The next two lines impose *all-distinct* constraints on each class of variables, ensuring that all variables are assigned different image features. The following `fag_arc` constraints restrict the given variable pairs to have only adjacent features for values. Finally, the `line_parallel` constraints state that lines must be parallel to constitute valid values for the given variable pairs. To keep this example simple, we omit the variables representing point features and the other, mostly geometrical constraints. Most of the constraints refer to intentional relations of the FRG (parallelism). Hence, instead of a simple lookup, explicit arithmetic computations are needed, and this degrades performance considerably. Appropriate representation of geometry, however, reduces necessary computations. If, for example, lines are specified by polar coordinates, the test for parallelism is immediate. Other representations might require time-consuming calculations.

We have found that the following different types of constraints are essential for our application context:

- Adjacency of features: `fag_arc`(F_1, F_2) is true if the features F_1 and F_2 are neighbored and, thus, (F_1, F_2) is an edge of the (extensional) feature-adjacency graph.
- (Approximate) parallelism of lines: `line_parallel`(L_1, L_2) is true if the enclosed angle between the lines going through the two line segments L_1 and L_2 is smaller than a given threshold.
- (Approximate) collinearity of a line and a point: collinear(P, L) is true if point P is closer than a given threshold to the line going through line segment L.

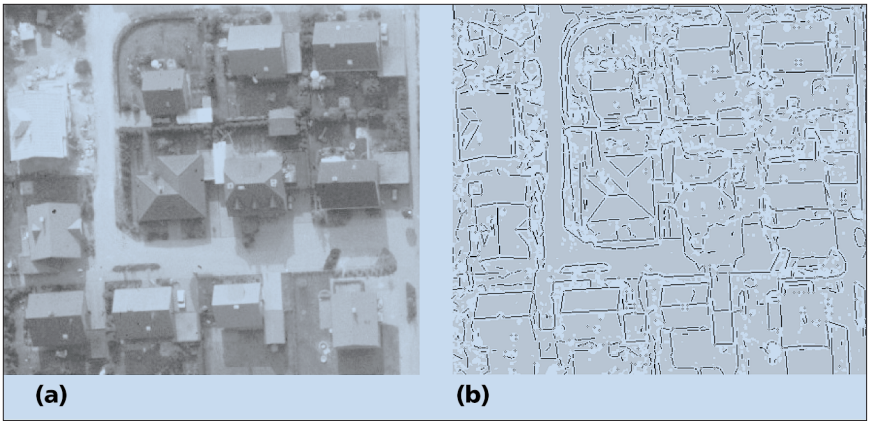

Figure 3. (a) Portion of a greyscale aerial raster image; (b) its segmentation into points, lines, and homogeneous regions. This small image already contains about 3,300 points, 3,100 lines, and 2,100 regions. The corresponding feature-adjacency graph contains 24,000 edges.

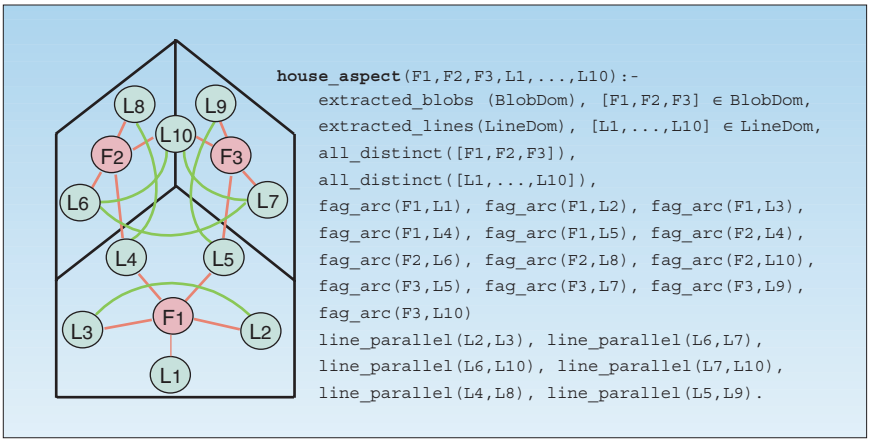

Figure 4. Constraint representation of an aspect of a saddleback-roof house containing lines L_1, …, L_{10} and regions F_1, …, F_3. The extracted image features build the variable domains. The `fag_arc` constraints express the adjacencies between lines and regions.

