A method of interactive recognition of spatially defined model deployment templates using abstraction

Steven Woods
Defense Research Establishment Valcartier,
Combat Intelligence Automation Group,
Courcelette, Quebec G0A 1R0.
woods@jupiter.drev.dnd.ca

Abstract

Abstraction has been suggested intuitively and shown experimentally to be an effective method of improving the efficiency of various types of problem solving applications such as scheduling and planning [11, 13]. In this paper we present an interactive system for model deployment template recognition that uses abstraction as the basis for both model representation and user interaction, and which provides a convenient model for iteratively improving confidence in proposed template matches or solutions.

The problem

An intelligence analyst must draw on specific experiential and learned knowledge of enemy behaviour in order to recognize specific deployment patterns signifying organizational relationships of enemy elements sighted. One aspect of this complex task is to recognize 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" 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 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. Sightings are derived from intelligence messages received from a diverse set of sources and include direct sightings and contact reports and electronic or other sensor telemetry. 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 extremely complex, dynamic and incomplete "view" of the enemy. Both the current situational battlefield view and the deployment templates themselves are approximations of the real situation. Messages from the field may contain vague or conflicting composition or location information, and inaccurate messages may be propagated into inaccurate conclusions. 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 inter-connected 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 we may wish to determine which of many partial solutions is more promising, possibly 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 assist the analyst recognition task with limited automated tools will need to be able to take advantage of the experience and flexibility of an intelligence analyst, and any such system must therefore be designed with this interaction as a primary goal.

There are many interesting research issues in the formulation of spatial deployment templates as well as in the development of an interactive process capable of assisting in template recognition in noisy situations. While we concentrate here on the construction of the process, this work builds upon the representation work in [9] where we introduced one structure capable of capturing analyst's *Expectations* based on reported sightings, and allowing for the explanation and propagation of these into later conclusions about the situation. The interactive approach discussed in this paper is essentially a continuation of our work on modelling analyst's expectations by allowing for the definition of commonly seen or expected patterns of deployment, and of modelling the analyst's process used to recognize these patterns.

This paper has two primary goals. First, we describe the way in which we model templates and use these models to locate template instances in large data sets. Second, we detail the model of interaction and instance identification we have developed, noting the importance of abstraction as an interaction methodology which allows an intelligence analyst to locate instances in a timely manner, provides approximate solutions where complete ones are absent, and which can be tailored to fit specific analyst needs or problem instances.

Template modelling

The intelligence messages containing enemy unit or equipment sighting information are of varying accuracy and completeness with respect to perceived structure, location and time. Messages describe a sighting at some point in time $T1$, are sent for collation at some other time $T2$, and arrive later at $T3$. The analyst attempts to organize hundreds of these messages at some still later time $T4$ into a coherent picture of the world at $T4$. Any sighting or perception of the world at $T1$ decays or loses its information value over time in a dynamic setting where units move, and groupings disintegrate into smaller elements, or reintegrate to form new formations. As a result, analysts commonly attempt to form a picture for some past time TP somewhere before $T4$ and after $T1$ based on all reports available at $T4$. This single-time model created of the past at TP will be considered to be the input to our system. Essentially we are working with a blurry snapshot of a moment in the past. While the ultimate goal of this research is to attempt to recognize templates according to a dynamic model over time, in this paper we discuss a static model only.

Templates defined as groupings of constrained element slots may be represented by a constraint graph, where the nodes of the graph represent the slots, and the arcs between nodes represent constraints. Constraints between slots typically but not exclusively capture spatial relationships, and reflexive constraints for each node may identifying features such as type, size, or orientation. Problems specified in this fashion, where a solution is composed of a set of consistent element-to-slot assignments are well described in [4], and are known as a Constraint Satisfaction Problems (CSPs). An example of a template constraint graph is shown in Figure 1 with five node slots where each slot is constrained by several attribute ranges, and 11 inter-slot spatial constraints on distance, position, and orientation.

We are interested in finding all possible instances of a particular template in a set of situation elements, and consequently we are interested in finding all solutions to a CSP modelling our template recognition problem. We have constructed a non-interactive system known as the MOdel Template Object Recognition System (MOTORS) [10] which is capable of solving CSPs in general, and spatially oriented CSPs such as template recognition in particular.

In the literature, CSPs 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. Recent experimental results [7] have shown that hybrids which perform either some degree of constraint propagation *before* or *during* search tend to have better performance than either individual approach. 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, and various strategies more easily compared empirically. Although [7] 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 doctrinal template recognition problem provides an excellent example of a moderately complex problem which can be addressed in a reasonable length of time utilizing hybrid search.

