Spatial Template Recognition: Learning, Complexity and Implementation

Steven Woods*
University of Waterloo

Abstract

This report details three aspects of work-in-progress involving the automatic or interactive recognition of spatial templates using constraint satisfaction techniques. The first section discusses the incorporation of information about template models acquired through interactive search for template instances into *updated* models; a preliminary study the complexity of the spatial template recognition problem and determination of *key problem aspects* which may most immediately affect this complexity; and an evaluation of the applicability of certain recent attempts to apply a general constraint satisfaction strategy to real problems to the task of recognizing spatial template instances.

*Computer Science Department, Waterloo, Ont., Canada, N2L 3G1. Email: sgwoods@logos.uwaterloo.ca

Contents

List of Figures

1 Learning refinements to Spatial Template Models based on false-positive and false-negative examples

Spatial templates are used to model a spatial concept held in the mind of a domain expert or set of domain experts. These templates consist of objects of varying types that share a set of spatial relationships or constraints that have been recorded as representing the concept in question. These templates define a space of instances that "fit" the template constraints. A situation is a set of objects of varying type and attributes arranged in a finite area. Search algorithms have been applied previously to attempt to locate one or all mappings from a particular situation to a particular template (or set of templates) such that the template constraints are satisfied by the mapped situation objects. In addition, previous work has focused on developing expert-interactive search strategies and representation models for identifying these template instances.

Due to inaccurately specified templates or error in the specification of a situation, it is possible that a complete search method may locate a mapping from a template to a situation that a domain expert may indicate is *not* representative of the concept the template is meant to model. Similarly, it is possible that a complete search method my fail to locate a mapping that a domain expert identifies as a representative instance of the concept the template is modeling. These cases are known as *false-positive* and *false-negative* respectively. Based on these identified error cases, we wish to modify our template model to more accurately represent the ideal spatial concept.

One previously developed approach to dealing with uncertain template definitions and "noisy" situational data is to represent the template as a hierarchy of templates, where an "abstract" template captures the "key factors" of a spatial concept, and subsequently more specific templates capture more, possibly less integral, detail of the concept. Use can be made of this hierarchy in identifying possible template matches in situations even when the mapping between situation and most-specific template is not present. Here we are interested in learning error cases not only within a single level model, but also in determining how refinement at a particular level should or should not propagate to other levels of a template hierarchy.

In this work I will discuss the role and representation of spatial templates as approximations of an ideal spatial concept, as well as outline how the use of expert-identified errors as catalysts for model refinement corresponds to learning improved spatial templates. This work reports ongoing research towards both interactively and automatically learning revisions of spatial templates and in refining spatial templates represented in constraint graphs according to rules for constructing abstract representations of these graphs which have been applied in various domains, and reported in earlier work.

1.1 The problem

1.1.1 Introduction

> *Intelligence* means getting better over time ... Real AI means a machine that learns ... AI depends on computers that have real knowledge in them. Thus, the crux of AI is in the representation of this knowledge, the content-based indexing and modification of this knowledge through the exercise of this knowledge (Schank 1992, p. 42,p. 47).

This work is about learning. Specifically it is about learning how to improve models through failures of a model to "fit" or classify new, previously unseen examples. In this section I will introduce a specific domain, Spatial Template Recognition (STR), in which I will be discussing model revision learning, and position the learning aspects of learning model revisions in the broader context of the specific domain problem. In later sections I will further examine what information needs to be learned, and discuss possible methods of integrating this new knowledge into useful and more accurate models. This paper is intended as an exploration of the STR problem with regards to representation and representation revision using machine learning. I will examine current work in the domain itself, approaches to model representation using abstraction, and learning approaches in several related areas.

STR is the problem of finding an instance of some predefined configuration of objects that may be deployed amongst a given situation, or set of object instances. An example of a spatial template in a two-dimensional molecular domain would be a "map" of a typical molecule of some type that describes the molecule in terms of the relative positions, types and orientations of the various atoms of the molecule. An example of a situation might be some two-dimensional sample containing many instances of different types of molecules, and the question formed by STR would be: "Is there an instance of this molecule in this solution?".

Other domains for application of STR are more dynamic, for instance, consider a group of objects on a gameboard being deployed according to some opponent strategy. During a game, a player often would like to identify the plan of his/her opponent. While this determination can depend heavily on the perceived context of the game at a particular moment, the personality characteristics, experience and goals of ones opponent and upon many other factors, a "quick" or "trigger" heuristic that suggests to us the intent of our opponent can be our identification of some representative or typical pattern in either his movements or static board positioning. Basically, given that we understand certain "key" board configurations that our opponent may wish to utilize under certain circumstances, it is possible that we may recognize a particular board configuration and infer that its existence is motivated by a particular plan in our opponent's mind[1]. Thus, this kind of limited pattern recognition can be an integral component of a larger plan recognition system.

[1] Of course for the moment ignoring the obvious misdirection possibilities

As a specific example of this opponent inference, in military deployments, units are positioned according to specific goals and opposition deployments. The "group" or collective deployment patterns are consequently representative of these goals. Typical configurations include such group and activity combinations as "tank regiment in advance", "rifle regiment in attack". So, if we could take a "picture" of the current deployment positions, filter out any "noise" in the form of overlapping or confusing deployments, and identify the elements involved in a particular maneuver, we have essentially completed one part of an intelligence analyst's task in recognition of a local plan.

1.1.2 Model specifications

In many of the domains of STR, experts use expertise built up over a long period of time to construct a "typical" model of each of many templates. Each of these models may be physically represented or held mentally by individual experts. To varying degrees these models cover a breadth of example instances in their domain, and essentially capture an elaborated domain *concept* as is described formally in the literature of machine learning, one instance of which is Anthony & Biggs (1992). Typically, one standardized generalization is accepted or agreed upon as canonical, and is often referred to as if it were a "definition" of a particular concept for other more naive initiates to the domain. Often these canonical definitions include specific details of the variation allowed within the concept. For instance, a concept in the military domain might define a deployment pattern composed of individual units each related to some subset of the others according to spatial constraints. These constraints could be *rigid* such as "The lead unit must be *ahead of* the trailing unit", or *flexible* as is the case in "lead unit A must be between 5 and 10 km of lead unit B". The later range-constraint is referred to as flexible in this context since it has its tolerance encoded directly. Of course this flexible constraint is still "hard" in the sense that anything (however slightly) outside 10 km or inside 5 km will result in a failed constraint. These arbitrary cut-off points rarely reflect the real-world situation, and we will see subsequently how to handle these more smoothly using abstracted concept definitions.

1.1.3 Learning as model revision

Given a concept or template definition including both *rigid* and *flexible* constraints, we may attempt to systematically (either manually or automatically) identify these template instances in a specific situation by locating a set of object instances that satisfies all of our template constraints[2]. If a particular template concept "exceeds" in some sense the "true" modeled object concept, then some positive matches may be made which would be classified in the negative by some "oracle" or expert. If the concept expression is unduly restrictive, then some instance groupings that the expert would classify as positive are identified nega-

[2] Unfortunately, there are an exponential number of possible sets of object instance mappings to template slots.

tively based on the model. The role learning may play in such a framework is to integrate these misclassifications identified by an expert into the current concept model so that these and hopefully other unseen examples are classified correctly. Essentially we are interested in revising a given model to more accurately reflect the expert's belief about the concept in question. This model revision will be the first half of the focus of this paper.

1.1.4 Abstracting models

The inaccuracy of expert knowledge modeling has been a concern in the design of most expert systems, and the problems of spatial template inaccuracy have been discussed briefly in the work of both Woods (1993*a*) and Lapointe & Prouix (1994). In Lapointe & Prouix (1994), the problem is addressed by matching templates to situations probabilistically based on the existing template definition, thus hopefully avoiding the "model too restrictive" failures by at least classifying the "missed" case as possible in some measure, and avoiding the "model too accepting" problem of too many matches by some ordering based on a measure of the template-situation fit. In contrast, Woods (1993*a*) proposes a template concept representation model where the spatial templates are described not by a single constraint based description, but rather by a connected hierarchy of concept models. This hierarchy defines a concept at ordered levels of specificity, much as people might hierarchically define objects within classes. For instance, the chair on which you are sitting might be considered most abstractly a member of some class (say) *chair*, and more specifically as a member of the a subclass (say) *office chair*, and then most specifically (say) as a member of the subsubclass *computer office chair*. Within this framework we have three levels of expression of the concept "your chair", each conceivably more specific than its predecessor. The intent here is that in absence of complete context, we might benefit by first trying to identify your perch as a "chair" before jumping ahead and trying "office chair". We may consider that the essential "chairness" of the concept is encoded in the most abstract model, and the additional "officeness" is then mapped from chair to office-chair. Thus, we first identify the chairness of an object and then attempt to refine this further according to possible specified mapping(s) in a particular hierarchical concept definition.

This abstracted concept model maps well to the approach of a human expert in the spatial template identification domain. This expert tends to look for key (or obvious) features of a template (like four legs of a chair, for instance) and then attempts to resolve additional detail if possible, essentially "zooming in" with a conceptual magnifying glass. In this way, the expert could be said to be forming an abstract model in their mind of what a "match" looks like in terms of carefully but loosely defined spatial relationships and other key relationships among the objects.

1.1.5 Inaccuracy in representation and perception

Some of our earlier work (Woods 1993*a*, Woods 1993*b*) has focused on attempting to model these templates in consultation with real experts in an effort to construct an automated tool to assist in the recognition of these templates in object sets that are often very large, and which may quite possibly be full of error and inaccuracy with respect to object instance position, orientation, or other vital characteristics. Thus we see that the STR problem faces inaccuracy at both ends: in the modeling of a particular template concept, and in the perception of the object instance world in which we wish to identify some concept model.

Using a hierarchically specified template concept, a template match may be made at a very abstract level based on only "key" or obvious features. This parallels the concept of viewing an object at a distance where only the outline or "largest" or most "vivid" details stand out. Essentially we get an idea of what an object is at this distance, and then move closer for more detail and refine our belief somewhat. One can imagine in this way trying to analyze a large situation with many different template instances, and examining the situation by focusing in on specific details when examining one possible match, and then essentially taking a step back to view the whole situation at a higher level. This process of "zooming in" to resolve specific objects from the larger palette, and "zooming out" for the larger perspective follows the divide and conquer method that has long been suggested as a sensible way to attack many problems where the complexity of the whole task seems insurmountable. More will be said about this approach later in this paper.

1.1.6 Learning and hierarchies

Model revision was discussed earlier in this section, but now we are faced with an even more difficult problem, that of learning revision to a hierarchical model. Misclassifications may result in model revisions at a certain abstract level that may (or may not) need to propagate to other levels of the hierarchical concept representation. The examination of some of the various propagation possibilities will form the crux of this paper.

1.2 Spatial template representation at a single level

1.2.1 Template models and constraint graphs

A typical model of an expert's beliefs about a particular spatial template as outlined in Section 1.1.2 may be modeled as a graph of constraints amongst the elements or the template. For instance, a molecule might be composed of atoms which obey positional constraints amongst themselves. The "structure" imposed by these constraints is the essential "templateness" of the definition. Figure 1 details a simple template capture in a constraint graph consisting of three nodes, 3 constraints between pairs of nodes, and four constraints on various nodes. The constraints on a single node represent a constraint that a node or slot maintain a particular attribute: in this case "type" is represented by one of three possible

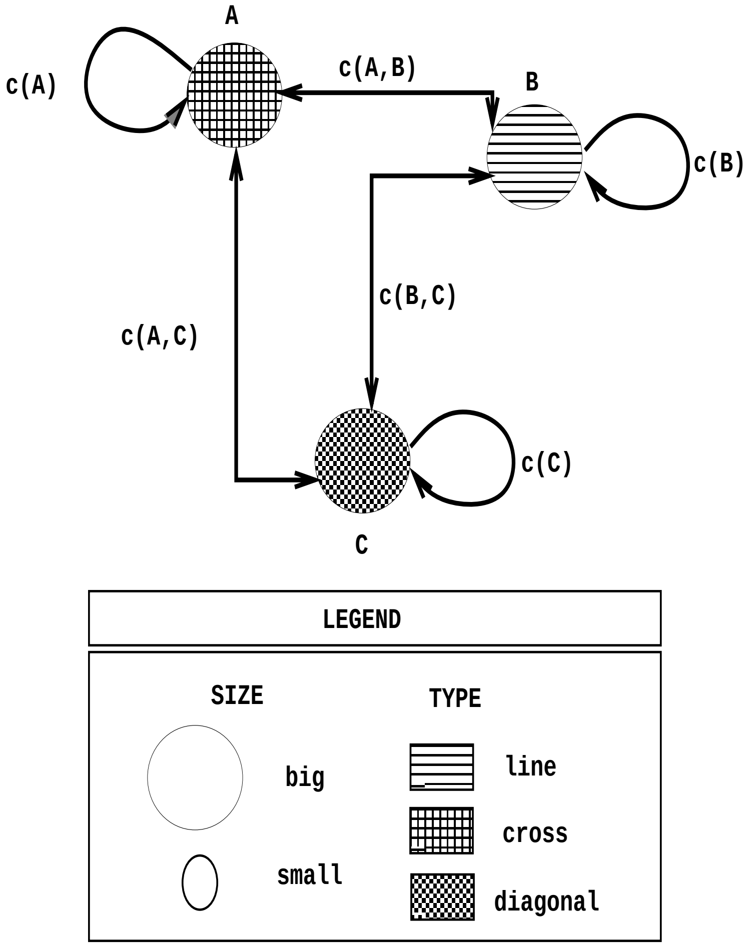

Figure 1: A simple template definition via a constraint graph.

colorings of the nodes, and another attribute or attribute set (including for example, "size") might be represented by a reflexive arc on each node. The spatial constraints of the template are represented as constraints among the nodes, for instance, C(A,B) might specify that A and B must be between 1 and 2 km apart perhaps. These constraints need not be binary, and may involve more complex relationships about relative positions. A legend in the figure identifies clearly the possible attribute values for "size" and "type".

1.2.2 Template instance identification

Given such a template representation we wish now to attempt to locate instances in a given situation, such as that portrayed in Figure 2. In this situation, we have fourteen objects, each either big or small in size, and each "typed" in one of three ways, just as our template definition earlier was typed. Now, if we had specified precisely the constraints C(A,B), C(B,C) and C(C,A) of Figure 1, we could have attempted to locate an assignment of situation objects to template nodes that satisfied the constraint set. Each such satisfying assignment would have constituted a solution to our template model definition. For instance, if we took the constraints in the template of Figure 1 as representing physical distance as drawn, then we could observe that one solution "mapping" or assignment from template

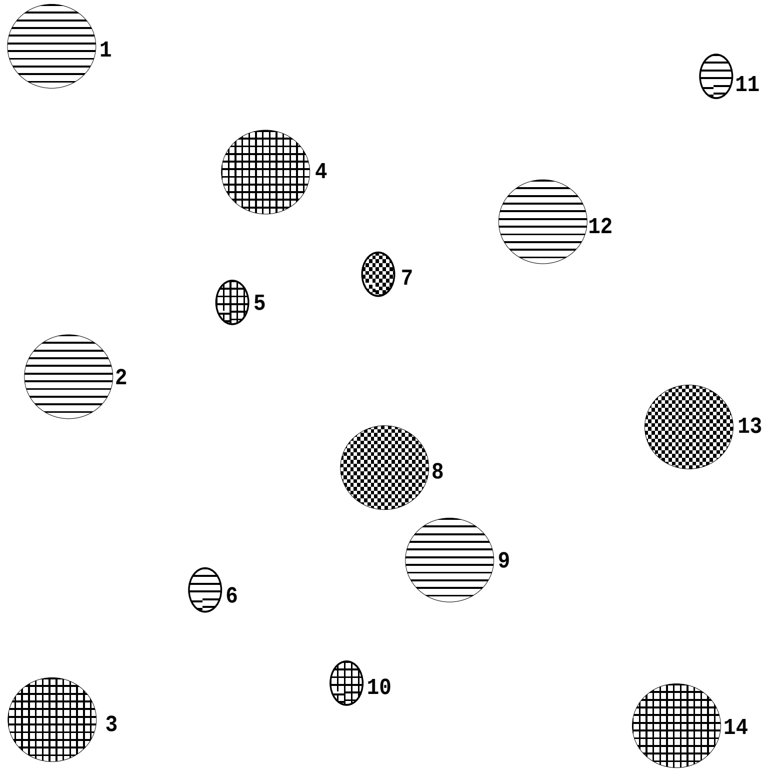

Figure 2: A situation in which template instances may be found.

node to situation object would be: (A,B,C) $\Rightarrow$ (4,12,9). While the problem of locating all of these mappings is NP-complete[3], the complexity of this problem does not necessarily indicate that in certain domains a search-based approach will not suffice. For a detailed discussion of the search and performance aspects of solving this and other similar Constraint Satisfaction Problems (CSPs), the reader is referred our earlier work in Woods (1993*a*) and that of Prosser (1993) and van Run (1994). A discussion of the complexity of this problem in relation to some other related "hard" problems such as graph and subgraph isomorphism may be found in Section 2.

1.2.3 Template instance misclassification and learning

Section 1.1.3 motivates the importance of learning in such a matching system. Since our template model is only an approximation of some concept, it must be able to change to accommodate new information about the concept. Since our system is a classifying one, it may err in one of two ways:

- It may identify a mapping between the template and situation objects as a template match that some oracle or expert denies is a "true" instance of the concept to be modeled, i.e. a *false positive example.*
- It may fail to identify some particular mapping of situation objects that an expert identifies as a "true" instance of our template concept, i.e. a *false negative example.*

Figure 3 depicts this system of an automatic Model Matcher identifying a false-positive model match of a template concept in a situation. The Expert Classifier (or oracle) identifies the positive match as false, and the expert can potentially explain this identification in terms of a possible revision. In various domains, one or the other of these classification errors may be thought of as "more important" than the other, however, for the purposes of this paper we will consider them equally bad, and we would like to be able to modify our model to properly classify these examples with the hope that our template will more accurately model the concept, and thus learn from these examples.

Case 1: False negative example model revision How can we change our model? Well it is immediately apparent that *false positive examples* suggest a "tightening up" of the model so that it is more restrictive, and *false negative examples* suggest a "loosening" of the model to that it is more accepting, at least for the case of the new examples. If we also assume that the model that we have so-far is still valuable and representative in most cases, then we will want to perturb our existing model as little as possible. Specifically, in Figure 4, we see the space of all possible situations that would be categorized positively by the template as opposed to those that "should" be categorized positively according to some "ideal" expert or oracle from which we wish to learn to improve our template. The initial template space

[3] For example, consider a reduction to graph coloring, which is a special case of CSP.