- Two line segments lie in the same half plane: `same_side_lines`(L_{ref}, L_1, L_2) is true if the two line segments L_1 and L_2 lie on the same side of the line going through the third line segment L_{ref}.

Whereas `fag_arc` is a topological constraint, the other three types are geometrical constraints. We empirically determine the needed thresholds for the `line_parallel` and collinear constraints by statistical analysis of training matches. For simple detached buildings such as bungalows, saddleback-roof houses, or hip-roof houses, each aspect is typically specified by about 25 variables and more than 100 constraints.

Applying constraint logic programming

Applications of constraint logic programming in domains with spatial attributes, although promising, are rare. Nevertheless, CLP is a language context that can represent and link the different kinds of models used in the interpretation process. The application context just described suggests using the CLP(FD) (that is, CLP over finite domains) paradigm to solve the difficult combinatorial problems arising in image interpretation. In fact, David Waltz's seminal paper on the interpretation of line drawings, which initiated the AI-related research on constraint propagation and from which the technique of forward checking that underlies CLP(FD) emerged, was entitled "Understanding Line Drawings of Scenes with Shadows."[9]

Finite domains of structured terms. As we described, the aspect model implies a domain variable for each expected feature in the image. To instantiate this template and identify the corresponding object, the task is to find a matching set of extracted image features. Solving the given constraint-satisfac-

Glossary

CAD	Computer-aided design
CSG	Constructive solid geometry
CSP	Constraint-satisfaction problem
CLP	Constraint logic programming
CLP(FD)	Constraint logic programming over finite domains
DEM	Digital elevation model
DFG	Deutsche Forschungsgemeinschaft
FRG	Feature-relation graph
Tcl	Tool command language (available from Sun Microsystems/Scriptics Corporation)
TK	Toolkit (the graphical user interface extension to Tcl)

tion problem accomplishes this. The CSP either fails (if there is no possible solution) or succeeds with a complete and consistent variable binding. The extracted image features are given as facts of a database as the result of the previous image-segmentation process. The object domain's finiteness suggests the application of a CLP(FD) constraint solver. We decided to use the Prolog/CLP system Eclipse because of its good scalability, extensibility, multiple argument indexing, and efficient implementation of domain reduction, propagation, and suspension handling.[4]

The three different image feature classes (point, line, and region) are objects of complex structure. They are represented by Prolog terms. This representation causes a problem, because in most implementations of CLP(FD), variable domains might contain only integer numbers or atoms. This makes it impossible to represent a complex object as one value of a domain variable. To overcome this problem, we define the variable domains D_{blob}, D_{line}, and D_{point} as the sets of the (atomic) object identifiers of all extracted objects of the corresponding classes. The general feature domain $D_{feature}$ is then defined by $D_{feature} := D_{blob} \cup D_{line} \cup D_{point}$.

Adapting and extending the FD constraint solver. Given the problem just described, constraints relate names of the respective terms specifying aggregated objects. Thus, the syntactical format is satisfied. The semantics of our constraints just illustrated refer to specific object attributes. Parallelism, for instance, refers to the coordinates of points of line features.

Therefore, we cannot directly apply the given standard (and often built-in) constraints of most CLP(FD) systems (that is, $<$, $\leq$, $=$, $\neq$, $\geq$, $>$, at most, and element). We have implemented an extension to the built-in solver that can reflect this indirection and handle the constraints specified in the "Object models and their transformations" section. This extension lets the constraint solver handle arbitrary test predicates. As with the CLP system CHIP's lookahead and forward declarations, the user augments constraints by specifying the inference rule to be used for solving them. So, the approximate parallelism constraint is implemented in the following way:

```
line_parallel_test(L1,L2) :-
  line(L1,_,_,_,_,Alpha,_,_,_),
  line(L2,_,_,_,_,Beta,_,_,_),
  Diff is abs(Alpha-Beta),
  Diff < MAXANGLE_THRESHOLD.
line_parallel(L1,L2) :-
  forwardcheck(L1:L2:line_
    parallel_test(L1,L2)).
```

As the listing shows, the indirection is handled by the test predicate, which first fetches the needed object-attribute values (here, the line angles) from the database and then checks for the condition. The second predicate specifies that the forward-checking handler must handle `line_parallel`(L_1, L_2), and that L_1 and L_2 are this constraint's domain variables. The `line_parallel` constraint is forward-checkable if both arguments are ground or if one argument is ground and the other is a domain variable.