In our work with MOTORS detailed further in [10], we experiment at some length with combinations of search, propagation algorithms, and various heuristics. For instance, template recognition is a domain that is inherently spatial, and while the spatial relationships may be precisely represented in the form of inter-slot constraints, a template-wide constraint that requires that all situation elements in consideration for a particular instance exist within a certain "range of focus" has been adapted in MOTORS as an example of a specific domain-dependent heuristic for template recognition. The interactive model discussed in this paper is designed so as to provide a framework in which further research in appropriate combinations of heuristic search and propagation can be undertaken.

Figure 2 details a situation with one embedded Figure 1 template instance amid 300 randomly created and positioned elements. The time required by a simple backtracking configuration of non-interactive MO-

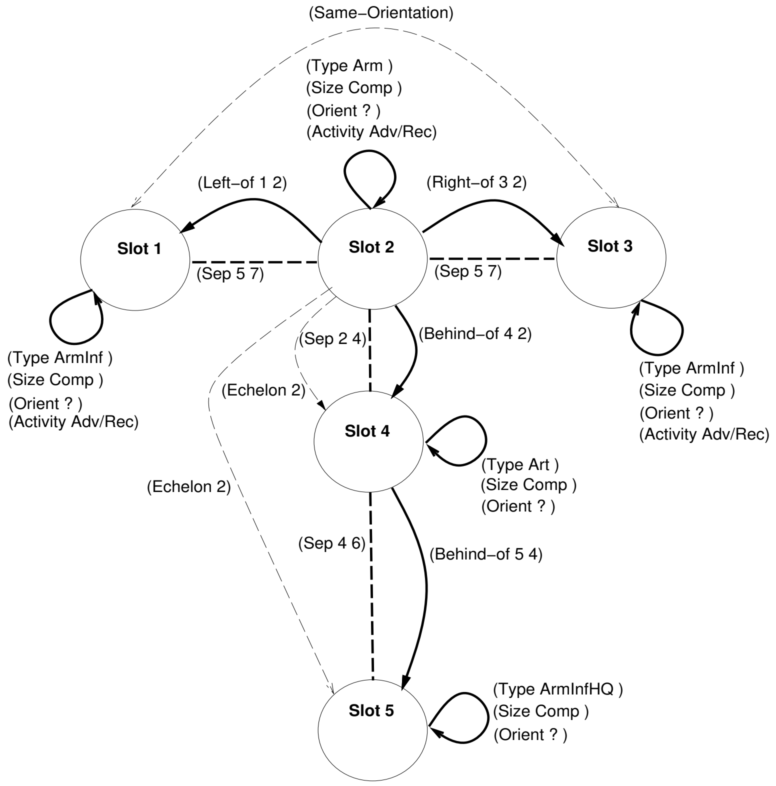

Figure 1: Whole CSP 5 variables, 11 arc constraints.

TORS parameters to find all instances (in this case one) of the template in this situation on a Sparc10 workstation is less than 1 second. Figure 3 shows the template instance in its same orientation as in the example situation, along with two instances that only partially match the template slots. It is quite easy to see how difficult the process of identifying these templates can be amidst noisy data. The manner in which inter-slot spatial relationships can flex greatly complicates the identification process for the human observer, and template recognition can be seen as much more complex than identifying objects with regular structure and orientation.

Experimentation and research involving hybrid search using domain-independent heuristics for selection of propagation methods, methods of sharing constraint propagation work during search, and domain-dependent spatial heuristics are discussed further in [10].

Abstraction and partial solutions

In light of our preliminary results which showed that moderately complex situations could be solved completely for one template instance with relatively little difficulty, we considered how to better model the partial matches that must be accommodated in a real situation. Completely matching a particular template involves matching the template's slots with specific enemy elements such that the all spatial and identity constraints between slots are satisfied. When such a complete assignment cannot be found within the element set or situation, it may be acceptable that some subset of slot assignments that only satisfy some "sufficient" portion of the constraints be accepted. This partial solution to the problem is a candidate which may suggest the existence of unsighted situation elements, and the analyst must accept or reject these based on past experience. Advance analyst work in describing acceptable or "plausible enough" partial solution characteristics and identifying template constraints which may be relaxed in the search for usable solutions will limit the partial instances located.