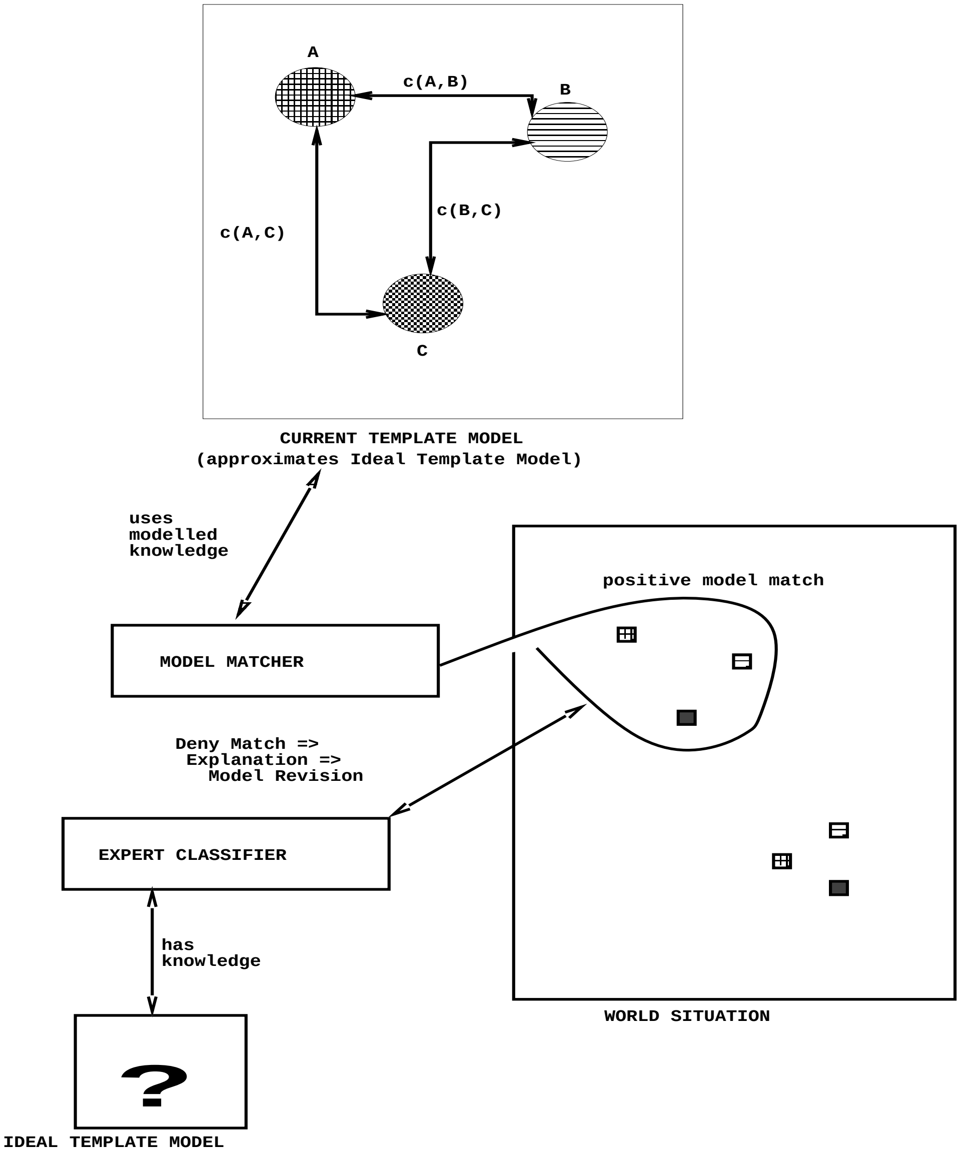

Figure 3: Expert revision of automatic classification.

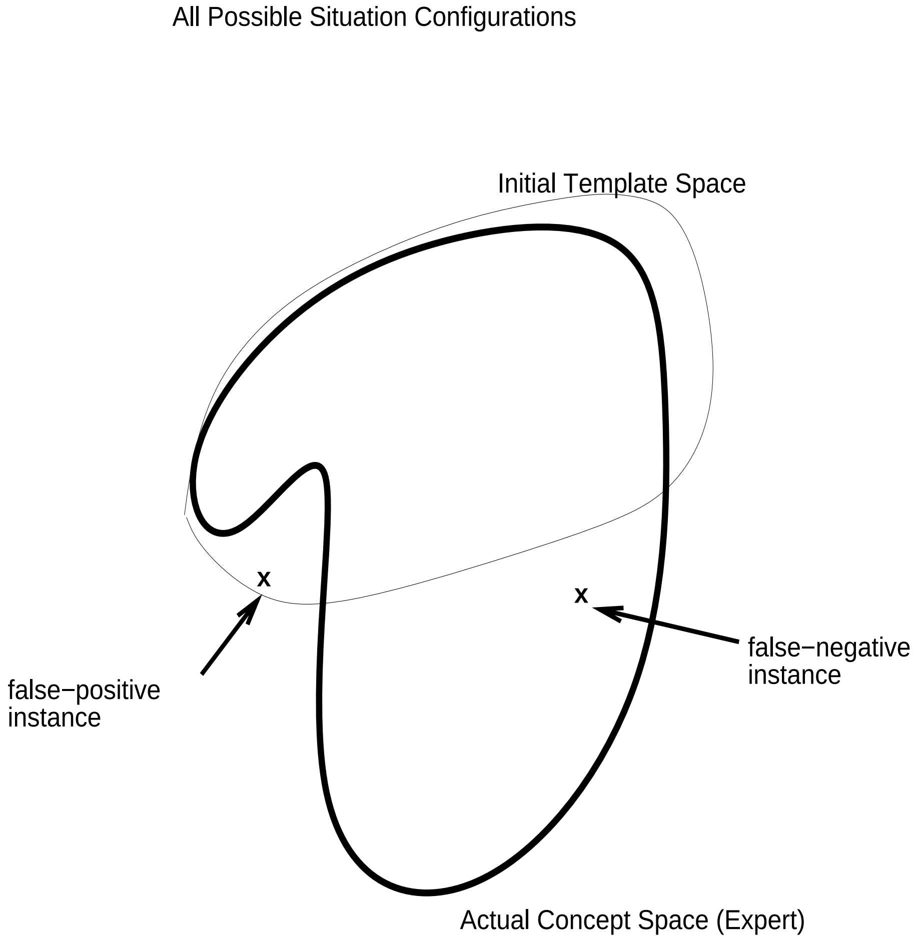

Figure 4: The initial coverage of a template vs actual concept.

exceeds the expert or ideal concept space in certain areas, thus allowing for the existence of false-positive classifications, and the expert space exceeds the template space in some areas, thus allowing for false-negative classifications. Figure 5 shows how we might like to improve on our model so that the identified errors are accommodated. As is made clear

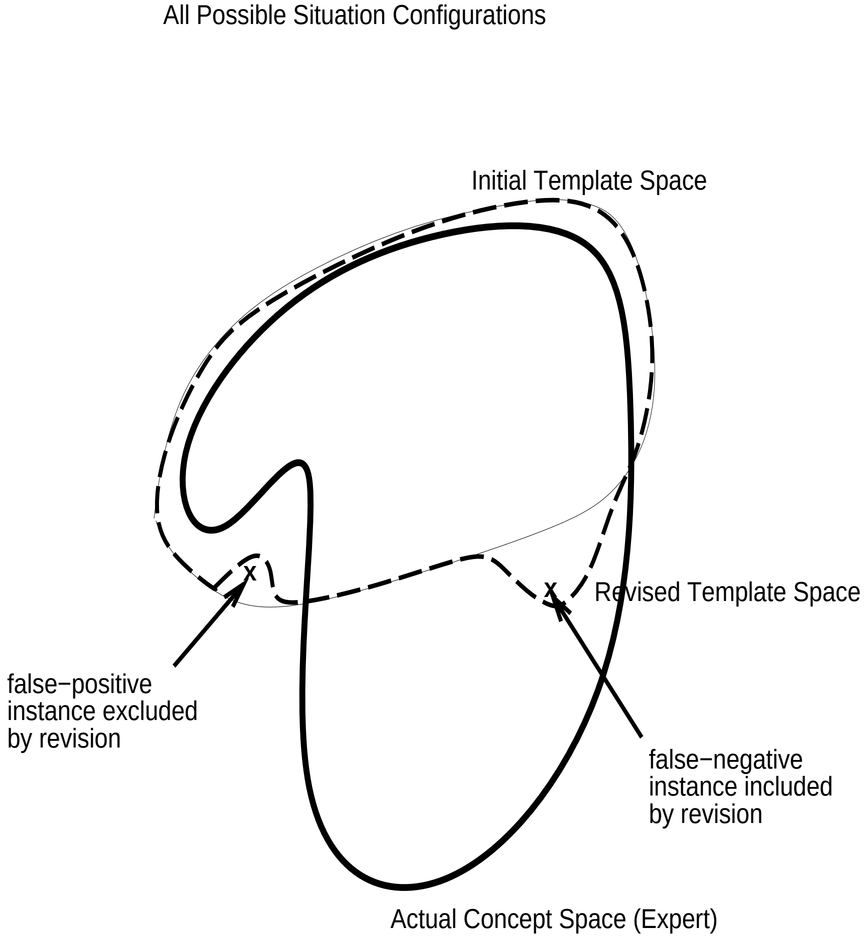
All Possible Situation Configurations
Initial Template Space
x
x
Revised Template Space
false−positive instance excluded by revision
false−negative instance included by revision
Actual Concept Space (Expert)

Figure 5: The revised coverage of a template vs actual concept.

from this figure, there is a possibility that changing the space to attempt to accommodate a single false-positive example or false-negative example may result in a space coverage that also excludes or includes other than the intended examples. Essentially we are changing the shape of the space and excluding or including whole ranges of situations, not necessarily only those that we wish to affect. A question arises as to how we may learn our new example(s), and yet minimally change the existing template concept in some sense.

In the case of a false-negative, we will wish to allow our model to be more accommodating. Essentially we wish to "relax" some aspect of our model so that the excluded example is included. Since we are assuming that our initial model is "largely" correct and only needs to be refined, we also assume that a false-negative example, classified as a positive instance by an expert, is a "near-miss" to our model. In terms of CSPs, a "near-miss" essentially means that some subset of the total constraint set failed to be satisfied during the situation matching process andor some nodes failed to be matched to situation objects. These "failures" are at least partially known as a result of the failure of the matching process (i.e. the "reason" for the failure" is the constraint that was determined to not be satisfied, or the node that had no acceptable situation object correspondence).

We would like to relax our model "minimally" in some sense in order to include the given example. In Freuder & Wallace (1992), the concept of Partial Constraint Satisfaction Problems (PCSP) is introduced in order to systematically define the relaxation of constraint satisfaction problems. Our problem of matching the given template (essentially the CSP definition) to the given situation (essentially defining the domain of the variables or nodes in the template) requires that we locate a "minimally" simpler template or CSP definition that will accommodate our new example. In essence, there are three ways in which a CSP or template may be "relaxed" or simplified:

1. The domain of a particular variable can be extended to include more domain values. In our context, this means that the node constraints on size or type may be relaxed to allow for additional possibilities.

2. A variable or constraint may be removed from the original CSP. Essentially, either some node or some constraint in the template is dropped from the concept definition entirely.

3. The domain of a constraint may be enlarged. A constraint on spatial distance, for example, might be changed from "between 1 and 2 kilometres" to "between 1 and 3 kilometres".

In Figure 6 we consider the original CSP as the common root of a graph representing all possible relaxations of that CSP. Each node in the graph represents simpler CSPs obtained by applying a sequence of relaxations. This graph is not a tree since application of relaxation steps in different orders can give the same PCSP as a result. We may make use of this PCSP space by searching in some fashion through it for the "first" relaxed CSP that classifies our example properly. Subject to the approval of some template revision expert, this relaxation could replace the previous template definition. In Yang & Fong (1992), we see an application of this relaxation technique that heuristically searches for relaxed solutions in a scheduling environment, providing solutions that are "closest" to those desired in the sense that a problem is solved that is "minimally changed" from the original problem.

In this fashion, search can yield *many* possible relaxations that also satisfy our demand of accommodating the new example. Relaxed CSPs at the same "level" of the PCSP space

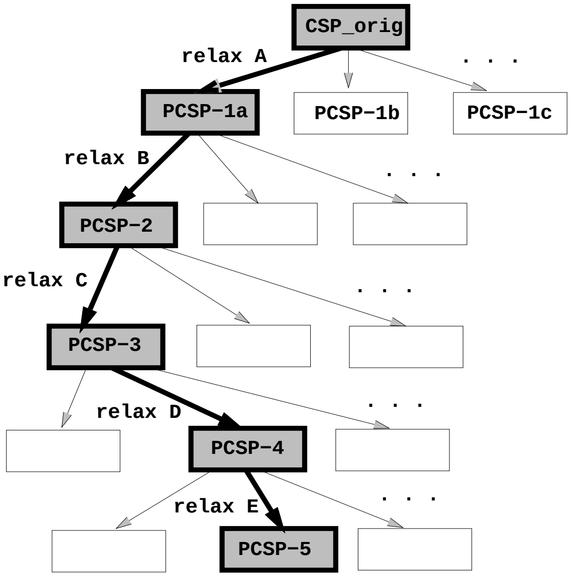

Figure 6: One abstraction hierarchy in a PCSP space.

might be considered "equivalently relaxed" in some sense, or perhaps relaxations of constraints might be preferred over relaxations involving node removal. Further, the definition of what exactly constitutes a "constraint relaxation" will need to be clarified at some length, since spatial constraints are often continuous, and a step by step relaxation suggests some discrete relaxation of a constraint at one step. The decision about which relaxation(s) to "prefer" can be extremely important since the perturbation of the "space" of situation examples made to accommodate the single example may have allowed for many other possible matches, some of which are undesirable. Since we are using the template as a "memory" of previously seen examples and not using the previous examples explicitly we wish to avoid as much as possible changing the space so that previously (correctly) rejected negative examples are now included.

What we have identified with this relaxation technique is a general approach to "minimally" relaxing our CSP (or template) definition so as to learn a new CSP (or template) definition that accommodates a new false positive example. Of concern is that the relaxation used in the refinement "makes sense" with regards to our concept in question. Simply using the minimal relaxation may accommodate the change with little adjustment to the actual model, however, selection of the actual constraint or set of constraints that should be used in this change is best accomplished when approved as sensible by a domain expert or oracle if available. What needs to be done in future work is to further identify the way in which these relaxations are specified in terms of allowable and desirable constraint relaxations.

Case 2: False positive example model revision In the case that we wish to learn a false positive case into our concept model, we are faced with a problem symmetric to the false negative case in that we wish to create a model that is minimally (in some sense) more *restrictive* as opposed to more *relaxed*. We may apply our PCSP space generation and search approach precisely as before, with the exception that now we will restrict the CSP at each stage of the PCSP space generation. A candidate template for replacing the current template is one that is "minimally restricted" from the original, and yet successfully excludes the false-positive example. From the previous example we can see that there are three ways in which a CSP or template may be "restricted":

1. The domain of a particular variable can be restricted to include less domain values. For instance, the node constraints on size or type may be restricted to allow for fewer possibilities.

2. A variable or constraint may be *added* to the original CSP. This is problematic since without further contextual information, it seems to make little sense to randomly insert a constraint or variable. There are at least two interesting options at this point: we can refer back to the domain expert to suggest a constraint, variable, or set of constraints and variables to add to restrict the template further, or, we can attempt to incorporate some other learning algorithm here that might be able to identify commonalities amongst other positive examples that would exclude this negative example. We will leave the second option for future research and mention it here only as a point of interest.

3. The domain of a constraint might be reduced. A constraint on spatial distance, for example, might be changed from "between 1 and 2 kilometres" to "between 0 and 1 kilometres".

As was the case for the false negative example, the perturbation of a template based on a single example with the intent of accommodating that example affects more than the small part of the situation space indicated by the single example. Excluding a false positive essentially reduces the space covered by the template model and could possibly exclude previously (correctly) identified positive examples. The conception of "minimally" perturbing the model applies equally well in this case, however, there are no guarantees about the space more accurately reflecting the "true" concept that are immediately apparent. The key to the utilization of this PCSP relaxation and restriction approach is that the expert guides the process and is in a position to verify the model frequently.

It is interesting to note that in the false-negative case, we may obtain an explicit machine-generated explanation of the failure of a particular match based on the constraint values when the mapping from template to situation objects is made. This machine-generated (partial) explanation might potentially be used as a starting point for model revision. However, in the false-positive case, the expert is identifying a particular instance where the automated matching process succeeded, consequently the explanation of the model failure must come entirely from the expert, and any automatic restricting process would start without any possible "helpful" starting point.

1.3 Spatial template representation with multiple levels

To this point we have envisaged the template-based concept approximation at a single level represented by a CSP. We have attempted to show that this single level CSP could be refined through a learning process based on the concept of selecting minimally relaxed or minimally restricted versions of the original CSP in a PCSP space. In practical application, the single level template model we have described as an approximation of the actual template concept has several important failings. First, the initial error in defining the template based on expert information is often inaccurate in practice. In a domain where there are only a few positive examples to work with in refinement, this initial lack of knowledge can serious impair the accuracy of the matching process. Second, the situation that is being matched often contains information that is only partially known (for instance not all objects are typed or sized correctly, if at all), and when it is known, it may be only probable rather than certain. Consequently, the matching process may be attempting to identify poorly defined templates with "noisy" situations. If our matching process was very "rigid" and discounted all but certain matches we would be in trouble since this circumstance might rarely occur. In Section 1.1.4 the concept of abstracting the template model so as to allow for partial matching was outlined as one way to help alleviate the problem of identifying matches in uncertain circumstances.

The PCSP space introduced in Section 1.2.3 and detailed in Figure 6 can be utilized as a framework on which to build a hierarchy for a particular template concept. In building a PCSP space, we have constructed an abstraction space in which the least constrained PCSPs are the most abstract, and the most constrained PCSPs are the most specific instances of this CSP. However, there are many possible versions of the least constrained PCSPs. How do we differentiate amongst these in some fashion? What we would wish to do is to arrange these PCSPs into a more systematic hierarchy where some domain knowledge can be used to structure the hierarchy that we will use to define a template concept.