Although there are many constraints, in most cases a single constraint is not very specific. On the other hand, the variable domains are of considerable size. Thus, forward checking alone is too weak and requires explicit generation of values for variables in most cases. The feature-adjacency graph's special characteristics—mainly, the small degree of its nodes—make lookahead feasible. It plays a crucial role in our context, if implemented efficiently, as the benchmarks given in the last section illustrate. Lookahead has an efficiency problem: the respective constraints are evaluated repetitively, and computation costs for each single evaluation might be rather high. A special caching mechanism lets us replace these time-consuming computations with simple lookups. The cache is a set of Prolog facts of the form

```
cache(constraint_name,
old_domain1, old_domain2,
new_domain1, new_domain2,
changed1, changed2)
```

where `constraint_name` denotes the type of binary constraint (for example, `line_parallel`), and the two flags `changed1` and `changed2` indicate reductions in the respective domains. In a certain sense, (`old_domain1`, `old_domain2`) specify a context where a domain reduction for the given constraint type might be reused.

Handling unobservable objects and constraint violation. To find a matching, the constraint solver must assign every variable an extracted image feature and satisfy all constraints. Unfortunately, because of occlusions, noise, low contrast, and segmentation errors, building parts are often unobservable in the image, and some constraints might be violated. In such cases, the matching between the extracted image features and the corresponding building model using standard constraint techniques would fail.

The simple relaxed approach. One approach would be to relax violated constraints to find a solution. In fact, several established schemes, such as Maximum Constraint Satisfaction and Molly Wilson and Alan Borning's Hierarchical Constraint Logic Programming (HCLP) are based on this idea. Every constraint violation causes costs, and the best solution comes from minimizing the total cost of all constraints. However, for the identification of buildings in aerial images, this approach has severe deficiencies.

First, it is not clear where the constraint weights come from or how they can be derived. How can we combine the different constraint weights to build a global evaluation function? Also, the semantic of the evaluation function with respect to the recognized building is unclear.

Second, this approach does not reflect the difference between unobservability of building parts and the violation of constraints. Although in both cases constraints cannot be satisfied, we must treat them differently. Because in certain cases the unobservability of objects for low contrast is predictable, the unobservability of an object on the one hand might not simply be rated by the sum of the

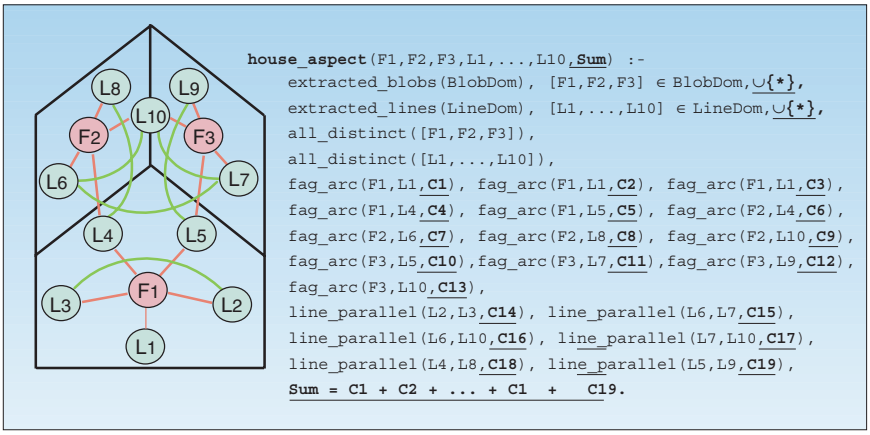

Figure 5. Extended constraint representation of an aspect of a saddleback-roof house. With respect to Figure 4, variable domains are extended by the wildcard value *. Furthermore, constraints are extended by the indicator variables $C_1, \ldots, C_{19}$, reflecting both the satisfiability status and the weight of each constraint. Finally, the clause header is extended by an argument containing the global evaluation function, which is defined as the sum of all indicator variables. The best matching comes from maximizing this function using built-in predicates of the Eclipse system.