Given a template definition, two approaches of locating partial solutions for CSPs are immediately evident: solving some portion of the original CSP, or solving an abstraction of the original CSP. In the first case, the problem could be partitioned into independent parts, where the lack of a solution to one part would not preclude the solution to another part, and the solved part would represent a partial solution of the whole problem. In the second case, the problem could be considered wholistically where the most "important" or "key" aspects of the whole problem could be identified and solved initially. Solutions for the ab-

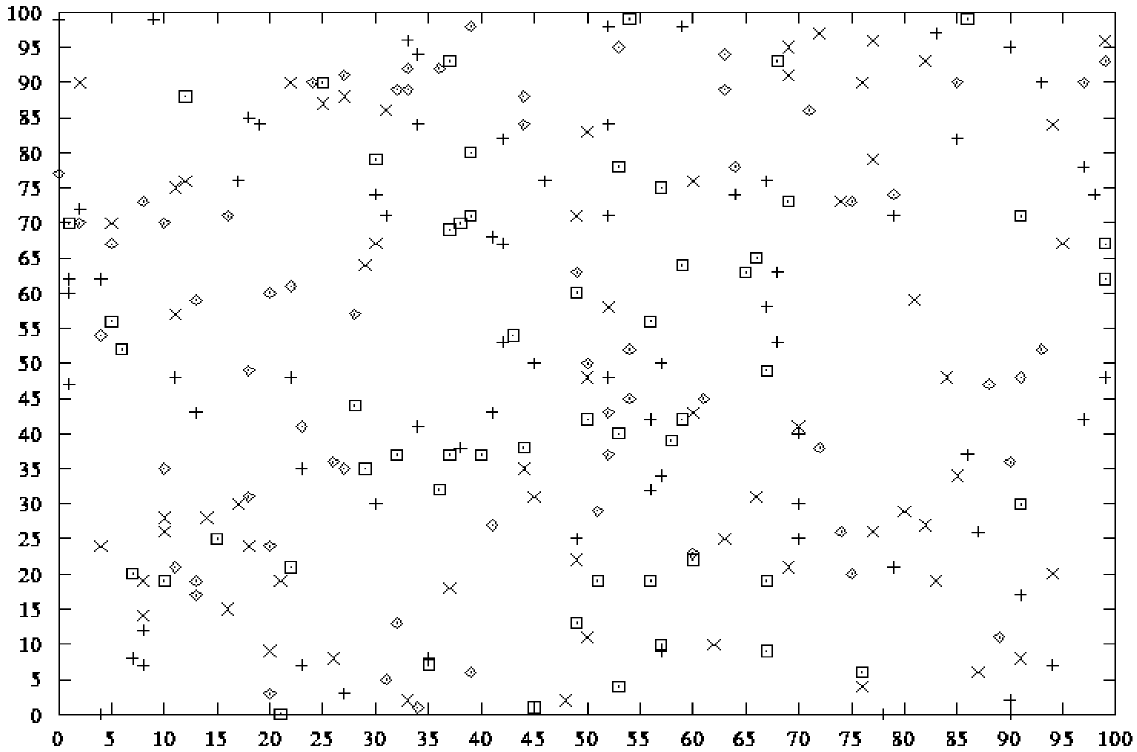

Figure 2: Situation with 300 unit sightings and 1 whole template instance.

stracted problem composed of only "key" constraints and slots could then be refined by progressively fitting problem detail if possible, increasing the perceived degree of fit for those solutions satisfying additional constraints. Intuitively the progressive fit is attractive when considering systems that must offer solutions in domains where the problem structure itself has been defined according to incompletely captured expert knowledge. In [10] we discuss partition of CSPs further, but this paper will concentrate on developing a model for the solution of CSPs based on solving abstract problem representations before more specific or detailed ones.

In [2], the notion of Partial-CSPs (PCSPs) is outlined. A PCSP is a "relaxed" or "less constrained" version of the original CSP. As originally formulated, if a solution could not be found to the fully constrained problem, one could relax the problem in one of several possible ways and attempt to solve this new, simpler problem. This process of attempt, fail, simplify and repeat continues until some "minimally relaxed" CSP is solved. The repeated relaxation of the original CSP defines a space of relaxed CSPs with the original CSP at the root, and the most relaxed CSPs at the leaves, as shown in Figure 5.

The specification of an abstraction hierarchy for a particular CSP involves the identification of the most "important" variables and constraints. Abstraction is

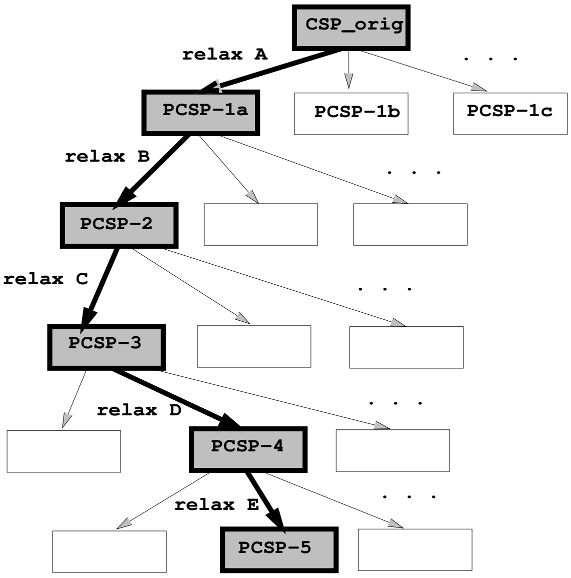

Figure 5: One abstraction hierarchy in a PCSP space.