Consider that we could extract the "key" or "critical" information from a CSP, and form a "most abstract" PCSP (call it PCSP-3) which encapsulated this knowledge. Next, we extract several common distinguishing features or identifiers which would reflect the way in which the set of partial solutions is limited. Each of these sets of distinguishing features is applied to PCSP-3 as an inverse instance of the previously described relaxation techniques, specifically we can now add a variable, restrict a variable domain, or restrict a constraint domain. Essentially we now construct the more constrained PCSP-2, then PCSP-1, and finally our complete problem, CSP. Figure 6 on page 15 outlines this process. The number of stages or the groupings of the sets of "making stricter" actions can be dependent on a particular problem, and on the typical features or keys of that domain.

The advantage of this concept definition as a systematic structuring of the hierarchy of PCSPs is that a constraint satisfaction approach which solved PCSP-3 first resulting in a set S1 of solutions, then attempted to solve PCSP-2 for S2 given the initial starting point as each of the solutions in S1, and so on, would essentially have the quality of iteratively refining rough or partial solutions into specific or total ones. The effort spent arriving at the set S1 is not spent again in solving for set S2, as we build on the work done before. We are attempting to deal with the possibly flawed template model and the uncertain situational information by first attempting to identify instances that seems to satisfy the "key aspects" of the template, without additional detail. These "key aspects" may be thought of as the "minimally sufficient" portions of the template that would suggest a possible instance. Thus, "rough" matches are found first based on key indications, and then these rough matches are resolved by attempting to resolve the initial match into a match with the less-abstract template model in the hierarchy. This process can be repeated for as many levels of abstraction that are defined. We then have an iterative-improvement method for matching, where extra work provides more confidence in certain matches over others based on the identification or absence of less-abstract template features in the situation.

The process described above works very much in a top-down manner, solving PCSP-3 giving set S3, then refining S3 in solving PCSP-2 giving S2, finally refining S2 in solving CSP, giving our final solution set. One obvious difficulty with such an approach is that it finds all of the PCSP-3 solutions in S3 before attempting to find any solution elements for S2, for example. Basically each level is completely solved before the subsequent level in a breadth-first type of manner. This approach has the advantage that if search were halted at an arbitrary point in time, all of the "best-so-far" answers are on the same level. It has the disadvantage that time spent searching the less constrained CSPs might have resulted in more concrete solutions. In Woods (1991), a tradeoff search strategy known as 'Left-Wedge" is described for exploring the multiple level of abstraction solution space which progresses downwards towards more concrete solutions, AND from left to right through the solutions at each level at the same time. This strategy is a variation on A* search which can provide us with both concrete solutions and less concrete solutions as more time is spent searching.

To some extent, the selection of an approach for this type of search depends highly on the way in which we would wish to take advantage of interaction with the user during search. It is conceivable in a certain case that a user may wish to view and prioritize or eliminate higher level solutions in order to guide the discovery of concrete solutions.

Figure 7 details one level of an abstract concept definition based on our earlier template example. In this figure we see an abstract or generalized template labelled TA_0, and a more specific or specialized template labelled TA_1. Between these two templates is the "hierarchical glue" that maps both from specialized to general and general to specialized, detailing the exact operations (relaxing operations from specialized to general, and restricting operations from general to specialized) that define a path in the PCSP space that has been identified at initial template definition time as representative of the "key features" and "subsequent evidence" for this template.

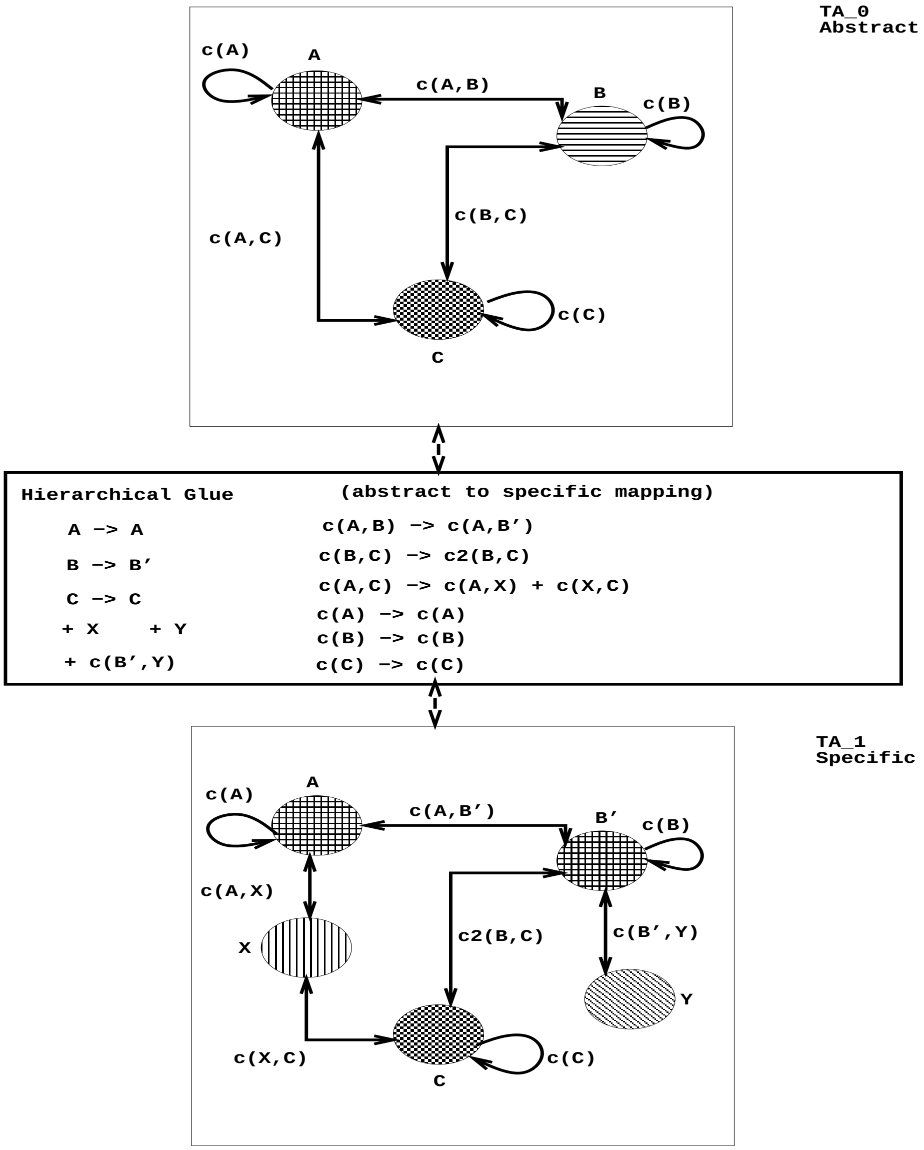

Figure 7: Explicit expression of representation change.

We can see that hierarchical specifications improve over single level specification for locating spatial templates in at least the following ways:

- The inaccuracy inherent in the representation of spatial templates and in the representation of situations makes a single-level model very brittle or rigid in terms of searching for mappings with a situation, and consequently particularly subject to errors in classification even in near-miss cases.

- The hierarchical structure provides a degree of "Anytime-ness" to the matching process, and allows us to iteratively improve confidence in solutions with additional effort in search.

- Abstraction support a model of interaction of user and search, particularly with the anytime search-and-refine type of approach. Matching can be done in an "intuitive" top-down fashion from "key factors" to "additional detail", thus providing matches at the greatest degree of detail or confidence possible, but still providing partial (abstract) matches in the absence of either time to search further, or when the template or situation error has prevented more detailed identification.

Our template model now has multiple levels, with the levels mapped by connecting "functions". While this new structure may provide flexibility in recognition in noisy situations and may still provide "high level" matches when a complete match is non-existent, the problem of refining a template model based on expert identified false-positive or false-negative cases has become significantly more complex. In the next section we will outline the cases that present themselves, and suggest a starting point for a method of learning by refinement of our model.

1.4 Refinement of abstracted template models

Many earlier works in designing abstraction models in other domains including Sacerdoti (1974), Yang & Tenenberg (1990), Christensen (1990), Woods (1991), and Yang, Tenenberg & Woods (1993) often involved specification of abstraction hierarchies with no specific connection between the hierarchical levels. Other work in abstraction and philosophy has suggested that this type of abstraction is a poor reflection of the way people form abstractions. A richer model that is often described, for instance in Holte, Zimmer & MacDonald (1992) and Holte, Zimmer & MacDonald (1993), is one where the "mapping" between layers is at least partially specified. Further, this mapping in describing all or part of the way in which one level of specification can be transformed into another, provides a perfect framework in which changes in one level of a hierarchical model may be reflected in other adjacent levels. Effectively, changing our belief about what constitutes an accurate generalized description of a concept *may* affect what we believe is an accurate specific description of that same concept. Further, a change in belief about a specific description may entail a change in a generalization. Accommodating these changes in belief about an accurate depiction or representation of a domain concept into the representation itself is precisely what we mean as "learning" within a hierarchical spatial template model.

The process of identifying template matches or partial matches in a situation was described in the previous section as a "top-down" method, where the most abstract model is matched to a situation before a less abstract model. Further, any abstract matches are refined according the the mapping to less-abstract detailed in the "hierarchical glue". Rather than a single case covering each of "false positive" or "false negative" matching, we now have the more complicated problem of determining how to modify our entire "hierarchy" based on expert identified errors at various points in the hierarchy. Identified changes to a level of the hierarchy may imply the need to propagate these or similar changes upwards or downward, or in some cases, even suggest the creation or deletion of a level of the hierarchy. While these decisions seem to be best made by an expert, we are interested in identifying the most likely or sensible changes. We will discuss a hierarchy that has at least three-levels. In such a hierarchy, it is conceivable that changes to one level may need to propagate up, down, or both up and down in the hierarchy. We will refer to the levels as *abstract*, *middle*, and *specific* for simplicity. The classification problems we will wish to accommodate in our model refinement approach to learning are:

1. False-positive at *abstract*. In this case we are identifying that we may wish to restrict a part of the model that is parent to more specific parts. Essentially, we are stating that the most abstract model specification is too abstract. The question is, do we wish to propagate this "restriction" downwards in our hierarchy, and if so, how far down? Certainly a restriction may have to be reflected in the "hierarchical glue" mapping, even if a restriction is not made at the more specific level. It is possible that a restriction at an "upper" level may render the level change to the subsequent level of little or no use, and the levels may most sensibly be merged.

2. False-negative at *abstract*. The most abstract model specification has failed to accommodate an expert identified instance that should succeed. The abstract model is supposed to encompass the "key indicators" of the model, and the failure to accommodate a positive example indicates that the abstract model must be relaxed further. This further relaxation could potentially be propagated downwards in the hierarchy as other relaxations, or even suggest that a new level in the hierarchy be created above the current level.

3. False-positive at *middle*. As was the case for *abstract*, relaxations may need to propagate downward causing one or more relaxed lower levels. Additionally, a relaxation may indicate that a key-feature of the model that exists at a more abstract level should also be relaxed. In this case, changes would propagate upward via the hierarchical mapping function.

4. False-negative at *middle.* Once again, as was the case for *abstract*, restrictions may need to propagate downward causing one or more restricted lower levels. In addition, a restriction may indicate key-feature of the model that exists at a more abstract level should also be restricted. In this case, changes would propagate upward via the hierarchical mapping function.

5. False-positive at *specific.* There will be no relaxations pass downwards, however, they potentially could pass upwards one or more levels.

6. False-negative at *specific.* There will be no restrictions pass downwards, however, they potentially could pass upwards one or more levels.

Hard and fast rules are not immediately clear regarding how these hierarchies should refine in each situation. It is obvious that in absence of a better understanding about how the "space" of example instances by the model is affecting by changing the model in one direction or other, we must rely primarily on the expert for determining the degree of inter-level change that is supported for a particular model change. However, it would seem that there are two primary cases for inter-level refinement:

1. False-Negative adjustment to a level, implying in-level space expansion

   (a) Upward propagation of restriction, implying that the upper level(s) are similarly (or partially similarly) restricted. Essentially if the more abstract level is functioning as expected, this likely is not necessary, and a misclassification of the specific does *not* seem to imply a necessary misclassification of the general, particularly if the misclassification was noted by the expert during a top-down search.

   (b) Downward propagation of restriction, implying the lower level(s) are similarly restricted. Essentially is would seem to make sense that a restriction of the abstract should restrict the specific in the case that a relevant mapping exists from abstract to specific for the adjust problem aspect or constraint affected. As an example, consider modifying an abstract such as chair to include only those things with four legs, changed from a previous description of three. If the abstract now has four, subsequent more specific versions of this concept such as office-chair should have four also.

2. False-Positive adjustment to a level, implying in-level space shrinkage

   (a) Upward propagation of relaxation, implying that upper level(s) are similarly (or partially similarly) relaxed. Once again, it would appear that if the abstract level is functioning correctly the expert would not wish to reflect these specific changes at that abstract level.

   (b) Downward propagation, implying that lower level(s) are similarly relaxed. Once more, a change at the abstract would seem to imply an appropriate change at the specific if applicable.

The determination of revision in a hierarchy is certainly an open question, and the development of some experimental evidence and further investigation into representation change of a concept which is described at several connected levels of abstraction simultaneously is certainly warranted.

1.5 Summary

1.5.1 Why is this work learning?

We are faced in this work with a problem of refining an expert-defined model based upon subsequent examples that fail to fit the current model. Learning new versions of the model when past examples incorporated into the model either no longer are available for direct evaluation, or when these past examples are already represented in an expert's mental model differs dramatically from learning from a set pool of examples, as is the case with Probably Approximately Correct (PAC) learning theory as described in Anthony & Biggs (1992). What is important in this work is refinement that accommodates a given counter-example, while perturbing the existing concept definition minimally, or if possible, perturbing the existing concept definition exactly enough to accommodate the new "class" of problems represented by the given counter-example. In the ideal case, a counter-example identified a single constraint error in some template and a single relaxation or restriction of this constraint is identified and made in the template model. In less ideal cases, we have to somehow select one of many possible relaxations or restrictions that will accommodate the new information. At this point, expert interaction in the process seems advisable if additional domain information or examples are unavailable. We have attempted to outline the major issues and problems in developing an interactive system for refining both single-level and hierarchical template models of spatial concepts.

The general conception of a problem as a graph of constraints and the associated solution set of this graph within a particular situation or domain value set is very powerful. Learning changes to a single-level constraint graph representation from single examples is problematic, especially when considering coverage of previously accommodated examples. Determination of problem aspects for relaxation or restriction is heavily domain dependent, and the specification of explicit candidate problem aspects such as slot removal, specific constraint removal or adjustment and degree of restriction/relaxation needs to be determined by an expert either prior to use of any refinement method, or during refinement. The *addition* of problem aspects such as slots or constraints would appear to be a very difficult problem without many examples present, however, the possible application of a "meta-learning" approach to determining additive components would appear to be an interesting area of future research.

Certainly the definition of a concept approximation such as a spatial template is often intended to "outlive" a particular expert or set of experts, and essentially serve as a definition of the concept to future users or experts. Consequently, model refinements may also be viewed as a transformation from an approximation of a concept held of one user set to an approximation of a (possibly different) concept held by another user set. Similarly, if one spatial template represents a concept A, then if another (known to be similar but slightly different) concept B has no currently known representation, then accepting some spatial template representing A as an approximation of B and then modifying it by example to be closer to an approximation of B seems a sensible approach when specific information about concept B is unavailable.

1.5.2 Additional Related work

As mentioned, the before-revision template is an approximation of a concept based on experience and previously encountered examples; essentially it is a form of memory of both of these. Wasserman (1989), describes the problem of revising a "memory-based" model based on a single example as the *stability-plasticity dilemma.* The dilemma essentially is whether or not a model can remain plastic enough to incorporate new examples as they arrive and yet retain stability needed to ensure that previous example adjustments are not lost. Incorporation of a new pattern by changing a model may result in the model misclassifying examples it classified correctly previously, and essentially get no closer in approximating the ideal concept. Wasserman (1989) points out that in some domains, neural networks have been shown in such cases of temporal instability to never converge to the desired concept. As a solution, Wasserman (1989) describes Adaptive Resonance Theory (ART) as a way of "distributing" the cumulative or past-experience "memory" so that subsequent examples have only "local" effect and thus do not in a single stroke replace previous knowledge. In a sense the cumulative existing model is preferred of immediacy of a new example. This same type of behavior is essentially reflected in the "minimal" accommodation of templates to the new examples that has been discussed earlier in this work.

A problem with many similarities to learning constraint graphs based on new information is reported in Lam & Bacchus (1993*a*) and Lam & Bacchus (1993*b*). In these works, Bayesian Networks are learned based on the "Minimal Description Length (MDL)" principle. The MDL principle is based on the idea that the best model of a collection of data items is the model that minimizes the sum of:

1. the length of the encoding of the model, and

2. the length of the encoding of the data *given the model*,

both of which can be measured in bits. A good example of this principle is given in Lam & Bacchus (1993*a*), for polynomial learning based on n points in the plane. Essentially, to find a function or model that precisely fits these points, $n + 1$ numbers are required representing the coefficients of the polynomial, and these represent the (first item) encoding of the model. Additionally, the store the data given this polynomial (the second item), n x-coordinates are required so that the n y coordinates could be computed from the model. Consequently the sum of the description lengths would be $2n + 1$ times the number of bits required to store the numbers for some precision. Now, another model might use a lower order polynomial, say of some order $k < n$, and thus only $k + 1$ numbers are required to store the coordinates (first item). Now, the data points can be specified by specifying the x-coordinates, however, this model will not fit the data precisely, and there is going to be some error e_i between the y_i value computed for each x_i and the actual y-coordinate of the i-th data point, y_i. To encode the data points, these error factors would need to be stored along with the x-coordinates (item 2). Clearly it is possible that there may be some polynomial of degree $k < n$ that yields the minimum description length.

Essentially the MDL conception is an alternative to our definition of "minimal" as the least possible number of restrictions or relaxations in the PCSP space. Since we do not have the previous examples stored explicitly in the model as specified, it is difficult to establish how to calculate the MDL for template revision. Lam & Bacchus (1993*a*) defines the two item calculation of MDL for Bayesian Networks, and thus prefers models in a search space of possible learned models accordingly. An interesting future work would be to attempt to define the MDL in terms of constraint graphs.