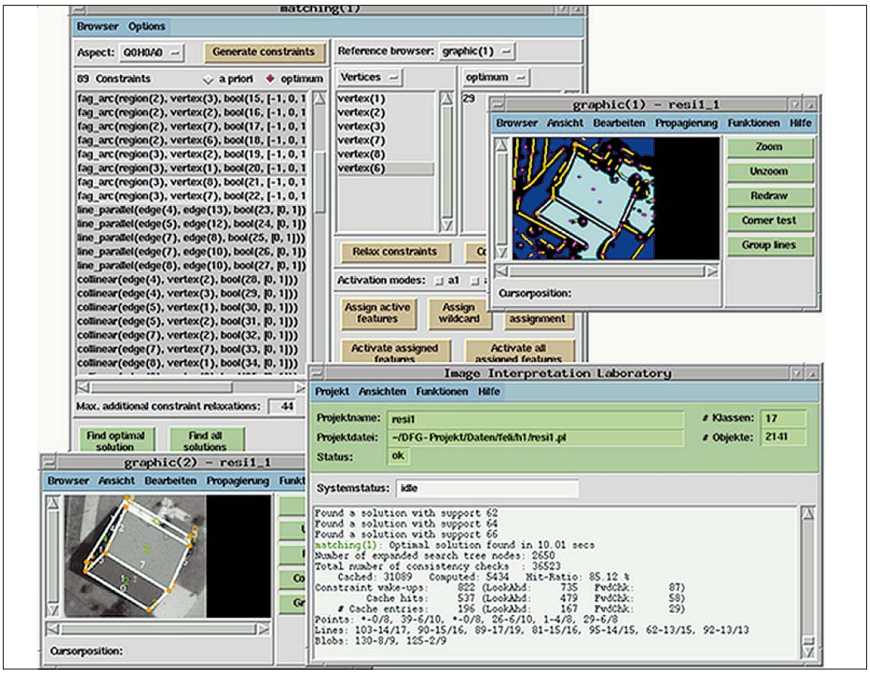

Figure 6. Screenshot of a working session with the Image Interpretation Laboratory. The big window lists the constraints and variables representing an aspect of a saddleback-roof house. The deselected constraints were relaxed because of unobservability or constraint violation by the matching process. The upper right window highlights the assigned image features of the best matching found for the given building model. The window at the bottom left shows the reconstructed building.

relaxation costs of all incident constraints. On the other hand, the costs for dropping the respective variable and the relaxation of its incident constraints must be greater than zero; otherwise, the best solution (for minimal costs) would be to drop all variables and constraints.

For these reasons, we take a different approach.

Our approach. To reflect the difference between unobservability and constraint violation, we extend every variable domain by the wildcard symbol * and every constraint $c(v_1, \ldots, v_n)$ by a three-valued domain variable $b \in \{-1, 0, 1\}$ to $c'(v_1, \ldots, v_n, b)$. The handler for the extended constraint c' works as a wrapper for the original constraint c. It prevents failure of the whole CSP if c is violated or cannot be satisfied, because of unobservability of an incident object, by setting b to the appropriate value. The b variable indicates the status of the original constraint c, where –1 means violated, 0 relaxed, and 1 satisfied.

The wildcard is a special value standing for a feature that was expected in the model but not observed in the image. If a variable gets assigned the wildcard value, the solver relaxes all incident original constraints c by setting the status variables of the corresponding extended constraints c' to 0. Thus, assigning the wildcard value to a variable effectively eliminates it from the given CSP.

To give a well-defined meaning to the best matching, we use George Vosselman's evaluation function for the rating of relational matchings.[10] This method applies concepts from information theory to compute constraint weights from probabilities about the satisfiability of the different constraints. These probabilities are determined empirically from perfect training matches. Every constraint gets two weights: one for violation (mostly having a negative value), and one for satisfaction (usually having a positive value). The weights replace the –1 and 1 values and express that a constraint's status either contradicts or supports a match. If a constraint becomes relaxed because of an incident object's unobservability, the constraint is rated 0, because it neither supports nor contradicts the mapping. Using information theory, combining constraint weights involves simply summing all weights to construct the global evaluation function. Finally, the best matching is found by maximizing this function. Based on Vosselman's results, we know that maximizing such a function yields the most likely match in a strict probabilistic sense.[11] Figure 5 shows the extended constraint representation of the house aspect with respect to Figure 4.