often taken to mean specifying the differences between "more specific" and "less specific" problem representations. There are exactly 3 ways of creating an abstracted CSP: removing variables or slots, removing constraints, or relaxing constraints. If the "most abstract" problem representations have the "most im-

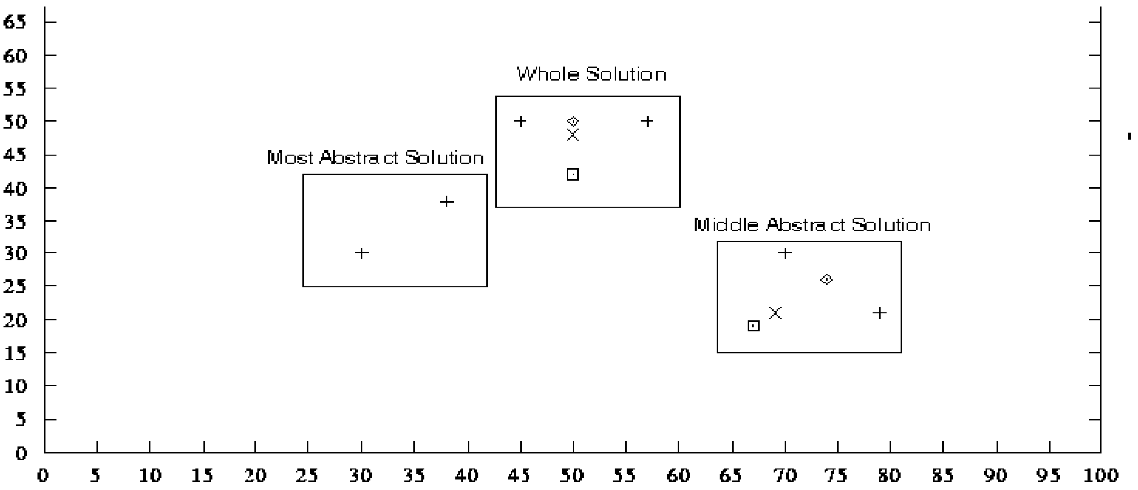

Figure 3: Template instances hidden in Figure 2.

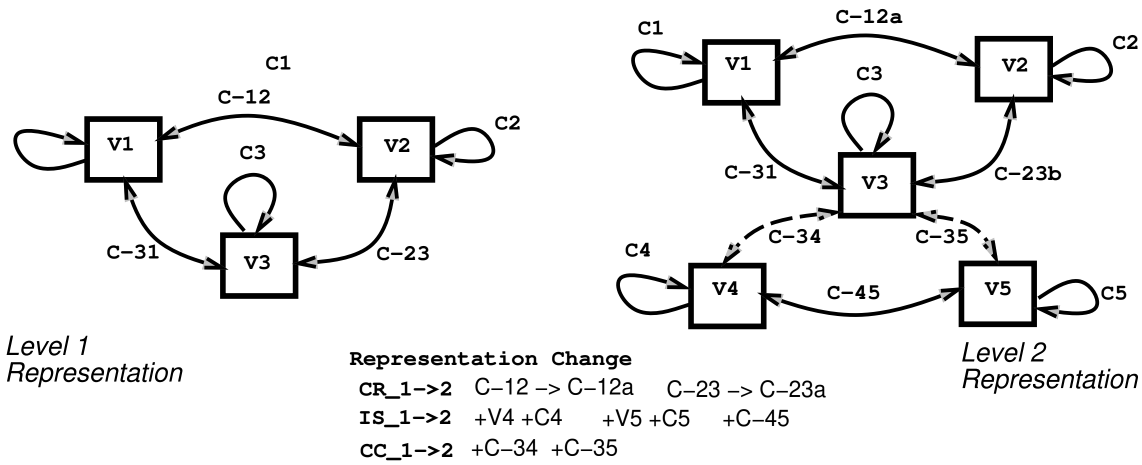

Figure 4: Representation Change between abstraction levels.

portant" nodes and constraints, and less abstract representations have more "detail", then an analyst will have to develop an abstraction hierarchy for the recognition of templates by first identifying a set of the most important or critical template aspects*, and forming one model using only these aspects. Next, the analyst will have to develop one or more subsequent models that successively incorporate more detail. Arranging these models such that the "least detailed" model is considered "most abstract" and the "most detailed" is considered "least abstract" gives rise to an abstraction hierarchy where a solution to the most detailed model can be viewed with the most confidence, while a solution to a less detailed model would have to be viewed with less confidence. If an abstraction hierarchy were created with three levels, we might consider labelling the first level "plausible", the second "supported", and the third "highly supported". Partial solutions within each level, could be ordered further according to other heuristic measures.