In Lam & Bacchus (1993*c*), the earlier work is extended to account for refining an existing network based on new data (or examples). The focus in this work is on taking advantage of local revision where possible (since new data may often be localized in the existing Bayesian Network). This work is reflective of the work presented here in that they attempt to preserve as much as possible of the original network as is consistent with the new data. Ideally this is exactly what we would like for template revision. The learned Bayesian structure is desired to be be as simple as possible, to be similar to the original structure, and to accurately represent the distribution of the new data. In the paradigm of this paper, "as simple as possible" reflects into the abstract representation in some sense - perhaps the idea of not adding new constraints, but rather modifying existing ones parallels this concept. The aspect of "similar to the original structure" maps directly to our concept of least relaxed or restricted in the PCSP space, and "accurately represent ..." covers our requirement that the new example be encompassed by the new model. What is lacking in our work to make this kind of mathematical evaluation of a "minimal" change to a model is evaluation functions for change, and a method of representing how individual previous examples are reflected in a new model.

1.5.3 Conclusion

I have examined here the revision of spatial template represented as constraint graphs as an attempt to learning the "ideal" spatial concept held by one or more domain expert(s). Further, I have attempted to outline the issues in attempting to incorporate these revisions into a more complex hierarchical template model. While this is certainly a work in progress, it is clear that learning revisions to hierarchical models is both important and difficult, particularly when the exact context of examples is not immediately apparent. The presence of an expert can effectively guide the learning process through these "holes", however, the issue of what general and domain knowledge needs to be modeled in order to automatically make these types of decisions is an open question. In essence, we are asking the very hard question "What do we need to know about the domain, knowledge representation, spatial problems, and learning itself to be able to successfully refine our model?". In the absence of all of the pieces of this puzzle, this work is directed towards obtaining an interactive, but intelligently-guided system that supports spatial template model revision and instance identification.

2 Graph Isomorphism : Relationship to the Spatial Template Recognition problem

This work describes the work of Lubiw (1981) with reference to subsequent literature from the perspective of investigating Spatial Template Recognition.

Spatial Template Recognition (STR) is an interesting problem with application to many different domains, including intelligence analysis and "high level" problems in machine vision. Essentially, a human concept with many spatial aspects and a high degree of variability is represented according to some scheme, and instances of this concept are located in a possibly "dirty" or "noisy" situation. A natural way of modeling STR is as a *Constraint Satisfaction Problem* (CSP). While CSP in general is NP-Complete, in practice certain characteristics of spatial problems have been exploited to make realistic problems solvable in reasonable amounts of time. Of concern is whether these results will "scale up" to larger templates and situations.

Graph Isomorphism (GI) is a well-studied, related problem that has yet to be shown to be a member of the class NP-Complete. It represents a class of problems known as "GI-Complete", hypothesized to be "simpler" on the basis of certain problem characteristics. In this work these characteristics are discussed with reference to similar aspects of STR, and several related problems are examined with emphasis on problem complexity. Of particular interest is the fact that GI has been shown to have both "restricted" and "relaxed" versions that are NP-Complete. A version of restricted STR based on "spatial locality restrictions" has been demonstrated to be particularly efficient, and it seems possible that this restricted problem may have a complexity related to restricted GI. This paper introduces some of the main problem aspects of GI and some other closely related problems and serves as a basis for future more detailed analysis of STR complexity.

2.1 Problem Introduction

Static **Spatial Template Recognition in Situation** (STR) which has been an active area of interest for practical application across various domains is defined in Section 2.2. Section 2.3 outlines an approach to solving of STR with a complete search strategy. Various formulations of the **Graph Isomorphism** (GI) problem are described in Sections 2.4.1,2.4.2, and analogies between STR and these formulations are shown in Section 2.4.4. In Section 2.4 the results shown in Lubiw (1981) are outlined, and several restrictions and generalizations of GI subsequent to Lubiw (1981) are discussed. Lubiw (1981) observes that GI may be representative of a class of problems know as GI-Complete that is in NP[4], but neither in P

[4] P represents the class of problems that are deterministically solvable in time polynomially related to the problem input. NP represents the class of problems solvable in non-deterministic polynomial time. Two problems are polynomially related if an encoding of either can be transformed into an encoding of the other in polynomial time. NP-Complete represents a class of problems in NP with the property that if any one NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time

nor in NP-Complete. In addition, Lubiw (1981) and Booth & Colbourn (1977) describe a breadth of problems polynomially equivalent to GI, and Lubiw (1981) attempts to explore the boundaries between problems known to be NP-Complete (such as **Subgraph Isomorphism** (SGI)(Garey & Johnson 1979)) and closely related problems that appear to be members of GI-Complete. In Section 2.4.5 the related work of Andrew Wong in pattern matching is briefly related to STR, and this paper's main conclusions are summarized in Section 2.5, including some conjectures about STR's complexity.

2.2 The Spatial Template Recognition Problem

2.2.1 What is STR and how complex a task is it?

STR is the problem of finding an instance of some predefined configuration of objects in a given situation, or set of object instances. An example of a non-typically rigid spatial template in a two-dimensional domain would be a "map" of a typical molecule that describes the molecule in terms of the relative positions, types and orientations of the various atoms. An example of a rather discrete, exact situation might be a two-dimensional sample containing many instances of different types of molecules discernible only by individual sub-parts. The question formed by STR for this problem instance would be: "Where are the instances of this molecular template in this situation?". In this example both the template and the elements of the situation have relatively "rigid" design - the molecular model has little variability from instance to instance, and the situation has potentially little ambiguity with respect to both identity and position of situational elements.

A more complex domain for STR is intelligence analysis. An intelligence analyst must draw on specific experiential and learned knowledge of enemy behaviour to recognize specific deployment patterns signifying organizational relationships of sighted enemy elements. One aspect of this complex task is recognizing the existence of enemy "doctrinal templates" which detail the patterns in which military units are commonly deployed for specific activities. These "templates" represent a snapshot view of how the elements of a military grouping are commonly positioned. Since military units are often strictly structured and deployed according to fixed procedures, a fairly accurate "typical" spatial representation can be built of particular formations undertaking certain activities in the field, such as "Motor Rifle Regiment (MRR) in attack". The analyst must fuse spatially and temporally disjoint messages or sightings into a coherent picture of the entire situation, including higher level formations and their activity. The individual elements of the MRR spatial template may be envisioned as "slots" in the template, and the rules or restriction regarding deployment of these elements can be represented as constraints amongst the template slots. The current set of sightings is referred to as the "situation", and each sighting as a "situation element".

One major difficulty with such a task is that these templates are embedded within an

solution, that is, P = NP (Cormen, Leiserson & Rivest 1992).

extremely complex, dynamic and incomplete "view" of the enemy[5]. Intelligence messages from the field may contain vague or conflicting composition or location information, and inaccurate messages may have been modified by experts into inaccurate conclusions. Also, any template is simply an approximation of some expert-held "ideal" concept, related work on improving these approximations with experience may be found in Section 1. Templates defined in terms of spatial relationships between components of specific types, sizes, activities and orientation approximate reality by allowing certain tolerances on the attributes or spatial constraints. Considering a template as a set of "stick-pins" of varying types and sizes interconnected by a set of rubber-bands with a minimum and maximum stretch capability, the process of template matching reduces to finding appropriate situation elements in which the stick pins may fit and where the bands between the pins allow this fit.

Matching of these templates to incompletely specified situational views can result in many partial solutions where only some subset of the pins could be fit to a situation element. In these cases which of many partial solutions is more promising may possibly be determined according to predefined criteria or measures. In addition, the knowledge of the precise structure of these templates themselves is only approximate, and the templates themselves may evolve over time. Consequently, any attempt to perform the recognition task algorithmically will find approximate solutions only, however, these may form a base-point for expert utilization.

While it is important to understand how the STR problem in reality is rife with complexity, for the purposes of this paper, I will assume a simpler STR in which I am interested only in finding a set of all matches of a template in a situation. Template flexibility is modeled using range constraints between slots, and situational elements inaccuracy modeled with locations that have varying degrees of specificity.

2.2.2 STR Representation

A relatively obvious formulation of STR as a **Constraint Satisfaction Problem** (CSP) was proposed in Woods (1993*b*) and Woods (1993*a*). I initially define STR as follows: A situation consists of a set of elements each with a set of characteristics (such as color, type, etc.). A template consists of a set of variables, each possibly limited by a set of range or specific characteristic values. Additionally, each variable is possibly connected to any other variable (or possibly set of variables) by a constraint-arc or set of arcs which impose relative characteristics such as relative position, orientation, or possibly which insists upon pairwise variable characteristic assignments. A template instance is "found" in a situation if a 1:1 mapping exists from situation elements to variables such that no situation element maps to more than one variable, and based on this assignment, all constraints represented by constraint-arcs are satisfied. A decision problem based on this definition could be expressed

[5] Other related work in pattern analysis such as Stacey & Wong (1988), Ghahraman, Wong & Tung (1980) and Wong, You & Chan (1990) models problems with a far greater "rigidity" in terms of the spatial template model.

as "Given a Template T = (set of Variables V + set of Variable-constraints VC + Arc-constraints AC), and a Situation S = (set of Situation elements), is there an instance of T in S ?". Problems specified in this fashion, where a solution is composed of a set of consistent element-to-slot assignments are well described in (Kumar 1992). An example of a template constraint graph is shown in Figure 8 with five node slots where each slot is constrained by several attribute ranges, and 11 inter-slot spatial constraints on distance, position, and orientation.

I am interested in this paper in describing the complexity of locating all possible instances of a template in a given situation. Consider a template with n slots (each possibly related to another by one or more constraints), and a situation with some number m of situation objects. We need to find all of the possible mappings from situation objects to template slots such that the "template matches" (i.e. all of the constraints are satisfied). Essentially we are trying to solve a restricted version of general CSP[6].

A "perfect" or total solution to a STR problem is an assignment from the situation elements to the template slots. There are n template slots to be filled with from the pool of m situation objects, with the ordering of the selection corresponding to an assignment to the slots. There are m choose n candidate solutions for our set, that is, the number of n-combinations of an m-set:

$$NumberCandidateSolutions = m!/n!(n-m)! \tag{1}$$

Thus there are some number of solutions polynomial with degree of candidate solutions n, with a base of the size of the number of situation elements, m. Note that typically the number of situation elements m far dominates the number of template slots n, however, real-world templates have some substantial (¿10) number of slots. Naively generating these solution combinations is not going to prove to be a good solution! However, in the next section several spatial heuristics will be discussed that seem to greatly localize the problem.

2.3 Templates and Search : Complexity

We are interested in finding all possible instances of a particular template in a set of situation elements, and consequently in finding all solutions to a CSP modeling our template recognition problem. In the CSP literature, these problems are typically solved via some search strategy coupled with domain-dependent heuristics, via some graph-reduction method which attempts to propagate consistency throughout the problem graph, or through some hybrid approach utilizing both search and propagation. It has been shown that search strategies with "look-ahead" properties can be thought of as augmenting search with some degree of during-search constraint propagation. By defining search strategies in this way the amount of constraint propagation adopted during search can be more easily identified and adapted,

[6] General CSP is known to be NP-Complete: consider a proof by restriction to Graph 3-Coloring for instance.

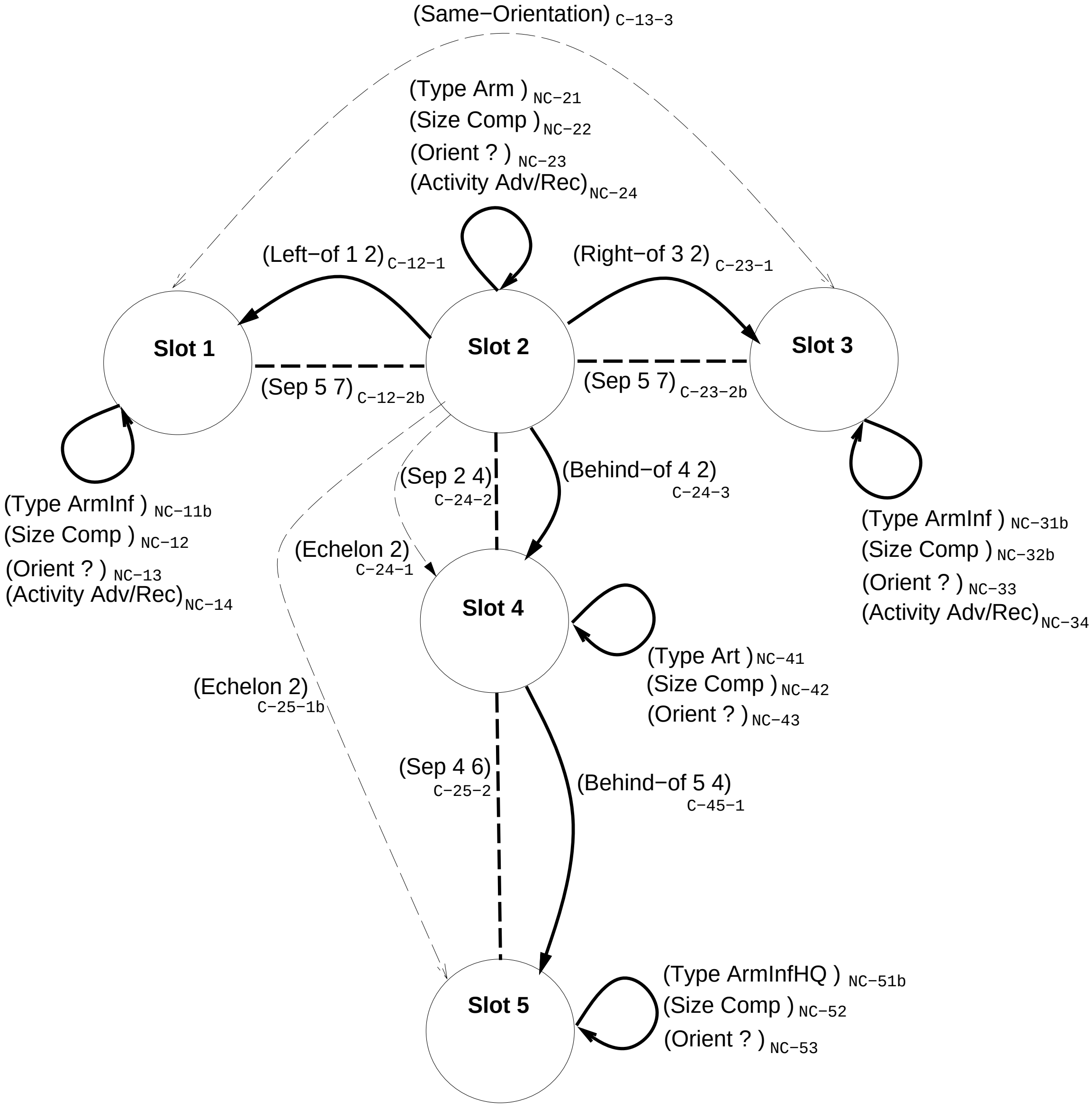

Figure 8: Whole CSP 5 variables, 11 arc constraints.

and various strategies more easily compared empirically. Although Nadel (1989) indicates which type of hybrid strategy is best for some small toy domains, published results solving interesting problems using this strategy are difficult to find. The spatial template recognition problem provides an excellent example of a complex problem which can be addressed in a reasonable length of time utilizing hybrid search.

The hybrid search based approach was demonstrated to be useful even for large situation datasets as a result of effective use of specific domain heuristics such as spatial locality and general domain independent heuristics such as applying constraint propagation during search, or abstracting the CSP. The question is, how did these properties overcome the computational complexity of inherent to the problem ? Related problems are discussed later in this paper.

If we view the space of candidate solutions to an STR as being determined by the template representation, then we must ask what happened to our problem complexity as the problem is either relaxed or restricted. Section 2.3.1 and 2.3.2 discuss some relaxations and restrictions possible in the STR domain.

2.3.1 Relaxing STR as CSP via Partial Solutions

In the automated portion of the approach defined at length in Woods (1993*b*) and Woods (1993*a*), search is used to satisfy as many constraints in the original problem as possible. If no solutions are found for the original template specification, it is possible to "relax" some of the constraints in a systematic way so as to find instances of some "least relaxed" version of the original template. These "relaxed" templates may be thought of as abstracted views of the original spatial concept. This relaxation essentially increases the size of the potential solution set. In the absence of other factors, generating all relaxed template instances in a situation is more difficult since there are more to generate.

In other work such as that of Lapointe & Proulx (1994), a different approach is used in which the fuzzy *distance* between a perceived instance and a template is measured and is used to determine a "match" between situation and template. Once again, both domain heuristics and domain independent heuristics were used to obtain an efficient solution. In this approach, the spatial template acts less "precisely" as a direct determiner of the solution space and more like a "classifier" of the degree of fit of a particular mapping. In any event, this is once again a relaxation, except that the resulting solution set is ordered precisely by some possibilistic distance from an "ideal" represented in the template.

Both of these works make some claim to being "good enough" in the absence of a perfect solution, however, a better understanding of the conditions under which these "good enough" solutions may be obtained would be of great value in evaluating possible real applications of these algorithms.

2.3.2 Restricting STR as CSP via Spatial Locality

Spatial locality is an example of an important "simplification" or restriction of the general CSP nature of STR in our formulation. This heuristic is applicable to CSP problems that are grounded in spatial coordinates. Essentially, we take advantage of the fact that our spatial templates may be "precompiled" such that it is possible to encode rough boundary information for an entire solution based upon only partial variable assignment information. For instance, if a template has no two slots farther apart than some measure *max*, then during search if an assumption is made about the location of some template slot *t*, then we may conclude that no subsequent template slot can be assigned a situation element farther than *max* from the slot *t* assignment. Given that we may easily determine the distance between situation objects (perhaps through a pre-built index or similar method), it seems easy to dramatically reduce overall search complexity by pruning large numbers of situation elements as candidates for slot assignment "locally".