Controlling the search: heuristics

Identifying objects has an immanent combinatorial complexity. In the previous discussion, the number of involved variables, their domain sizes, and the number of constraints are all very high. This is due to the magnitude of extracted image data and the large number of features required to specify buildings, from aspects to aspect graphs to various CSG models. Obviously, the identification of strong heuristics supporting the inherent capabilities of CLP(FD) is of utmost importance.

We apply heuristics in the two (successive) reasoning phases of constraint placement and variable instantiation. In both phases, the heuristics determine the order

in which the constraints and the variables are processed.

CLP(FD) has an incremental solver, where the actual constraint store is only updated and not completely rebuilt when new constraints are added. Therefore, carefully selecting the next constraint to add to the constraint store can help minimize domains after each placement step, thus reducing the overall computation time. For example, let's say there are three domain variables A, B, and C with domains $D_A\{1 \ldots 2000\}$, $D_B\{1 \ldots 2000\}$ and $D_C\{1 \ldots 30\}$ and two lookahead constraints cons(A, B) and cons(B, C). Processing the second constraint first would be better because D_B would very likely be reduced significantly. If D_B is significantly reduced before processing the first constraint, this constraint's computational costs will be a lot less. Although this special heuristic sounds rather obvious, Eclipse unfortunately does not supply it as a built-in feature.

When all constraints are in the constraint store, and no further domain reduction (due to constraint propagation) is possible, the system must explicitly instantiate variables to reach a solution. Because each instantiation initiates propagation and, therefore, determines the domain reduction of all directly or transitively related variables, the instantiation order is crucial for the effective complexity. Here, two different types of heuristics are considered:

- *Application-dependent heuristics* consider qualitative information imposed by the application models. For example, both the model and our observations show that, after feature extraction, the lines found are more likely to be fragmented than are the regions. So, it is advantageous to start variable instantiation with region variables and instantiate line variables afterward.
- *General, application-independent heuristics* are based on general measures for the propagation potential of variables and the strength of constraints. Eclipse already provides the two instantiation heuristics "most constrained variable" and "first failure." We added the heuristic for constraint placement "smallest domains first," which is of special relevance for lookahead constraints.

Table 1. Benchmarks giving an idea of the computation times needed to find all solutions of the house aspect shown in Figure 4. In this example, the segmented aerial image contains 1,185 lines and 212 regions. The large number of solutions is due to model symmetry and fragmentation of image features. When a priori knowledge about some feature assignments is available, search time dramatically decreases.

Cache	T_{PLCMNT}	$T_{SOLVING}$	Σ_T	Number of solutions	Cache hit ratio (%)
Without a priori knowledge:					
Disabled	2.45	22.76	25.21	4,608	—
Enabled	1.15	17.87	19.02	4,608	32.52
With knowledge of roof regions:					
Disabled	1.11	4.45	5.56	4,608	—
Enabled	0.77	2.84	3.61	4,608	57.55
With knowledge of front region and top edge:					
Disabled	1.52	1.30	2.82	2,160	—
Enabled	1.06	0.96	2.02	2,160	44

An illustrating example

Our "Interpretation Laboratory" is an interactive tool for the design and evaluation of models, interpretation strategies, and heuristics. This software platform, which is implemented in Eclipse-Prolog and Tcl/Tk, offers immediate access to all system components (such as the extended constraint solver, image and model data, and graphical user interface) supporting a step-by-step extension. All kinds of data are represented by Prolog terms and stored as facts in the database. Therefore, they are immediately accessible at any time and for all reasoning components. The tight coupling to the graphical user interface allows not only the graphical presentation of matching results but also the visualization of the entire reasoning process. Figure 6 shows a screenshot of a working session.

Table 1 lists some benchmarks done on a Sparc-10 using the environment described and are meant as a hint of the processing time's magnitude. The aim was to find all solutions for the house aspect shown in Figure 4. The data are from a portion of an aerial image about four times larger than the one shown in Figure 6. The domain sizes in this example are $|D_{line}| = 1185$ and $|D_{blob}| = 212$. Column t_{plcmnt} shows the time needed for the placement of all 45 constraints (before explicit value generation). Column t_{solving} shows the time for calculating all solutions. All time data are given in seconds.