As part of the process of creating a hierarchical description of template recognition, the expert analyst must rank critical problem aspects according to importance. These problem aspects that need to be considered are:

1. Ordering Variables by "importance".

   In ordering variables, the intent will be that the abstract representation of a particular problem will include only those variables of most importance. At each successively more specific layer of the abstraction hierarchy, variables would be added to the representation.

2. Ordering Constraints by "importance".

   As for variables, the intent is that an abstract representation of a particular problem includes

*One example of military critical element identification that supports this model is *Signature Equipment* [8, 1], defined as "any item of equipment which reveals the type and nature of the unit or formation to which it belongs".

only the most important constraints. Clearly the definition of "important" constraints depends to some extent on which variables are applicable at each level. In fact there are two classes of constraints which differ fundamentally. Node or "reflexive" constraints affect only a single variable and as such may be ordered in terms of importance either with respect to a particular variable, or grouped by constraint type. Arc constraints affect a set of variables and may be categorized with respect to either that set of variables or by constraint type.

3. Relaxing Constraints.
   In addition to constraint ordering, constraints or groups of constraints may themselves have their restrictiveness softened.

After identifying the relative importance of these problem aspects, a problem representation hierarchy must be formed. Work has been done elsewhere towards automatically creating these hierarchies [3], however, here we consider only expert constructed hierarchies. Creating an appropriate hierarchical model is of great importance since the model will completely determine which, if any, candidate template instances will be found. A particular concern for the analyst lies in careful construction of abstract templates, since construction of a template in too strict a fashion will result in desired matches being missed.[†] Additionally, where we examine only those abstract solutions where key or important elements were identified, it is quite possible that a template containing "key" equipment or patterns has been missed despite its presence as a result of the key equipment not being sighted. In these cases, where some partial template could potentially be found, but where key equipment is missing, we can either fail to include such partial template occurrences or allow for more searching, possibly by using some preference heuristic that only "prefers" partial solutions with key elements over non-key partial solutions. The way in which we define both our abstraction hierarchies and our control strategy will determine the amount of flexibility with which we match templates. One specific method of creating a hierarchy based on importance factors is described in [12].

One immediate benefit of applying abstraction to CSPs is the ability to use hierarchies to represent problems in a way which will suggest a reasonable manner of solving problems in an approximate-first approach where the lack of a complete "least-abstract" representation solution will not leave the user without some "best-guess" near-solutions. Further, these near-solutions are arrived at systematically in that the user has implicitly indicated what constitutes progressively improving confidence for a particular problem domain. If we consider creating a hierarchical model as the process where a complex problem or object is successively simplified in specified ways, we see the creation of one path through the entire PCSP space as described in [12], from original CSP to least specific PCSP on the leaf. Figure 5 shows a PCSP space with a highlighted path showing how PCSP-5 was derived from the root CSP.

The PCSP space reflects all possible relaxations of the root CSP. A path from root to leaf indicates the set of relaxations undertaken to yield a certain PCSP. This path essentially defines an abstraction hierarchy for a particular problem, where the leaf node on the path is the most abstract, and the problem is made successively more detailed as we ascend towards the root. Since each successor in the PCSP space is found by applying one relaxation on its parent, the change in representation of the problem from one level to the next is exactly one operation. If we delineate certain points in the PCSP space as fundamentally changed problem representations, then representation change operators between levels may be formed into groups or sets. An abstraction hierarchy is therefore formed that maintains all of the representation change operators between root and leaf, while each group of these representation change operators fully describes the representation change between one abstract level and the next. Explicit use will be made of these representation change descriptions in our abstract search method, and it is interesting to note that many other abstraction-based search systems [13] fail to take advantage of this explicitly represented domain information, using only information implicit in the descriptions of each abstraction level.