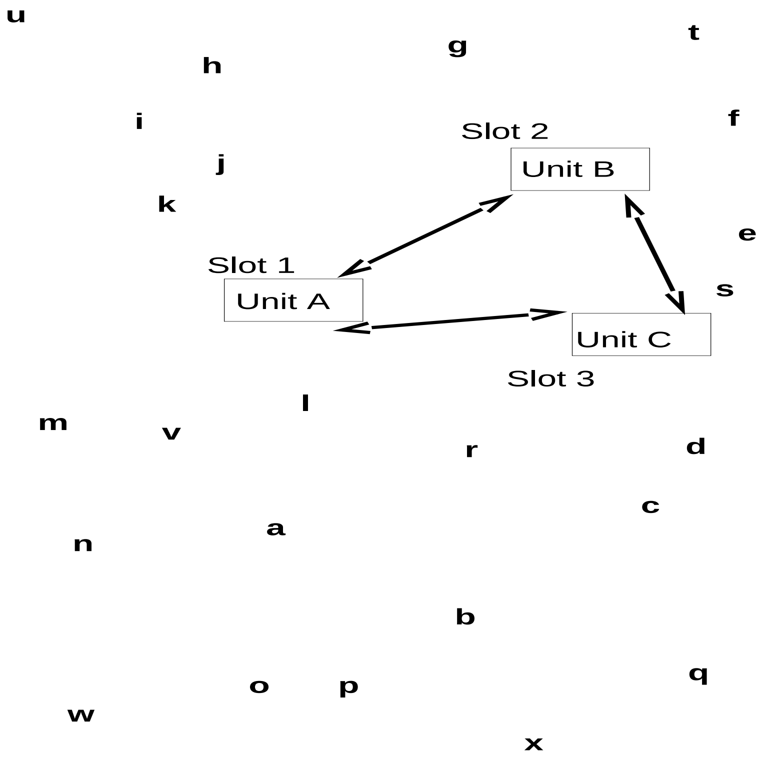

Figure 9: Situation with partial solution overdrawn.

Once assignments have been made to some variable (or set of variables), a boundary is created limiting the scope of this particular solution. Now, when subsequent variables and constraints are added making the problem more specific, the scope of the search is restricted according to the outlined area for each problem. Thus the earlier partial solution

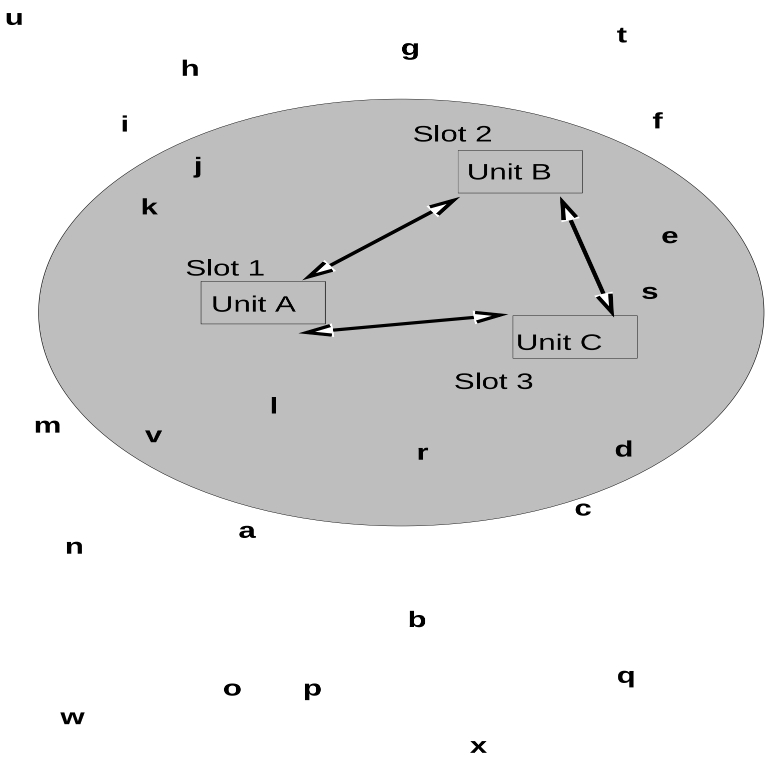

Figure 10: Situation with spatial pruning based on partial solution.

has not only been used as a basis for limiting later constraint checking, but also in quickly eliminating some domain values for each of the new-level variable additions. In our example, Figure 9 shows a partial solution where some template slots 1, 2 and 3 are matched to situation elements Units A, B and C respectively. Before search continues, we apply our spatial locality heuristic derived from A, B and C assignment positions, limiting the area of search for the remaining slots as shown in the shaded area of Figure 10. Consequently the only candidate instances in the selected area for the next level slots are *j, k, v, l, r, d s and e* , while the others outside this area are excluded.

Other restrictions are certainly possible, such as one mirroring the discussed relaxation in Section 2.3.1 where constraints might be tightened thus limiting the number of candidate solutions. In Section 1 the use of abstraction as a relaxation and specification as a restriction is discussed further.

2.4 Isomorphism Formulations

Given two graphs we may wish to find a correspondence between the graphs according to certain restrictions. For instance, Graph Isomorphism (GI) might be used to define the structural equivalence between two graphs, Subgraph Isomorphism (SGI) might be used to define the structural equivalence of part of one graph with another graph, and Graph Automorphism (GA) might be used to define a certain type of useful structural symmetry in a graph. GI has never been shown to be NP-Complete or a member of P, and GI along with problems that are polynomial-time reducible to GI are shown as belonging to the space GI-Complete. Figure 11 on page 36 outlines some hypothetical complexity spaces in NP. Lubiw (1981) points out that earlier work has shown that if, in fact, P $\neq$ NP then there exist problems in NP which are neither in P nor NP-Complete, and this fact could possibly explain the lack of a classification of GI in P or NP, if GI belonged to such a class[7]. GI-Complete problems are shown to be "harder" than P since no GI-Complete problem has been shown to be in P, and "easier" than NP-Complete for a similar reason.

We are interested in how STR with restrictions or STR with relaxations corresponds to these problems. Certainly generalized STR as CSP is NP-Complete. But what of "STR-spatial-restrict" or "STR-partial" ?

Figure 12 details four hypothetical complexity spaces in NP: P, Graph Automorphism Complete (GA-Complete) intersect NP, Graph Isomorphism Complete (GI-Complete) intersect NP, and NP-Complete. I will outline each of these problems below and discuss the key aspects of each that places it in a class other than P or NP-Complete by discussing the boundaries or at least the changes that shift a problem across these boundaries[8]. In this figure, we have labelled Graph Automorphism "GA", and those problems which are polynomially reducible to GA are shown as belonging to the class GA-Complete. SGI represents Subgraph Isomorphism, while GI-bipartite, GI-regular, GI-line, GI-tree, GI-planar, and GI-interval all represent restrictions of GI where the graphs belong to the graph type specified.

2.4.1 Graph Isomorphism (GI)

The *Graph Isomorphism* (GI) problem is essentially to determine whether two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are isomorphic, that is, is there a mapping function (bijection) f between G and H such that (u, v) is an edge in G if and only if $(f(u), f(v)$ is an edge in H.

Graph Isomorphism has been observed (Lubiw 1981, Hoffmann 1982) to possess certain structural qualities[9] that keep it out of NP-Complete in general. Further, the destruction of

[7] It should also be noted that if it is the case that problems are proved to be between P and NP-Complete such as GI-Complete implies, then there will also be problems in NP strictly between P and GI-Complete, and we have labelled these (or at least some of these) with GA-Complete in Figure 12 on page 37. In addition, Hoffmann (1982) examines some generalizations of GI which appear to be harder than GI and yet are not known to be NP-complete, implying the further existence of a class between GI-Complete and NP-Complete. This is labelled on our Figure 12 as "Hoffman Space".

[8] It is important to note that these *boundaries* are not known to exist, only that they may exist. Essentially then, the four classes obey the following relationship: $P <= GA - Complete <= GI - Complete <= NP - Complete$, and the inequalities could possibly be strict.

[9] Lubiw (1981) and Hoffmann (1982) point out that this structural quality is difficult to determine,

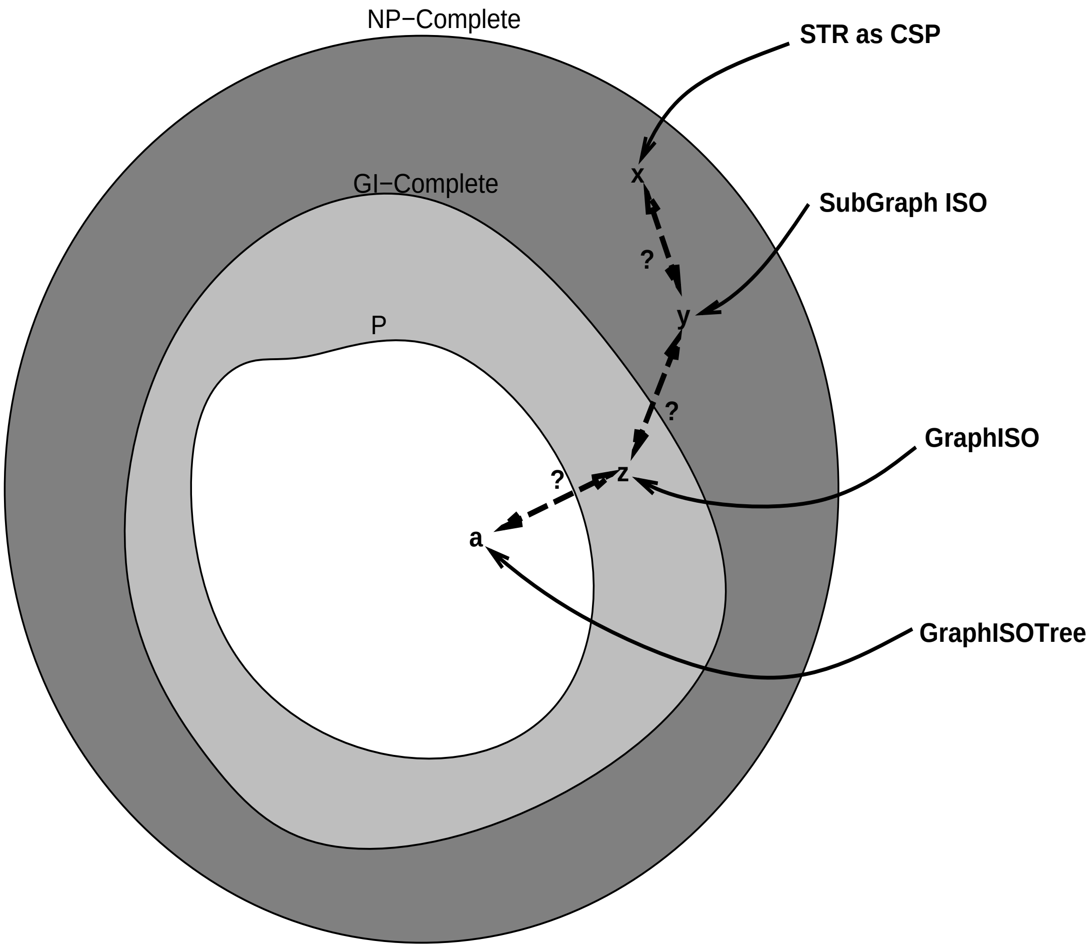

Figure 11: Some hypothetical complexity spaces in NP.

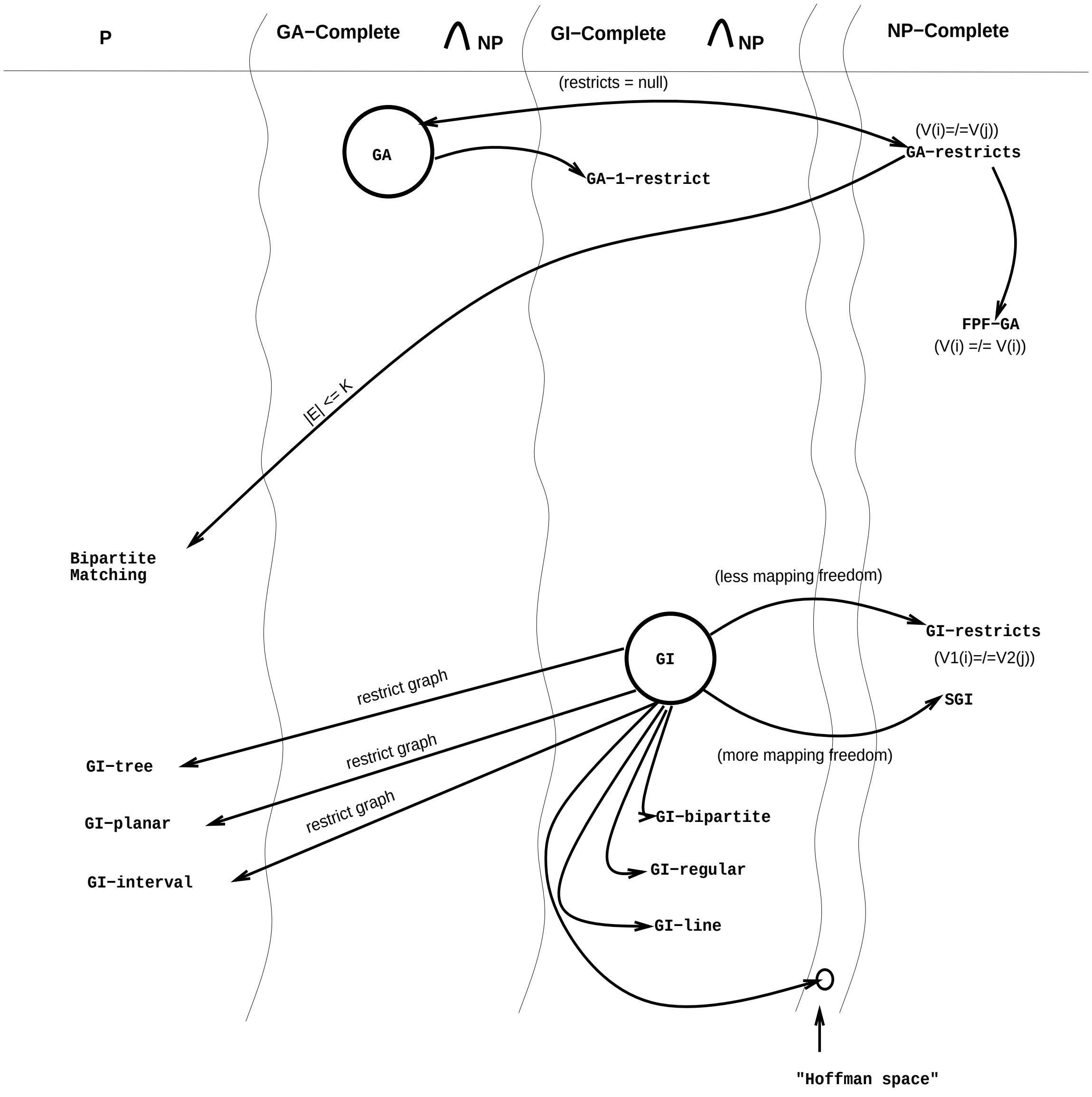

Figure 12: Some complexity spaces and problems possibly not NP-Complete.

these structural qualities through either certain types of relaxation or restriction results in a problem in NP-Complete. For instance, a generalization of GI to Subgraph Isomorphism (SGI) results in a destruction of GI's "structure', more freedom in mapping the initial graph to the target graph, and an NP-Complete problem. The structure "destroyed" in this case is related to the previously described equivalence relation between graph vertices - essentially the question of which vertices to map to one another is further complicated by the selection of which subgraph to attempt the mapping on.

Interestingly, GI can be made NP-Complete with either restriction and less mapping freedom amongst the graphs as in the case with GI with general restrictions allowed on the possible mappings between the two graphs, or with relaxation and more mapping freedom as in the case with SGI.

GI is clearly related to STR according to the search for an equivalence relation, but seems simpler in a sense. Specifically the fact that the graphs in a GI success are of the same "size" with respect to vertices and edges seems counter to the conception of a relatively small template in a large situation. SGI would seem to fit this conception better. Further, in a complete identification of template instances we are interested in finding *all* equivalence relations, not just determining the existence of one. It would appear that STR corresponds to the counting version of this problem. However, as we have seen, the counting version of problems in GI-Complete are GI-Complete themselves! This result is quite different from NP-Complete problems where the "counting" complexity is more than polynomially worse than NP-Complete[10]. We would certainly prefer to locate STR in some less onerous class. Unfortunately, the consideration of mapping freedom with GI-restricts seems to also parallel STR. Perhaps some form of isomorphism such as Subgraph Isomorphism with restrictions fits best, however, this problem would appear to be NP-Complete also.

Several restrictions of GI are known to be in P. Specific restriction of the structures of the graphs involved such as planar or tree have known polynomial solutions. This feature could be an important starting point for future template specification or refinement efforts[11].

but observes the following: The counting versions of all known NP-Complete problems belong to a class apparently not polynomial time equivalent to NP-Complete. However, the Automorphisms of a graph G induce and equivalence relation $==$ on the vertices of the graph. Specifically, two vertices, a and b obey $a == b$ if there is an Automorphism A of G such that $A(u) = v$. The set of equivalence classes of this relation $==$ makes up an automorphism partition of the graph, and finding this partition is known to be GI-Complete. With this fact it can be shown that the problem of counting the number of isomorphisms between two graphs is GI-Complete. Thus there are essentially two evidences that GI-Complete is not equivalent to NP-Complete: first, no know GI-Complete problem is known to belong to NP-Complete (despite much interest in the problem), and second, known NP-Complete problems have counting versions that seem to be significantly more complex than the counting versions of GI-Complete problems.

[10] The complexity of counting versions of NP-Complete problems is referred to as #P-Complete.

[11] Essentially, if it was known that a graph that was more "tree-like" in a sense was easier to locate an equivalence relation for than a more general graph, then preference could be made for templates of this kind. Further, the "triggers" or cues to failure used in the polynomial time GI-tree algorithm could potentially suggest similar strategies for "tree-like" templates. Other work in pattern recognition (both parallel and serial) has focused on these critical "features" or sets of features as points of discrimination in template

2.4.2 Subgraph Isomorphism (SGI)

SGI is a generalization of GI. Specifically, we are interested in asking not if two graphs are isomorphic, but rather if one graph possess a subgraph isomorphic to the other. SGI is known to be NP-Complete. As we have already stated, SGI would appear to model STR rather well if the question about the way in which constraints are mapped to arcs is ignored. If the situation reflects one graph and a template the other, then the graph represented by the situation possesses "virtual" arcs that are only evaluated during the actual equivalence determination, and in fact the "values" or even existence of these arcs is then dependent on the selection of the subgraph of the situation graph that is being matched to the template graph. For instance, if two slots "A" and "B" are joined with a distance arc, and slot "A" is mapped to a situation element "a", then the attempt to map slot "B" to some situation element "b" would only result in an equivalence (so-far) if the determined distance between "a" and "b" where "equal" to the distance arc between "A" and "B". If the arc represented some other attribute (such as type equivalence between "A" and "B") then the arc evaluation between "a" and "b" would be implicit until evaluated. The point is that we will have to consider graphs in STR that have multiple constraint arcs between nodes, and further, in solving STR, arcs will have to be evaluated dependent on the already completed element-to-slot assignments.