The large number of solutions looks rather astonishing at first. Apart from symmetries, the fragmentation of one line on the object level into several lines on the image level results in many different possible mappings between the same image part, on the one hand, and the same aspect on the other—thus generating distinct solutions by our current implementation. Focusing on distinct regions, however, reduces the number of solutions to just a bit higher than one.

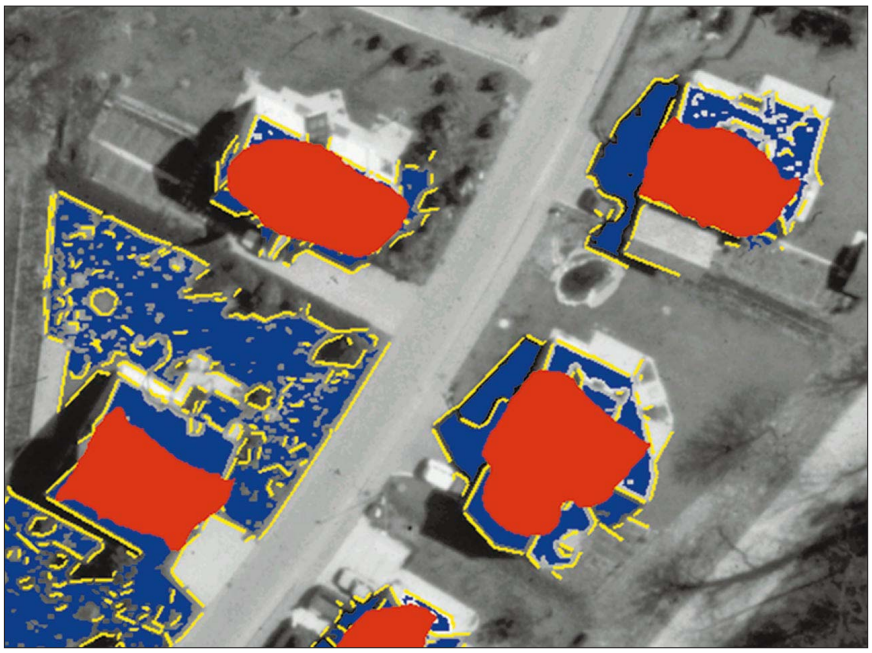

Figure 7. Focusing the search by identifying roof regions. The spots are derived from the analysis of the corresponding geographical area's digital elevation model.

Figure 7 illustrates how applying the tools of the Image Interpretation Laboratory incorporates additional information in the reasoning process. The spots visualize the hypothesis of building positions derived from analyzing a digital elevation model. A DEM is a rectangular or triangulated point raster where every point has a height coordinate. Such DEMs can be automatically derived when stereo images showing the same scene from different viewpoints are available. The spots focus on the features to be considered in the interpretation process. The benefit of such a priori knowledge about possible region features for roof faces is far smaller computation times, as shown in the benchmarks.

TO THE BEST OF OUR KNOWLEDGE, this is the first attempt at applying constraint logic programming over finite domains to object recognition. This is astounding, because constraint solving has its roots in this application domain. The technique of propagating domain reductions emerged from Waltz's seminal paper on the interpretation of line drawings for 3D object reconstruction.[9]

From our perspective, applying CLP(FD) to this task is not straightforward, for several reasons. First, CLP techniques are based on categorical constraints; in real images, expected object parts are unobservable, or constraints cannot be enforced that often. Although the CLP scheme provides methods to efficiently prune the search space, many other functionalities are also needed:

- extendable constraint handlers to allow the implementation of relaxable constraints and wildcard assignments;
- database facilities providing efficient access to the many complex objects extracted from aerial images;
- a universal programming language for implementing model transformations and the overall strategy for building reconstruction; and
- graphical visualization and a user interface.