If we consider the example template of Figure 1 as the most detailed or root model of our full PCSP space, we must identify the important or critical aspects as described earlier in order to create a hierarchy for the identification of this template. As an example, we can create an example most abstract version by identifying Slot 1 and Slot 3 as critical identifying features of the whole model. Crucial constraints between these are identified through consultation with the analyst. Additional detail is added to our model at the next abstract level, as Slot 2 and 5 are added, with additional constraints, and restrictions on earlier constraints. Finally, the most detailed model of Figure 1 is created with the addition of Slot 4, constraints connecting Slot 4 to the earlier Slots, and the replacement of earlier constraints with more restrictive ones.

[†] In a related work, Nadel [6] shows clearly that different CSP representations of the same domain have radically different solution characteristics, suggesting that future research is required in order to further guide expert construction of templates.

Search and interaction

We are interested in combining search and constraint propagation in a model that can support user interaction during search. Analysts may possess specific domain knowledge allowing them to remove some candidate partial solutions early in the search process, or identify which are most promising. Other work [12, 5] details approaches to solving CSPs based on "local search" with "repair heuristics" which take an initial assignment of elements to nodes and attempts to repair the instantiations until some acceptable measure is met. This type of approach has been shown to be effective when searching for one or several solutions, however, in the template recognition domain, we may be forced to find all solutions in our situation, and as a result we focus on complete, constructive approaches for the basis for our system.

Search in CSPs is basically a repeating process of selecting a variable to instantiate, selecting a domain value to assign to that variable such that the assignment is consistent with previous assignments, and continuing until all variables are assigned. Each complete assignment of domain values to variables is a solution. Essentially, if a problem P initially has N variables, each subsequent assignment results in a remaining problem P_1 with one less variable than its predecessor. We know that the process of finding all possible solutions is simpler if the domain sizes of each variable are smaller. Application of various constraint propagation techniques with varying effectiveness and cost could conceivably reduce each variable domain. Whether or not the reduced search achieved will exceed the work spent simplifying the problem is difficult to determine in advance, and more research is warranted into characterizing situations where this propagation work is justified, and in determining which propagation methods which would most effectively be applied.

Breadth-first search (BFS) strategies tend to explore the many domain assignments for a particular variable without preference, and all of these are considered before attempting to instantiate another variable. Depth-first search (DFS) approaches are more greedy, and attempt to find a complete assignment set as quickly as possibly, instantiating each variable once, iterating over domain values for a variable only when seeking an assignment consistent with those done before. Still other approaches have attempted to gain some of the benefit of each BFS and DFS by forcing BFS to "prefer" partial solutions closer to a complete solution. In this way the search progresses angle-wise through the search space with the left edge of an imaginary line showing the boundary of search tilted down to the left. In [11] such a strategy is described and is referred to as *Left Wedge* search (LWS).

In designing an interactive approach, we must be aware that since the analyst must identify and re-order or remove candidates that appear unpromising, a common reference frame is important for these candidates. Comparing candidates that have similar shared attributes is certainly more intuitive than comparing unrelated ones. For example, if an interactive approach were used with search, one would need to devise a strategy that carefully chose the order in which variables are instantiated prior to search. A changing of variable instantiation order throughout search would require the user to make judgements about groups of partial solutions that are "partial" in different ways. A constant, predetermined instantiation order of all variables, or of at least sets of variables would provide a better 'uniformity" over the elements of comparison.

Interactive BFS with a constant variable instantiation order would involve the analyst in removing candidate solutions early in the problem solving process, where few variables are yet instantiated. In addition, a BFS approach finds many partial solutions at a single "level" of detail first, saving them all before attempting to resolve ones at more detailed levels. This characteristic of BFS allows the analyst to remove candidates not to his/her liking as search progresses.

A LWS approach is simply "tilted" BFS, and as such retains the "storage" advantage of BFS, although elements are considered as a set not at a single level, but rather on the basis of some measure preferring more complete solutions at the expense of additional less complete ones. The tiltedness implies that elements under consideration at any given time will be of varying degrees of completeness, and so any interactive approach would require that the user make judgements between candidates of this type.

Depth-first search quickly locates more complete assignments, and provides the user with partial matches much closer to a complete solution more quickly, however, depth-first essentially thrusts towards a complete solution in a somewhat haphazard manner, and any interaction would require some stage or pause in search where the user may intervene and remove unlikely candidates or impose some order on the ongoing search.

We need a model that will provide us the ability to intermingle search, constraint propagation, and interaction in a flexible enough manner to accommodate changing or different users' needs. Abstraction offers this model, and in fact provides several possible solutions.

Abstraction and search

Search strategies can be either coherent across levels, considering all partial solutions at different abstraction levels according to the same measure, or they may consider partial solutions at different abstraction levels separately.