2.4.3 Graph Automorphism (GA)

Similar to Graph Isomorphism, we are searching for an equivalence relationship between a graph and itself. GA is not known to be in P, and in fact has never even been shown to be as hard as GI-Complete. Lubiw (1981) shows GA with 1 restriction[12] to be GI-Complete. GA with arbitrary restrictions is also shown in Lubiw (1981) to be NP-Complete. Additionally, a similar problem known as Fixed Point Free Automorphism (FPF-GA) where no vertex may be mapped to itself in the equivalence relation is also shown to be NP-Complete.

This problem is interesting with respect to STR based on the observation made in Section 2.4.1 that equivalence relations are induced by automorphisms of a graph G. Prabhu & Narsingh (1984) points out that the number of isomorphisms between 2 graphs is equal to the order to the automorphism group of either one. Thus, if there exists an isomorphism I between two graphs G and H, then for any automorphism A of G, I composed with A is also an isomorphism between G and H. Essentially a symmetry is identified in G as an automorphism. This symmetry could potentially be exploited in some way in future STR formulations, either on occurrence or intentionally in advance formulation of templates, since we know that GA is possibly "easier" to solve than GI.

instance discovery, and while our problem is considerably more general than that, perhaps analogies could be exploited.

[12] A restriction in GI or GA is essentially deemed to be a prevention of an equivalence being drawn between two vertices, one in each graph. For example a restriction might be of the form: "V_1 in V_H of graph H may not be mapped to V_6 in V_G of graph G".

2.4.4 Isomorphism Problems and STR

The STR problem is about locating structural equivalence, however, between a particular form of attributed graph and a situation rather than between two arbitrary graphs. If we consider that the situation objects in a situation form the nodes of some complete graph, where the arcs between each node represent either the spatial relationship of the nodes (such as distance for example), or some other relevant relationship that can be measured, then, in a decision about the existence of an instance of a spatial template in a situation, we are looking at an equivalence relation between graphs. The situation "graph" arcs may in fact not be "evaluated" until such time as a check is made for equivalence with a template arc.

We know that GI for restricted graph forms (such as tree and planar) is polynomial. Consequently the problem complexity is not immediately apparent if only one of the graphs is restricted to one of these classes[13]. In such a problem, we might restrict our search to sub-parts of a template that possess one of the properties, identify these parts and re-assemble these partial solutions into larger solutions in a divide and conquer type of approach. In Woods (1993*a*), the template problem is discussed in terms of two different problem decomposition approaches: abstraction, and partition. In the partition approach, a template is divided into "key" subproblems, such as the case of an airplane being divided into two broad parts "wing structure" and "body structure". Solutions are found for each part, and a cross product match is attempted with all possible combinations according to additional locality restrictions. Conceivably these structures could be formed utilizing templates of restricted form, thus simplifying each subproblem.

Lubiw (1981) and Hoffmann (1982) describe several *generalizations* of GI which allow more freedom in mapping the initial graph to the target graph and Lubiw (1981) describes several *restrictions* on the possible mappings between initial and target which can result in problems that are NP-Complete. Johnson (1982) adds other examples. This suggests that an inherent structure of the GI problem is being destroyed in either related problem case, and it is precisely this structure which makes GI easier than either generalized versions such as SGI, or restricted versions such as **Isomorphism with Restrictions** (GI-restricts). The GI structure was destroyed in the relaxation to SGI by the additional complexity of selecting a subgraph to match, and also in the restriction to GI-restricts (Johnson 1982). Lubiw (1981) indicates that other similar structural destructions occur in restricting GA to GA-restricts when the equivalence relation may no longer be reflexive, symmetric, or transitive. In FPF-GA only reflexivity is destroyed. In essence, in each of these cases the "isomorphism" is eroded enough that the *counting version* of the problem in question becomes #P-Complete.

The following are typical of STR relaxations:

- Relaxation of STR as CSP through CSP relaxation as defined in Freuder & Wallace (1992) by removing template variables or slots, relaxing constraints, or removing constraints, resulting in partial solutions.

[13] If a complexity result is known for disjoint-type Graph Isomorphism, I am not aware of it.

- Situational locational (or other attribute) expansion where the identified possible range of a situational element expands results in an expansion of the size of the candidate solution set[14].

The relaxations essentially seem to increase the problem complexity in general. Implicit in the solution approach of Woods (1993*a*), where abstraction is utilized in the problem representation and solution approach, is a type of *Upward Refinement Property* where a specific solution is known to have had an abstract solution, and in this way a set of abstract solutions identifies a space of specific solutions that contains all of the possible specific solutions. These specific "spaces" are much smaller than the entire space. It would not appear that "small doses" of these relaxations are going to benefit the solution of STR as we have formulated the problem. However, a completely relaxed constraint set would imply a trivial although huge solution set, and a very small constraint set would imply a only slightly smaller solution set.

The following are typical of STR restrictions:

- Restriction of CSP constraints resulting in a much smaller candidate set.

- Reduction in the locational (or other attribute) range in the situational elements resulting in a much smaller candidate set.

- Introducing the concept of spatial locality thus reducing the complexity of a problem after some "part" has been solved.

Spatial locality is essentially a precompilation of all constraints into a type of boundary condition for future search. It allows dramatic pruning of the search space and quick location of dead ends. It does not decrease the overall breadth of a search tree since it may only be applied to a local "partial" remaining subproblem, however, search depth may be dramatically reduced. In general, however, as the spatial "difference" between a template and a situation is seen to decrease, we see a decreasing effect. Interestingly, this size difference reduction corresponds to the case in the isomorphism problem where the larger and smaller graphs in the Subgraph Isomorphism problem approach one another and form the Graph Isomorphism problem which is likely *easier*. However, it is not obvious how to approach this problem aspect since it is not clear to me how to map a "spatial locality" heuristic to a general graph problem.

The work of Manning (1991) and Eades & Ng (1987) is particularly interesting to STR problems since they deal with visual graph representation problems that exhibit properties similar to STR in terms of spatial relationships. For example, in (Manning 1991) it is shown that detecting axial or rotational symmetry in a graph is NP-Complete. A particular "drawing" of a graph is shown that demonstrates various properties inherent in a graph (such as planarity, biconnectivity, symmetry, or diameter) while in another drawing these

[14] A candidate solution set is essentially the possible mappings exist from template to situation.

properties are not so “obvious”. Manning (1991) is concerned with drawing graphs that display these properties, and to this end one would like to find an efficient algorithm for detecting symmetries in abstract graphs. Similarly, since it is known that STR is solvable much more readily by an expert when distinctive characteristics such as symmetry are present in a template and situation, this NP-Complete result for symmetry detection suggests that restricting STR to certain subclasses such as those with symmetry may not be any easier in general, although an optimal algorithm for all symmetry detection for points and lines in two dimensions is presented in Eades & Ng (1987). However, it is not hard to imagine an application where pattern recognition techniques could be used as a “preprocessor” based on some critical template aspects to suggest some “obvious” starting points for the actual searching for equivalence relations.

2.4.5 Other Related Work

Stacey & Wong (1988) present an algorithm for determining the “largest common subgraph isomorphism in attributed graphs”. This work has derived from an attempt to make efficient algorithms for the pattern recognition problem which is essentially a special case of STR with exact, rigid templates. The importance of “largest subgraph” is a little uncertain (as the authors themselves point out), although it would appear that the commonality represents a kind of “best-fit” measure between situation graph and template graph. The algorithm presented is based on a depth-first exhaustive search that makes use of local contextual and structural information of the template representation graph. In this work they identify that investigating additional exploitation of attribute information (how about spatial locality ?) in search as a future enhancement. It is interesting to note that the authors observed in experiments that increasing their constraint tolerance (paralleling an STR relaxation) resulted in an expanded search space (as I predicted earlier), and no additional solutions (as would follow from situational and template rigidity and little symmetry). Additionally, the authors point out that symmetry in their situation yielded ambiguous solutions that were unresolvable without further attribute information, nicely demonstrating the effect of a graph automorphism creating additional isomorphisms.

2.5 Conclusions

What may we conjecture about the complexity of STR? We have made a little progress from our investigation of isomorphism problems, primarily that in general, the problem of identifying a single template instance would appear to be mappable to no better than Subgraph Isomorphism in general, and consequently would be NP-Complete for any instance, and #P-Complete for all instances. However, the ramifications of the spatial heuristic in some kind of average, typical or common case would be interesting to investigate in some future work, particularly in that since STR has immediate application potential we are far interested in cases that can possibly occur! The spatial heuristic is limited in its “effectiveness” by the difference in size between a template and a situation, where the closer the template gets in maximum extent to a situation, the less pruning may occur. It is interesting to note that in a similar graph isomorphic sense, it would seem that the closer two graphs become to one another in size the closer the problem is to GI-Complete (as the number of selections of subgraphs decreases), while the more they differ the closer the problem becomes to NP-Complete (as the number of selections of subgraphs increases). Towards efficient implementations of STR in specific domains, the properties of templates can undoubtedly be exploited - both with respect to typifying the common and unique features of templates and extending the spatial heuristic to be more accurate in specific cases, and in attempting to model templates that offer either an improved problem complexity due to their simplified structure, or allow for the structuring of intelligent heuristics based on partitioning templates into more simply structured portions in a divide and conquer strategy.

3 Applicability of *extended Logic Programming* for representation and solution of *Constraint Satisfaction Problems*

Logic programming languages such as Prolog provide a convenient way in which to represent relations such as variable constraints. Additionally, the nondeterministic nature of the unifiction method underlying Prolog supports implicitly the search required to solve Constraint Satisfaction Problems (CSPs). The standard backtracking control structure of Prolog and other logic programming languages results in very large search spaces which makes it inadequate for interesting problems. However, an extension of the Prolog control structure which includes consistency methods has been shown to embody search procedures that utilize forward or active application of constraints which reduce the search space *a priori* rather than *a posteriori* during search.

Standard "AI" approaches to solving CSPs have been rooted in a melding of search strategies with constraint propagation algorithms to form hybridized search strategies that make use of domain independent and domain dependent heuristics during search. Additionally, stronger, richer representation languages are useful in terms of both controlling search refinement and providing more accurate problem descriptions.

In this paper we show that while extended logic programming with consistency methods is far superior to standard logic programming for addressing CSPs, it is still too rigid and inflexible in both representation and control issues to be considered as a *sufficient* toolset for the CSP application developer. Essentially, the tools provided are tied too tightly together to allow sufficient flexibility to take advantage of specific problem characteristics to best effect, and what is really needed is a toolset consisting of a strong representation language, search and consistency tools that may be merged to any degree, and support for dynamic changing of control strategy throughout the CSP search space.

3.1 Introduction

A broad class of problems may be represented as sets of constraints, and may be solved via methods described in the literature as constraint satisfaction techniques. While in general the Constraint Satisfaction Problem (CSP) is NP-complete[15], certain domains lend themselves well enough to heuristic approaches to problem simplification so that solutions or adequate partial solutions may be located in a reasonable amount of time and effort (Nadel 1989, Yang & Fong 1992, Minton, Johnston, Philips & Laird 1992, Prosser 1993, Woods 1993*b*, Woods 1993*a*). Recent work such as Nadel (1990) and Tolba, Charpillet & Haton (1991) has emphasized how important problem representation, with respect to selection of constraints and domain "objects", can be in terms of efficiency of solution. Other work dating from the early work of Waltz (1975) and Mackworth (1977), and including Mackworth (1981),

[15] CSP NP-Completeness is demonstratable by restriction to Graph 3-Coloring for instance.

Mackworth (1992), Dechter & Dechter (1987), Dechter (1990), Dechter (1992), Mackworth & Freuder (1985), Freuder (1991), Freuder & Wallace (1992), Mackworth & Freuder (1993) and others has focused on generalizing local reduction or consistency achievement in constraint problems, and on typifying the effectiveness of such approaches.

The "AI" community has primarily addressed constraint satisfaction in terms of techniques for hybridizing consistency methods with search, representation and complexity issues, and developing new approaches to achieving varying consistency in networks of constraints. However, no universal "paradigm" or "model" has really emerged as a generic catch-all for the discussion and modelling of constraint satisfaction problems, other than the typical constraint-graph, perhaps. Individual AI implementation-based work has been based on local implementations of published algorithms, perhaps novellized or merged in new ways with other techniques such as particular search strategies. Several works (Prosser 1993, Woods 1993*b*, van Run 1994) have indicated that they are part of larger works which hope to provide means of constructing a generic toolset for stating and solving CSP in many domains.

In the Logic Programming (LP) community, much interest in the past 8-10 years has been focused on providing support in logic programming for formulating and solving problems based on constraints. Logic programming languages have been shown to be capable of providing one way of representing both constraints and nondeterminism and of "freeing" an application developer from the overhead of specifying specific search routines. Unfortunately this lack of flexibility in creating variant search strategies or in taking advantage of local constraint propagation in advance of search failure is potentially extremely limiting in terms of performance. Consequently, logic programming languages have emerged, such as Logic Programming extended with constraint consistency (CCLP) (Van Hentenryck 1989), and Constraint Logic Programming (CLP), which function as constraint solvers by providing implicit representation possibilities and non-determinism while also providing "built-in" facilities for local consistency application or constraint propagation and search control.

In the course of this paper we will be examining the constraint logic programming paradigm from the perspective of the new languages and determining whether they have become general and powerful enough to capture the representation and solving control requirements of a "typical" real-world problem known as *Spatial Template Recognition* (STR) formulated as a CSP (Woods 1993*b*, Woods 1993*a*).

We will attempt to summarize the aspects of "Constraint Satisfaction" that would appear to be sufficient for this domain in the new extended LP world, and which are in apparent need of extension. Reference for the purposes of this paper will be made specifically to the recent work of Van Hentenryck (1989) and Van Hentenryck, Saraswat & Deville (unknown) in outlining the original extended (CC) LP framework. Other extensions of this work in LP with parallelism (Montanari & Rossi n.d., Kasif & Delcher n.d.), hierarchies (Menezes, Barahona & Codognet n.d.), and spatial domains (Dube & Yap n.d.), are also of interest in specification and solution of the STR problem, however, fall outside the direct scope of this paper.

3.2 Spatial Template Recognition (STR)

Generally speaking, STR is the problem of finding an instance of some predefined configuration of objects that may be deployed amongst a given situation, or set of object instances. An example of a spatial template in a two-dimensional molecular domain would be a "map" of a typical molecule of some type that describes the molecule in terms of the relative positions, types and orientations of the various atoms of the molecule. An example of a situation might be some two-dimensional sample containing many instances of different types of molecules, and the question formed by STR would be: "Is there an instance of this molecule in this situation?".

3.3 Constraint Satisfaction: an AI clearbox approach

A constraint satisfaction problem representation consists of three major components: A set of variables, a range of domain values for each variable, and a set of constraints amongst the variables restraining individual assignments of domain values to a particular variable depending on domain assignments to other variables. A solution to a CSP is simply an assignment to each variable of a domain value from that variable's domain such that all constraints restricting assignments are satisfied. In some cases we may be interested in only a single satisfying assignment, while in other problem instances all possible satisfying assignments may be of interest.

3.3.1 STR as CSP

A relatively obvious formulation of STR as a **Constraint Satisfaction Problem** (CSP) was proposed in Woods (1993*a*). We initially define STR as follows: A situation consists of a set of elements each with a set of characteristics (such as color, type, etc.). A template consists of a set of variables, each possibly limited by a set of range or specific characteristic values. Additionally, each variable is possibly connected to any other variable (or possibly set of variables) by a constraint-arc or set of arcs which impose relative characteristic contraints on aspects such as position or orientation, or which possibly insist upon equality of a pair of variable characteristic values. A template instance is "found" in a situation if a 1:1 mapping exists from situation elements to variables such that no situation element maps to more than one variable, and based on this assignment, all constraints represented by constraint-arcs are satisfied. A decision problem based on this definition could be expressed as "Given a Template T = (set of Variables V + set of Variable-constraints VC + Arc-constraints AC), and a Situation S = (set of Situation elements), is there an instance of T in S?". Problems specified in this fashion, where a solution is composed of a set of consistent element-to-slot assignments are well described in Kumar (1992). An example of a template constraint graph is shown in Figure 8 on page 31. The graph has five node slots where each slot is constrained by several attribute ranges, and 11 inter-slot spatial constraints on distance, position, and orientation.

Of course in a "real" situation, this 1:1 mapping may be difficult or impossible to ascertain due a number of circumstances such as uncertainty or error in the perceived view of the world, or due to an inaccurate conception of of a template that contains either less or more elements or contraints than an actual template. If we still wish to solve the problem we must somehow fall back upon either some measurement of the "degree of fit/match" and estimate our confidence in this approximate solution, or perhaps abstract out the "most important" features of our template and attempt to locate this in a situation. In the approach defined in Woods (1993*b*) and Woods (1993*a*), abstraction through carefully defined constraint relaxation "levels" or template descriptions is combined with domain dependent and domain independent heuristics in a search strategy to offer a way of handling this uncertainty[16].

3.3.2 What generic CSP formulation provides

Formulation of a problem in a generic CSP format brings us immediately into the world of search, heuristics and approximate solutions, rather than minimalist or best solutions. A problem such as STR is formulated in the CSP framework described previously, and then some strategy must be utilized to locate a mapping of variables to domain values that satisfies some utility measure. For instance, in many problem formulations distinction is made between *hard* constraints that must not be violated in any candidate solution or mapping, and *soft* constraints that we would prefer not to violate but which we may relax if necessary. Of course this type of distinction can also be extended in some domains such as that of course scheduling discussed in Yang & Fong (1992), where one is faced with a wide range of constraint or preference rigidity on various characteristics of a solution, and where many of these are possibly contradictory. The acceptance of violations of some constraints in a solution is known as *constraint relaxation.* Complex strategies for relaxation have been examined, including the work of Freuder & Wallace (1992), and later of Yang & Fong (1992) and Woods (1993*b*) where abstraction is used as a model for the relaxation of constraints.