From a practical point of view, what is important is not only a fine paradigm, but its implementation and embedding in a universal, flexible, and extendible software-development system. Eclipse turned out to be an appropriate platform fulfilling those needs. ■

Acknowledgments

We did this work largely within the project "Semantic Modeling and Extraction of Spatial Objects from Images and Maps," especially in the subproject "Building Extraction," which is supported by the German National Research Council (Deutsche Forschungsgemeinschaft, DFG). We profited heavily from discussions with our cooperation partners Wolfgang Förstner, Volker Steinhage, Felicitas Lang, and André Fischer. We thank the DFG for supporting our work.

THE BENEFIT OF SUCH A PRIORI KNOWLEDGE ABOUT POSSIBLE REGION FEATURES FOR ROOF FACES IS FAR SMALLER COMPUTATION TIMES, AS SHOWN IN THE BENCHMARKS.

References

1. P. Suetens, P. Fua, and A.J. Hanson, "Computational Strategies for Object Recognition," *ACM Computing Surveys*, Vol. 24, No. 1, Mar. 1992, pp. 5–61.
2. J. Jaffar and M.J. Maher, "Constraint Logic Programming: A Survey," *J. Logic Programming*, Vol. 19, No. 20, 1994, pp. 503–581.
3. P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, Logic Programming Series, MIT Press, Cambridge, Mass., 1989.
4. M. Wallace, S. Novello, and J. Schimpf, *ECLiPSe: A Platform for Constraint Logic Programming*, tech. report, Imperial College, London, UK, 1997.
5. C. Braun et al., "Models for Photogrammetric Building Reconstruction," *IEEE Computer Graphics & Applications*, Vol. 19, No. 1, Jan./Feb. 1995, pp. 109–118.
6. A. Fischer, T.H. Kolbe, and F. Lang, "Integration of 2D and 3D Reasoning for Building Reconstruction Using a Generic Hierarchical Model," *Semantic Modeling for the Acquisition of Topographic Information from Images and Maps*, W. Förstner and L. Plümer, eds., Birkhäuser Verlag, Basel, Switzerland, 1997, pp. 159–180.
7. J.J. Koenderink and A.J. van Doorn, "The Internal Representation of Solid Shape with Respect to Vision," *Biological Cybernetics*, Vol. 32, 1979, pp. 211–216.
8. W. Förstner, "A Framework for Low Level Feature Extraction," *Proc. European Conf. Computer Vision '94, Vol. II*, *Lecture Notes in Computer Science*, No. 801, Springer-Verlag, New York, 1994, pp. 383–394.
9. D.L. Waltz, "Understanding Line Drawings of Scenes with Shadows," *Psychology of Computer Vision*, P.H. Winston, ed., McGraw-Hill, New York, 1975, pp. 19–91.
10. G. Vosselman, "Relational Matching," *Lecture Notes in Computer Science*, No. 628, Springer-Verlag, New York, 1992.
11. T.H. Kolbe, "Constraints for Object Recognition in Aerial Images—Handling of Unobserved Features," *Proc. Fourth Int'l Conf. Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, No. 1520, Springer-Verlag, Berlin, 1998, pp. 295–309.

Thomas H. Kolbe is a researcher at the University of Bonn, Germany. He received his PhD from the University of Vechta, Germany, with a thesis on building recognition. His research interests include building recognition, constraints, logic programming, and information systems. Contact him at the Inst. for Cartography and Geoinformation, Univ. of Bonn, Meckenheimer Allee 172, 53115 Bonn, Germany; tk@ikt.uni-bonn.de; www.ikt.uni-bonn.de/kolbe.

Lutz Plümer is full professor and chair of Geoinformation at the University of Bonn. He received his PhD from the University of Dortmund with a thesis on termination proofs for logic programs, and his *venia legendi* from the University of Bonn with a thesis on verification of parallel logic programs. His current research focuses on geoinformation systems. Contact him at the Inst. for Cartography and Topography, Univ. of Bonn, Meckenheimer Allee 172, 53115 Bonn, Germany; Lutz.Pluemer@ikt.uni-bonn.de; www.ikt.uni-bonn.de/pluemer.

Armin B. Cremers is full professor and chair of Computer Science III at the University of Bonn. His scientific research is in the areas of information systems and AI. Contact him at the Inst. of Computer Science III, Univ. of Bonn, Römerstr. 164, 53117 Bonn, Germany; abc@cs.uni-bonn.de; www.cs.uni-bonn.de/~abc.