In a strategy such as LWS that was coherent across abstraction levels, it would be necessary to decide between locating additional partial solutions at a certain level, or pursuing some candidates at successive levels. This approach appears to be counter to the method of recognition employed by analysts. In a strategy that utilized different methods for search within an abstract level and search across levels, some set (possibly all) solutions at an abstract level would have to be found before solutions at subsequent levels are considered. In the case where all solutions at a particular level are found before advancing to the next level, if search were halted at an arbitrary point in time, all of the resulting "best-so-far" answers are on the same abstract level. However, time spent searching the less constrained CSPs might have resulted in more concrete solutions.

Search supported with non-hierarchical use of MOTORS is based on various depth-first strategies capable of finding one, some or all solutions. A unified hierarchical strategy might utilize a depth-first approach at each level, and a breadth-first approach across levels. The order in which abstract level solutions are considered between levels may be of importance to the analyst. For instance, if an analyst were searching a solution set for a particular solution, intervention in determining the order in which abstract solutions were examined could dramatically affect the search time required to find one particular solution or set of solutions.

Algorithm

Figure 6 details our interactive algorithm for pattern recognition based on abstraction. The following terms should be defined: *Rep_i* indicates the representation of the ith level of abstraction, where 1 is the most abstract, and N is the most specific; *Situation_Elements* indicates the set of individual sightings which are used to create the domain for each variable; $IS\_i \rightarrow j$ indicates the independent subproblems which result from the change of representation (Figure 4) between abstract level i and abstract level j; $CR\_i \rightarrow j$ indicates the constraint difference mapping between level i and level j; $CC\_i \rightarrow j$ indicates the new connective constraints which connect each $\mathrm{IS}\_i \rightarrow j$ to the level i partial solution.

Search commences with the most abstract representation, Rep_1. Initialization steps include initializing Domains for each Rep_1 variable according to each variable's characteristic constraints such as type or size, the representation change operations between each level (IS, CR, CC) are either identified or provided, and initial heuristic reduction of the Rep_1 problem by constraint propagation algorithms is performed. Search for all possible matches commences, using as input the ordered set of partial solutions from the previous abstract level (initially only Rep_1). A backtracking strategy is heuristically selected, and search commences. Use of heuristics during search within a level is discussed in detail for each option in [10]. During backtracking search, heuristics are utilized at Step 9 to select the next candidate node for further instantiation, at Step 11 to select a variable within a candidate node to instantiate, at Step 13 to determine how much (if any) constraint propagation is done to further simplify the remaining problem aspects based on the selection in 11, and at Step 14 to determine if any of the reductions performed at 13 should be propagated upward in the search space.

As the depth first strategy locates solutions to this abstract representation, these solutions are passed to the interactive *This_Level_Revise* process where the analyst can visualize and analyze the solutions on a workstation making use of other tools, and can then re-order or remove them according to his/her own criteria. Configurable critic heuristics can preview solutions, prior to handing them to the analyst, to filter out undesirable solutions according to analyst-indicated preferences. Duplicate sightings in close proximity can result in repeated solutions where the only difference is in a certain variable assignment. Further work is indicated in determining the spatial and other characteristics that indicate the "sameness" of these elements. However, this identification of nearby variable matches as part of a same-template instance can assist in the analyst's fusion task of these duplicate elements.

The analyst can allow the parallel search and review process to continue as long as desired, or until all abstract solutions are located. Once satisfied, the selected abstract solutions are passed to the next level for further refinement, and the process repeats for the next solutions. At the refinement stage, the representation change operators are applied, first pruning all partial solutions which fail to accommodate the constraint restriction between levels, second, initializing the independent subproblem variables according to variable characteristics, and third, extending the abstract solutions to less abstract problems via addition of the independent subproblems. The domains of the now attached independent subproblems can be reduced according to local heuristics, such as particular spatial constraints restricting the domain of the template based on the abstract solution in question, by selectively applying the connective constraints between abstract solution and independent subproblem, and by some heuristically selected constraint propagation method. Once simplified in this manner, all remaining problems are passed as input to a new interactive search phase. This process will continue until the analyst is satisfied with the results, or until all so-

```
Abstract_Interaction( Rep_1 ... Rep_N , Situation_Elements )

1 Create domains for Rep_1 variables, consistent with node constraints
2 For all level changes i->j
  3 Identify IS_i->j, Independent Subproblems added between level i and j
  4 Identify CR_i->j, Constraint Restrictions on level i constr at level j
  5 Identify CC_i->j, Connective  Constraints between level i and all IS_i->j

6 Reduce domains of Rep_1 variables via selected constraint propagation

// Search for solutions at level i, initially Rep_1
7 For each unsolved problem at level i,Select backtracking strategy,heuristics

   8 Loop unless User_Interrupts
        Interrupt: save backtrack state position, exit Loop.
     9  Select backtracking node for expansion
     10 If Variables_Left to assign
        11 Select variable for instantiation
        12 Create successor from domain values for selected variable
        13 Propagate constraints using selected domain value
        14 Upward propagate if possible
        15 This_Level_Revise( solution )
     16 Loop (8) till done backtracking.

// Dynamic user interaction
This_Level_Revise( solutions )

  17 Delete solution if critic heuristic fails
  18 Delete solution if duplicate from noise
  19 Order  solution by ordering heuristic

  20 Analyst displays, removes solution(s) as required
  21 Re-order solutions according to analyst preference

  22 Repeat This_Level_Revise
        until User_Advance_Next_Level
     or until User_Select_Previous_Saved_Backtrack_Postition.

// Descend Abstraction Hierarchy i->j
  23 Applying CR_i->j, pruning all level i solutions that fail.
  24 Assign domains to IS_i->j variables, consistent with node constraints
  25 Extend all level i solutions to level j problems by adding IS_i->j

  26 For all j extended level i solutions
     // Local Consistency enforcement
     27 Simplify IS_i->j variable domains by applying some spatial constraints
     28 Simplify IS_i_>j variable domains by applying some CC_i->j
     29 Simplify IS_i->j variable domains by selected constraint propagation

  29 Repeat Search for each level j problem until j > N or USER_INTERRUPTS

  30 When j = N, remaining solution set is returned
```