CSPs can be solved in several ways. One possibility is to "propagate" the domain restraints implied by finite variable domains and explicitly inter-variable constraint relations so as to iteratively reduce the size of variable domains until all remaining variable domain values are "consistent" with one another. These types of global constraint propagation algorithms have been explored at some length in the works of both Mackworth and Dechter. Another possible solution strategy (generate and test) is to generate all possible permutations of variable domain value combinations and test a complete assignment to see if the problem constraints are met. This is grossly inappropriate in even fairly simple problems for reasons of combinatorics. CSPs are typically solved with some kind of search. Basically a domain value is selected for each variable in some kind of turn such that the constraints are satisfied (or possibly satisfied to some degree), and then another variable is selected. If some

[16] In other work such as that of Lapointe & Proulx (1994), a different approach is used in which the fuzzy *distance* between a perceived instance and a template is measured and is used to determine a "match" between situation and template.

variable cannot be instantiated so as to satisfy the constraints, then backtracking is invoked, and a previous assignment is undone. A great deal of literature has surrounded search both in AI and other fields, and CSP provides an excellent environment for different search strategies and approaches or philosophies to flourish. Recent work has included that of: Woods (1993*c*) where application of constraints throughout search is shown to be formulable in such a way as to reduce the size of the search space subsequent to backtracking; Woods (1993*a*) in which STR is shown feasible across a variety of simplistic search algorithms and large domain ranges when making use of locality-based spatial heuristics; Prosser (1993), where the entire range of algorithms commonly applied to CSPs is evaluated and contrasted in varying circumstances; and van Run (1994), where the ordering and re-ordering of variables during search is discussed in detail.

In very many CSP domains, we are interested in a solution that is locally minimal with respect to some evaluation function. To this end, recent work (Minton et al. 1992, Yang & Fong 1992) has shown how effectively "good" solutions may be found for domains of substantial size, where finding the "best" solutions is exponential. These works have utilized a type of iterative-repair heuristic during search that minimizes the numbers of conflicts among constraints based upon an initial (possibly random) assignement of variables to domain values.

CSP solutions may be characterized by a variety of search algorithms, domain dependent heuristics that are used to control a particular search strategy, domain independent heuristics that essentially change the nature of one search strategy to another, possibly more CSP-specific strategy, and localized strategies that tend to focus on repairing a rough solution iteratively until it is "good-enough". Each of these areas are currently active research areas, and little agreement has been forthcoming to date, even about which CSP problem characteristics favour which approach. Essentially, solutions are created uniquely for a particular problem domain from this toolset, and generalizations are made where possible. One obvious problem with this type of approach is the intense repetition of implementation amongst researchers and application-builders. It is true that the algorithms and methods are frequently published in their entirety, and are discussed at some length in the literature, but algorithms must still be translated into code in the language desired, debugged, and application front ends need to be built for each use. Even in those rare cases where researchers make their code and documentation available, these provisions are rarely of a quality that could be said to be robust or standardized in any sense.

So then, CSP applications suffer from uniqueness. Each application is built upon a new set of algorithm implementations and this work is expended over and over in both the research and the development community. It would appear that a widely distributed and standardized tool that provided both a language capable of expressing a problem in terms of variables, domain ranges, and constraints, and the ability to generate solutions to these problems would be a great benefit. Each of these problems presents something of a different challenge. CSP has basically evolved as those who have written about it have extended it, and there is little agreement about what a "standard" problem might look like, and what types of constraints are typical. Additionally, CSP search covers a very broad range, and different strategies apply best to unique situations. Little if any work has attempted to categorize problem solutions in terms of the problem characteristics. Consequently, the provision of a generic "system" for solution of CSPs is problematic at best.

In summary, we see the following issues of prime importance for a tool that purports to provide a platform for the expression and solution of problems based in constraints:

- *Expressiveness/Representation* : can our problems be expressed concisely and "naturally" but also effectively in the language? For example, when some problem domains have been represented in terms of conceptual abstractions they have been observed to naturally fit certain search methodologies (such as downward refinement of solutions from abstract to specific) and demonstrate substantial practical performance and solution-quality improvements over single-level represented problems. In an abstraction hierarchy, a problem is defined in terms of several levels of components, initially containing only the problem *key* or *integral* features and subsequent changed representations containing additional detail. While some abstraction approaches may be seen as simple heuristic control strategies (for example, variable ordering), richer abstraction representations include mapping information between the levels that can be utilized in more detail control, for example in terms of defining a possible solution space where all "base level" solutions can be found through selecting upper-level abstract solutions. Figure 7 on page 19 details such a mapping feature in a CSP problem. The usefulness of this property in assisting search control has been examined in Woods (1993*b*) for the STR domain, and in other works including Holte (1993) and Bacchus & Yang (1991) and Bacchus & Yang (1992). A system that could not express these abstractions and mappings is by definition not capable of "naturally" representing important characteristics of the problem.

- *Search control* : can the strategy selected be modified dependent on the problem represented, or even possibly based on a given problem instance or dynamically based on problem characteristics?

- *Consistency propagation* : as a precursor to search or in conjunction with search, it often appears to be sensible and beneficial to "propagate" constraints and limit one domain size based on a previous variable domain value selection. Typically, these propagations are done based on problem-specific characteristics (heuristics), or CSP-specific heuristics (such as domain size).

- *Hybridization* : While both use and specification of search strategy and consistency propagation are important, many problems are best expressed through an interleaving or "hybrid" of the two methods. In fact, it has been claimed (Nadel 1989) that all search strategies can be expressed as an interleaving of some amount of in-search constraint propagation and standard backtracking. From this viewpoint, Foward Checking

(FC), LookAhead (LA), Partial LookAhead (PLA), and "intelligent" BT such as BackJumping (BJ) and BackMarking (BM) are all essentially primitive BT with a certain amount of in-node constraint propagation occuring before expansion of the search space to another level. Consequently, construction of interesting or novel search strategies depends upon being able to control (possibly dynamically) the amount of effort spent on consistency propagation at the expense of backtracking throughout the search space.

- *Domain dependent heuristics* are basically specifications about a problem aspect that makes the problem "easier to solve" in some sense. It is imperative that any specification language and solution strategy be powerful enough to support heuristics.
- *Domain independent heuristics* are specifications about CSPs or alternative ways of viewing CSP characteristics, such that this view makes CSPs in general "easier to solve". Using these types of heuristics essentially allows the user to adjust a particular generic strategy into a new one not initially provided in the language. While a "naive" developer may not wish to make use of this type of power, it is essential for the CSP researcher and application developer.

Many languages strive to provide solutions that function as black boxes and allow the language users to "get by" without "seeing" or being forced to learn all of the details the language supports. In fact, for a language to be well-accepted by a wide application-intensive audience, this is likely a prerequisite. However, this goal is often directly in opposition to those who wish to make full use of the features of a language.

Typically, real applications of CSPs have been implemented on a case-by-case basis with all of the software development problems of medium sized software constructed from scratch for a single use. The applications have been overly specific and difficult to modify. Identical work has been repeated ad nauseum in the construction of search and consistency algorithms. Specific domain constraint algorithms have had to be constructed from scratch and integrated into a single framework. These factors are consistent with the world that has been software development. Generally when an application area seems to be widespread and generalizable enough to justify (perhaps financially) the existence of a distributed toolset, one or more emerge and become accepted (take for example the various window manager software for X). The questions for the case of CSP solvers are many. What should go in the toolset? What model should be the basis for arranging the various tools in the toolset? How can constraints be "best" expressed in such a toolset? What does "best" mean - is it "ease of representation", or perhaps "flexibility of representation"? So, for imperative types of languages, a toolset could conceivably provide the "genericness" that will stave off repeated implementations of the same algorithms and also provide a "sensible" framework for constraint solving applications.

But what other possibilities might provide a similar solution framework? Logic Programming "languages" with extensions (CCLP) have been suggested, and in the next section of this paper, we will examine CCLP specifics, and attempt to determine what level of service CCLP provides, and whether this is adequate or desirable for either the researcher or application developer in the domain of combinatorial constraint based problems.

3.4 Extended Logic Programming: a blackbox alternative

So far we have described the "traditional" approach to solving constraint satisfaction problems and have outlined a "typical" real domain with real world application. The "traditional" approach provides a set of tools for the research or application developer that basically serve as a starting point for the construction of efficient problem solvers in one domain or possibly in a broader class of domains. Certainly many of the applications studied have suggested that certain strategies perform better than others, however, little headway has been made to date in explicitly categorizing these domains in terms of which strategy suits which domain and why. Identification of critical domain characteristics that suggest particular solution strategies is an active and interesting research area today.

3.4.1 Logic Programming

Van Hentenryck (1989) states "Logic Programming is based on the idea that *computation can be seen as controlled deduction* ...". In logic programming, a subset of first-order logic known as Horn claus logic was interpreted procedurally in a nondeterministic (i.e. backtracking based) recursive language that we know today as Prolog. Prolog corresponds to an efficient implementation of a particular sound and complete proof procedure for definite clauses.

Prolog, as representative of logic programming languages, has a relational form. Consequently, it provides a convenient way to state constraints which basically describe relationships. Since nondeterminism is embedded in Prolog, the Prolog user can be naive of the details of how this is carried out, and can essentially obtain complete solutions based on simple problem statements while remaining free from the tedium of programming and maintaining this functionality. Thus, logic programming languages such as Prolog provide:

1. A relational method of stating constraints between the arguments of a predicate that is at the root of the language.

2. Nondeterminism implicit in the language.

At first glance then, Prolog appears a natural for representing and implementing constraint problems, and a good conceptual tool for discrete combinatorial problem expression and solution. The obvious explanation of why this is not the case is the complexity of even a moderately sized constraint problem. The sheer combinatoric nature of the possible number of solutions given moderate domain sizes over a variable set results in a large search space. Horrid efficiency problems result when a generate-and-test type of approach is used, which simply generates combinations of variables and domain values and evaluated them as candidate problem solutions. A "perfect" or total solution to a CSP is an assignment from the domain values for each variable to each of the variables template slots. In one simplified problem that corresponds to the STR problem, each of n variables are to be assigned from a pool of m possible domain values, with the ordering of the selection corresponding to an assignment to the variables. There are m choose n candidate solutions for our set, that is, the number of n-combinations of an m-set:

$$NumberCandidateSolutions = m!/n!(n-m)! \tag{2}$$

Thus there are some number of solutions exponential in the size of the number of possible domain values, m. Note that in many problems, including STR, the number of domain values typically far dominates the number of variables. Naively generating and testing these possible solutions is not going to be a good idea, even if we attempt it through a marginally better approach such as backtracking. As pointed out in Van Hentenryck (1989), this type of approach is dedicated to "recovering" from a failed assignment of value to variable via backtracking, rather than "avoiding" such an occurrence without blindly wandering into it.

Van Hentenryck (1989) outlines the design of Prolog from a position where an algorithm is defined as $algorithm = logic + control$, where *logic* refers to the facts and rules specifying what the algorithm does, and control refers to how the algorithm can be implemented by applying the rules in a particular order[17]. Extensions or improvements to logic programming languages such as Prolog are typically based on the preservation of the *logic* aspect, and focus on refinement of *control*. Essentially this approach results in a situation where the generate and test "problem conception" can be maintained, with the supposed simplicity and ease of use this suggests, and yet the efficiency of improved backtracking can be enjoyed.

Assuming an underlying Generate and Test scheme, control in Prolog is based on the *Prolog Computation Rule (PCR)* which determines two things:

- The order in which goals will be attempted to be satisfied, where the goals are the elements of a query or resolvent.
- The order in which rules will be applied, where the rules are the facts in the database of facts and rules.

If the PCR were changed somewhat so that:

1. if constraints of the form of disequations[18] were taken as goals only if they are ground,
2. as soon in the resolution process as the goal set or resolvent contained some ground disequations, one is selected as the current goal

[17] Based on Kowalski's division of labor described briefly in Sethi (1990).

[18] Constraints that state that certain variables cannot take on the same value.

then the way in which Prolog searched would be based on Generate and Test, but would behave as Backtracking since selection of a *constrained to fail* variable domain value would immediately signal a dead end and a backtrack point.

In fact, Prolog implementations frequently make use of more complicated search strategies known as "Intelligent Backtracking". These strategies basically attempt to reason in a limited way about the "failure cause". On this basis, they backtrack to a previous variable instantiation that is conflicting, via some constraint, with the current instantiating variable. They do not blindly backtrack to some intermediate variable that may not be involved with the constraint, as is the case with standard Backtracking. Van Hentenryck (1989) points out that while this technique can be an improvement, there are theoretical limitations that imply the necessity of using heuristics for finding the correct backtracking points, and practical limitations as there may be significant overhead in locating the backtrack point. Further, it is observed that it is better to try to avoid these failures rather than to react to them in some manner after the fact. Towards a more systematic approach, Lookahead search schemes are suggested as a better way of reducing the search space *a priori*. In fact, a flexible form of this systematic reduction of the space is precisely what has been discussed in the context of our previous requirement of *Hybridization* for effective CSP formulation. In Section 3.4.3 we shall examine more precisely this extension of logic programming to see to what extend the hybridization requirement is met.

In our previous discussion of Section 3.3.2, consistency techniques were introduced which reduce the search space prior to the explicit discovery of a failure in a backtracking search approach. Spending work during search in removing variable domain values on the basis of certain local constraints is essentially preventative work - preventing a dead end before encountering it during standard backtracking. Van Hentenryck (1989) points out quite correctly that the value of this preventative work has been shown in many application domains[19] to produce substantial improvement over standard backtracking. Further, it is stated that it should be considered a fundamental primitive in CSP solution.

It would seem a natural extension to a logic programming language to provide consistency propagation tools as a basic part of a new language. Basically, that is the purpose of the work described in Van Hentenryck (1989). In this work, the intent is to provide consistency techniques *inside* logic programming so as not to obscure the descriptiveness that supports constraint expression. It is observed that logic programs can be written that support these techniques explicitly but that this approach "complicates" programs (surely they can be no less complicated than a standard procedural implementation and quite likely more difficult to understand), and are very inefficient.

3.4.2 From Logic Programming to what ?

Surely the construction of embedded consistency techniques in a search-based language with formal descriptive qualites is an important goal, but what form should this embedding

[19] Experiments typical of those described in Nadel (1989) support this statement.

take? Much research is being undertaken *currently* (Prosser 1993, van Run 1994, Woods 1993*a*, Woods 1993*c*, Yang & Fong 1992) on search strategies that are appropriate for CSPs in various domains, according to specific properties that have been observed in CSPs. New consistency propagation techniques are being developed and old ones improved on constantly as the nature of CSP problems in specific and general become better understood. Interesting work (Nadel 1989, Prosser 1993, van Run 1994) has shown that re-combination of search and consistency propagation directly results in unique and often more efficient search strategies, particularly in specific domains. Certain domains seem to benefit with more in-search propagation while others from less. Other work has attempted to view constraint propagation and search from a single perspective. In this view, the search space may be reduced at any time through constraint propagation techniques, and that portion explored via search is viewed as part of a single problem space that is navigated with a dynamically changing search procedure (Nadel 1989). Work such as Nadel (1989) and more recently Prosser (1993) have attempted to make generalizations about certain problem properties and relation on search space and search performance, however, there does not current exist a theory of knowledge structure and search that suggests definite causal links between problem structure and search performance.

We have seen the "naturalness"[20] with which logic programming seems to be capable of representing relationships such as constraints. Is this "natural approach" sufficient to meet our previously stated requirement for *expressiveness*? Since it has been understood for sometime (indicatively demonstrated for CSPs in Nadel (1990)), that problem representation can have a dramatic effect on the complexity of problem solution both formally and in practice, do the facts support the formal declarative qualities of logic programming as being "rich" enough to support alternative representations and approaches? For example, the work of Freuder & Wallace (1992), Yang & Fong (1992), and most recently Woods (1993*a*) has indicated that representing constraint problems in a hierarchical structure is potentially valuable in terms of problem solution time. Further, this representation structure, as shown in Figure 7 on page 19, may provide more than an ordering heuristic for variable binding in that an explicit mapping between subsequent hierarchical levels of the CSP is in effect supplying additional problem information that would be lacking in a simple declaration about variables and variable domains, even taking into account the semantics of constraints (Holte et al. 1992, Holte et al. 1993, Holte 1993). We have seen the logic programming paradigm of algorithms defined as the sum of *separately defined* logic and control where

[20] Van Hentenryck (1989) is quite fond of describing the *natural* link between logic programming representation with relations (as in Prolog) and formulating constraints. This claim is apparently based on two things: (1) the author is exceedingly familiar with the realm of logic programming, and therefore the logic programming paradigm is "natural", and (2) relations are a common feature to domain experts who would commonly wish to specify such things for a machine to solve, and consequently fit into some "typical" expert's world-view. It immediately comes to mind that the "natural" way of representation (or of doing anything else, for that matter) is not necessarily the "most beneficial", "most efficient", or "most effective" way.

control basically determines the way in which the logic is applied. However, abstracted problem representations may be utilized differently. The “logic” represented by the problem specification has a structure that may be exploited explicitly by the control, depending on problem instance. The “logic” of encoded connections between abstraction levels may actually serve as problem solution control information as well.

The intent in raising the issues of search and consistency technique integration and CSP representation issues is to point out that logic programming languages provide both more and less than standard imperative or functional programming languages in terms of tools for constructing algorithms. Logic programming languages provide an additional “assumed” formal representation style that must be acknowledged. However, while formalism restricts and standardizes problem representation, it can result in oversimpliflication and loss of important problem content. Additionally, since logic programming has important in-built algorithmic underpinnings, traditional declarative and functional programming languages provide little in the way of inherent “high level” algorithmic restrictions. Rather, these languages tend to support or provide these “advanced” algorithmic features through commercially or publicly available software libraries of toolsets. These toolsets must be directly manipulated by the program or application developer, and tool characteristics can be explicitly used or ignored as desired.

It is easy to imagine a situation where a particular language embeds a single multifaceted sorting procedure, and advertises to the language user this procedure as a generic “all-cases-encompassing” sorting tool. It is conceivable that this tool includes heuristics that are capable in many situations of adequately determining a search strategy based on input dataset characteristics. However, the question of all-case appropriateness would seem to be better left to someone familiar with the type of input or problem semantics than to heuristics that are at best partially effective, at least in the absence of a more encompassing theory about data sortedness and sorting algorithm applicability.

Standard logic programming languages embed a control strategy based upon one type of backtracking. This search strategy may be one of many “intelligent” strategies that have been shown to perform well in various cases, however, there is no existing theory that will map problem representation accurately to particular search strategy performance. We must be concerned with determining whether logic programming extended with constraint processing and modified control structures still falls into trap of limiting the developer unwarrantedly. Can the backtracking (albeit “intelligent” backtracking) incorporate consistency application, and consequently easily benefit from the wealth of information known about constraint propagation methods?

As we examine this extended version of logic programming we must attempt to keep in mind the primary issue of determining whether or not this extended language can provide the *expressiveness*, *search control*, *consistency propagation*, *hybridization*, and incorporation of *heuristics*, early stated in Section 3.3.2 as necessary componenets of any set of tools intended to construct solutions to Constraint Satisfaction Problems.

3.4.3 Logic Programming Extended

In this section we will describe the extensions to logic programming described explicitly in Van Hentenryck (1989) as they relate to the embedding of consistency propagation techniques, and explain how this new language is able to handle many constraint problems including STR with a very simple encoding. Essentially extended logic programming is providing a toolset that is capable of propagation of constraints to reduce domains during search and relies on three detailed search strategies that have been shown to be successful across many practical problems[21].

The primary extensions are:

1. **Predicate domain declarations** or constraints which restrict the range of the arguments of predicate's variables, i.e. domain variables.

2. **General (or weak) methods** for controlling the inference process in the absence of much domain-specific control information. Specifically, these methods result in the use of particular inference rules with specific constraints. Essentially, three types of declarations are potentially introduced, each which entails the creation of a different control rule. These declarations are:

   - **Forward Declarations**, which define *for each predicate* that the predicate is a constraint, and that forward checking inference may be applied for this predicate. Predicates or constraints are defined so that the last variable V_n (or forward variable) in a predicate $P(V_1...V_n)$ has its domain reduced when the first $n - 1$ variables in P are instantiated to ground. In this way, the domain of V_n is explicitly reduced during search, and the values discarded cannot be used again as values for V_n. The ground value of V_n is determined only when there is precisely one domain value remaining for V_n. These declarations support the construction of a Forward Checking (FC) search strategy in logic programming.
   - **Lookahead Declarations**, basically generalization of Forward Declarations, where when P is a constraint, at least one of the variables of the constraint is a domain-variable. For these predicates, instantiation of the ground variables results in a reduction of the space of the domain variables, possibly resulting in a reduction of one or more of them. This is described in Van Hentenryck (1989) as a general mechanism for enforcing n-consistency between the lookahead-variables.
   - **Partial Lookahead Declarations**, which are applicable in the same cases as the Lookahead Declarations, but which are basically a cross between Forward and Lookahead, where some of the domain values are reduced based on certain specific partial variable groundings in the P predicate/constraint.

[21] These domains include graph coloring, scheduling with constraints, and several other optimization problems.

Of critical importance to this paper is the fact that Lookahead may be interpreted as generalization of AC-3 and AC-4 consistency propagation algorithms as described in Mackworth (1977), Mackworth (1987) and elsewhere. Van Hentenryck (1989) observes that both techniques have advantages and disadvantages *dependent* on the problem in particular, and the choice is best left as a parameter to the user. This observation is *critical* since these two consistency techniques are only but two representatives of a continuous string of newly emerging algorithms (of both global and local applicability). Therefore this selection of two representatives is woefully inadequate, and their combination into search is insufficiently flexible to provide a full range of capability of the hybrid search algorithm. Essentially the search underlying this extended language is hybrid, but only a restricted subset of hybrids can be represented, and worst of all, there is no way to extend this coverage without further extension to the language.

3. **Specific (or strong) methods** applicable when specific information about a constraint is known explicitly in advance. Specific methods achieve the same pruning as is the case for general methods, however, are typically specializations of the general methods for particular constraints and domains. One obvious example of a specific is arithmetic rules in a mathematical domain where the instantiation of two variables might imply a new range on a third that is specified by some arithmetical combination of the first two.

4. **Choice methods** governing the selection of value ordering for a particular variable, and variable selection for instantiation. The following are supported[22]:

   - **Variable Choice**, based on the first-fail principle where we wish to try variables that are likely to fail first, essentially trying the most difficult part of the search space first. Thus, first-fail bases the selection on the variable that is the most constrained in terms of the smallest domain. Since the extended logic language is maintaining explicit domain ranges, this is possible where traditional logic languages had no equivalent explicit representation. Another provided optional heuristic improvement is choosing the variable (in case of domain size ties) that will have to satisfy the most constraints.

   - **Constraints as Choices**, constraints on the search space may take the form not only of assumed variable bindings from a domain, but also assumed constraint results without a constraint evaluation where one is not possible, perhaps due to insufficient binding information. This constraint assumption becomes a backtrack point just as a variable instantiation point[23].

[22] Notably missing from this list is domain value ordering, an often discussed search heuristic, that may have easy application in some domains (such as the case where a smaller domain value may be desirable).

[23] This is a very interesting extension that has not to my knowledge been explored very deeply to date in AI/Search research, at least not experimentally with CSPs.

5. **Optimization methods**, including branch and bound search that supports min/max types of problems based on problem splitting coupled with two options for search: best-first and depth-first.

3.4.4 What extended logic programming provides

Logic programming languages are well known and wide-spread. Consequently, it is reasonable to assume that many programmers and researchers are familiar with and comfortable with the representation style and behaviour of these languages, and will follow the conceptual jump to improved search behaviour and constraint support in such languages. The question of whether they provide adequate expressiveness appears to have been overlooked in favor of "familiarity" and convenience.

Use of extended logic programming for the representation and solution of constraint satisfaction problems is certainly going to be more efficient than was the case for standard logic programming for the simple reason that the extensions have made use of search procedures and heuristics that have been demonstrated in specific experimental work in the *imperative* world of problem solution, where search algorithms are coded and tested across various domains. Earlier logic programming backtracking and *a posteriori* constraint checking has been abandoned in the rest of the constraint satisfation world for many years in other than toy domains.

It is certainly true that provision of a standardized approach and toolset for the representation and solution of CSPs is beneficial, but extension of logic programming has accomplished this only in part, and from the point of view of encompassing the richness and flexibility of hybirdized search, it is not clear at all what justification can be given for limiting to just three control strategy basics. Standardization must creep in in any tool that hopes to be widely used, however, the overlimitation of these design assumptions appears to be only a product of the language used (logic programming) and motivated by little else.

3.4.5 What is missing in extended Logic Programming?

1. Representation Issues

   New languages derived from logic programming languages such as Prolog which retain the "relational" based problem description are limited in that richer representation schemes such as abstraction do not appear to fit cleanly into the language approach. Other work has been discussed that makes use of this additional expressivness to model problem characteristics such as explicit expert problem decomposition into parts or into a hierarchy of abstraction based specifications. It seems that the simplicity that lends the attractive intuitive "naturalness" of constraint specification as relations may actually be a strong restriction to problem description.

2. Control Issues

Logic programming intentionally draws a hard boundary between problem specification or *logic* and problem solution or *control.* Recent work has shown decisively that problem representation can not only be exploited in control or search strategies, but that richer problem representation languages that can account for abstration or problem decomposition can achieve great performance improvements specifically by crossing this boundary, and allowing representation to guide control. It is interesting to note that, in a sense, the control specification inherent in these richer schemes is exploited on the basis of other "naturalness" arguments such as "problems are more naturally represented as hierarchies of increasing complexity", or "problems are more naturally represented as a sum or their parts". The effectiveness of divide-solve-combine algorithms in many other areas of computer science is another example of this type of thinking. Decoupling of control and representation may be "clean" but it doesn't appear to be "smart".

Extended logic programming as described in Van Hentenryck (1989) extends traditional logic programming primarily by improving on the control rule underlying logic programming languages to take advantage of three "better" control rules based on Forward Checking search, Partial Lookahead search, and Lookahead search. These strategies attempt to reduce the search space through domain reduction during search *a priori* and consequently may be identified as three discrete combinations of search and consistency propagation selected from a continuous world of possible combinations of search and consistency algorithms. While many domains show dramatic improvement with the use of these search algorithms in place of standard "intelligent" backtracking, this author is unaware of any theory which would suggest that these three are definitively superior to other search strategies, or represent some "best merge" of consistency and propagation with respect to the types of problems they will be applied against. Consequently, it appears that limiting the degree of flexibility of search to these three control rules is unnecessarily limiting in general.

3.4.6 Conclusions

Logic programming is unnecessarily limited in terms of both representation and division between logic and control, possibly at the expense of both performance and accuracy of problem representation. In conclusion, the extension of logic programming with search strategies that embody a higher degree of in-search consistency propagation will provide a language that may be applied to domains of increased complexity. However, since there is no clear break-point between research and application that suggests these strategies as "superior" in some application sense, it seems arbitrary to limit to these only, when perhaps a more general control structure with a flexible approach to consistency propagation in-search would be more appropriate.

It is true, that extended logic programming as described in Van Hentenryck (1989) is now capable of addressing several real domains. It provides a toolset that includes the primary sources of efficiency demonstrated by earlier *imperative* application testing such as the AC-3 and AC-4 arc-consistency algorithms coupled at least partially flexibly with a search engine capable of accomodating several types of search heuristics.

Additionally, the representation expressiveness appears to be sufficient to achieve reasonable results across many common CSP domains, and with relative ease of programming since the underlying control and algorithmic necessities of such a system are effectively hidden from the user. However, these same features and functionality can be observed in library toolsets that provide AC algorithms and search procedures in parameterized and connected fashion along with a language capable of easily expressing constraints. Since such a toolset is not known to exist at present, extended logic programming languages seem to be the easiest and best available tool, however, their lack of flexibility and ability to include extended algorithms in place of the ones ensconsed in the language will preclude their longevity in such a dynamic and growing area.

Acknowledgments

This work has been undertaken as part of a collaboration with the Canadian Department of National Defense, Defence Research Establishment Valcartier. The author is a PhD candidate supported by an NSERC scholarship and the University of Waterloo.

References

Anthony, M. & Biggs, N. (1992), *Computational Learning Theory*, Cambridge University Press.

Bacchus, F. & Yang, Q. (1991), The downward solution property, *in* 'Proceedings of the 12th IJCAI', Sydney, Australia, pp. 286–292.

Bacchus, F. & Yang, Q. (1992), Downward refinement and the efficiency of hierarchical problem solving, personal correspondence.

Booth, Kellogg, S. & Colbourn, Charles, J. (1977), Problems polynomially equivalent to graph isomorphism, Technical Report CS-77-04, University of Waterloo.

Christensen, J. (1990), A hierarchical planner that creates its own hieararchies, *in* 'Proceedings of the 8th AAAI', pp. 1004–1009.

Cormen, T. H., Leiserson, C. E. & Rivest, Ronald, L. (1992), *Intoduction to Algorithms*, McGraw-Hill Book Company.

Dechter, A. & Dechter, R. (1987), Removing redundancies in constraint networks, *in* 'Proceedings of the 6th AAAI', pp. 105–109.

Dechter, R. (1990), From local to global consistency, *in* 'Eighth Canadian Conference on Artificial Intelligence'.

Dechter, R. (1992), 'From local to global consistency', *Artificial Intelligence* **55**, 87–107.

Dube, T. & Yap, C.-K. (n.d.), The geometry in constraint logic programs, c/o Dominic Duggan, cs642.

Eades, P. & Ng, H. C. (1987), 'An algorithm for detecting symmetries in drawings', *ARS Combinatoria* **23A**, 95–104.

Freuder, E. & Wallace, J. (1992), 'Partial constraint satisfaction', *Artificial Intelligence* **58**, 21–70.

Freuder, E. C. (1991), Eliminating interchangeable values in constraint satisfaction problems, *in* 'Proceedings of the 9th AAAI', Vol. 2, pp. 227–233.

Garey, M. R. & Johnson, D. S. (1979), *Computers and Intractability : A guide to the theory of NP-Completeness*, W. H. Freeman and Company, Bell Laboratories, Murray Hill, New Jersey.

Ghahraman, D., Wong, A. & Tung, A. (1980), 'Graph optimal monomorphism algorithms', *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-10**(4), 181–188.

Hoffmann, C. (1982), 'Subcomplete generalizations of graph isomorphism', *Journal of Computer and System Sciences* **25**, 332–359.

Holte, R. C. (1993), Automatic change of representation, personal correspondence.

Holte, R., Zimmer, R. & MacDonald, A. (1992), When does changing representation improve problem-solving performance ?, *in* 'Proceedings of the Workshop on Change or Representation and Problem Reformulation'. NASA Ames Technical Report FIA-92-06.

Holte, R., Zimmer, R. & MacDonald, A. (1993), A study of the representation-dependency of abstraction techniques, *in* 'Proceedings of the 1993 Workshop on Knowledge Compilation and Speedup Learning'.

Johnson, D. (1982), 'The NP-completeness column - an ongoing guide', *Journal of Algorithms* **3**(3), 288–300.

Kasif, S. & Delcher, A. (n.d.), Local consistency in parallel constraint-satisfaction networks, c/o Dominic Duggan, cs642.

Kumar, V. (1992), 'Algorithms for constraint-satisfaction problems', *AI Magazine* pp. 32–44.

Lam, W. & Bacchus, F. (1993*a*), Learning bayesian belief networks: An approach based on the mdl principle, Technical report, University of Waterloo.

Lam, W. & Bacchus, F. (1993*b*), 'Using causal information and local measures to learn bayesian networks', *UAI*.

Lam, W. & Bacchus, F. (1993*c*), Using new data to refine a bayesian network, advance copy.

Lapointe, S. & Prouix, R. (1994), Fuzzy geometric relations to represent hierarchical spatial information, *in* 'Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence'. to appear.

Lapointe, S. & Proulx, R. (1994), Statistical identification of templates in message data, Contract Report W7701-3-0847/01-XSK, APG inc. (1994). Combat Intelligence Automation Group, Defence Research Establishment Valcartier.

Lubiw, A. (1981), 'Some NP-complete problems similar to graph isomorphism.', *SIAM Journal of Compting* **10**, 11–21.

Mackworth, A. (1977), 'Consistency in networks of relations', *Artificial Intelligence* **8**, 99–118.

Mackworth, A. (1981), Consistency in networks of relations, *in* Webber & Nilsson, eds, 'Readings in Artificial Intelligence', Morgan Kaufmann Publishers Inc., pp. 69–78.

Mackworth, A. (1987), Constraint satisfaction, *in* S. Shaprio, ed., 'Encylopedia of Artificial Intelligence', Vol. 1, John Wiley and Sons, pp. 205–211.

Mackworth, A. (1992), 'The logic of constraint satisfaction', *Artificial Intelligence* **58**, 3–20.

Mackworth, A. & Freuder, E. (1985), 'The complexity of some polynomial network consistency algorithms for constraint satisfaction problems', *Artificial Intelligence* **125**, 65–74.

Mackworth, A. & Freuder, E. (1993), 'The complexity of constraint satisfaction revistited', *Artificial Intelligence* **59**, 57–62.

Manning, J. (1991), 'Computational complexity of geometric symmetry detection in graphs', *Lectures in Computer Science*.

Menezes, F., Barahona, P. & Codognet, P. (n.d.), An incremental hierarchical constraint solver, c/o Dominic Duggan, cs642.

Minton, S., Johnston, M., Philips, A. & Laird, P. (1992), 'Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems', *Artificial Intelligence* **58**, 161–205.

Montanari, U. & Rossi, F. (n.d.), Constraint satisfaction, constraint programming, and concurrency, Provided by Dominic Duggan for cs642.

Nadel, B. (1990), 'Representation selection for constraint satisfaction: a case study using n-queens', *IEEE Expert* pp. 16–23.

Nadel, B. A. (1989), 'Constraint satisfaction algorithms', *Computational Intelligence* **5**, 188–224.

Prabhu, G. & Narsingh, D. (1984), 'The ellipsoid algorithm and the graph isomorphism problem', *Technique et Science Informatiques* **3**, 327–333. in English.

Prosser, P. (1993), 'Hybrid algorithms for the constraint satisfaction problem', *Computational Intelligence* **9**(3), 268–299.

Sacerdoti, E. (1974), 'Planning in a hierarchy of abstraction spaces', *Artificial Intelligence* **5**, 115–135.

Schank, R. (1992), 'Where's the ai?', *Artificial Intelligence Magazine* **12**(4), 38–49.

Sethi, R. (1990), *Programming Languages*, Addison-Wesley.

Stacey, D. & Wong, A. (1988), A largest common subgraph isomorphism algorithm for attributed graphs, University of Waterloo, Systems Design Engineering.

Tolba, H., Charpillet, F. & Haton, J.-P. (1991), Representing and propagating constraints in temporal reasoning, *in* 'Proceedings of the International Conference on Tools for Artificial Intelligence', pp. 181–185.

Van Hentenryck, P. (1989), *Constraint Satisfaction in Logic Programming*, The MIT Press.

Van Hentenryck, P., Saraswat, V. & Deville, Y. (unknown), 'Constraint processing in cc(fd)', *unknown*.

van Run, P. (1994), Domain independent heuristics in hybrid algorithms for csps, Master's thesis, University of Waterloo.

Waltz, D. (1975), Understanding line drawings of scenes with shadows, *in* P. Winston, ed., 'The Psychology of Computer Vision', McGraw Hill, Cambridge, Massachusets, pp. 19–91.

Wasserman, P. D. (1989), *Neural Computing Theory and Practice*, Van Nostrand Reinhold.

Wong, A., You, M. & Chan, S. (1990), 'An algorithm for graph otpimal monomorphism', *IEEE Transactions on Systems, Man, and Cybernetics* **20**(3), 628–636.

Woods, S. (1993*a*), Interactive recognition of spatially defined model deployment templates, Research memorandum, Combat Intelligence Automation Group, Defence Research Establishment Valcartier, Courcelette, Quebec CANADA. unclassified.

Woods, S. (1993*b*), A method of interactive recognition of spatially defined model deployment templates using abstraction, *in* 'Proceedings of the 1993 DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics'.

Woods, S. (1993*c*), Upward exploitation of constraint propagation in hybrid constraint satisfaction search., Defence Research Establishment Valcartier, Dept. of Defence, Courcelette, Quebec, CANADA. unclassified.

Woods, S. G. (1991), An implementation and evaluation of a hierarchical non-linear planner, Technical report cs-91-17, Computer Science Department, University of Waterloo.

Yang, Q. & Fong, P. (1992), Solving partial constraint satisfaction problems using local search and abstraction, Technical Report cs-92-50, University of Waterloo.

Yang, Q. & Tenenberg, J. (1990), Abtweak: Abstracting a nonlinear, least commitment planner, *in* 'Proceedings of the 8th AAAI', Boston, MA, pp. 204–209.

Yang, Q., Tenenberg, J. & Woods, S. (1993), Abstraction in nonlinear planning, Was University of Waterloo Technical Report CS-91-65.