Figure 6: Abstracting Interactive Search Algorithm.

lutions are found at the most specific representation level.

Final solutions can be utilized by the analyst and accepted into the situational view, thus relating identified template elements. Partial solutions can be instantiated also, with the elements not identified inserted into the situational view as expected but not sighted representations. The potential ranges of these unsighted elements can be restricted according to existing constraints with the identified elements, and according to other available information such as terrain, and analyst preference. The existence of these elements, while hypothetical, is supported by the analyst's acceptance of a suggested template. Further work in analyzing and dealing with analyst's expectations can be found in [9].

The role of abstraction in interaction

The human intuition behind using abstract problem representation and solution strategies is that in complex situations, it is often easier to attempt to "rough-out" solutions considering only the most important issues first, and then attempt to refine one of the "rough" solutions into a usable solution. We have attempted to design an interactive system of this type with a model of recognition familiar to the user.

Just as in human problem solving, a user can quickly glance over "high level" options and discard those which violate some constraint or condition not represented explicitly in a domain. Since any domain model is only approximate, we make use of abstraction guided interaction to assist the user in recognizing partial solutions that yet fail to resolve at a more detailed level as a result of an incorrectly or too strictly defined model.

We have attempted to create a model of interaction which demonstrates the following characteristics:

1. A user may be content at some point with some subset of all possible abstract solutions, may explore these at more detailed levels, and may wish later to attempt to locate additional abstract solutions.
2. The solutions at each level of abstraction should be found as quickly as possible in order to offer this level of solution as quickly as possible to the user as user consideration time may exceed computation time.
3. The user may re-order or remove solutions at each abstract level before search proceeds at a lower level. Solutions should be presented as discovered for consideration at a single level while others are still being found.
4. The user should be capable of interrupting or postponing the ongoing search at a single level and indicate "enough" solutions have already been found at this level, and search should then proceed on refining those found at the next more specific level. The interruption can indicate that the time available for searching has ended, and approximate answers are required immediately.
5. Search at each level should proceed in a depth-first but complete manner, finding this-level solutions as quickly as possible for consideration. Each member of the solution set at this-level is a candidate to be refined at the next level.

The development of an interactive system using abstract or hierarchical descriptions of the recognition process may be seen as an attempt to formalize part of expert template recognition. The use of hierarchies provides a method of expression which is capable of capturing domain information not represented in a non-hierarchical representation, and naturally allows for the integration of certain domain concerns such as signature equipment. The application of spatial heuristics for restricting the domains of partial solution problem extensions across hierarchical levels acts as a method of restricting possible specializations of abstract solutions in a fashion similar to the way an analyst limits his/her focus of attention.

Further investigation of the analyst's ordering and rejection of candidate solutions between abstraction levels may provide insight into future solution ordering or rejection heuristics. Analyst confidence in any suggestion oriented system is dependent on the ability of the system to justify suggestions in familiar terms. In the case where a partial solution is suggested in the absence of either affirming or denying indicators, the hierarchical resolution process can provide information about what specific information might be gathered in order to reduce the number of possible explanations of current sightings, or to increase the confidence level to the next defined hierarchical definition. Since the structure which determines the change of representation between an abstract level and the more specific is fully described, the process of explaining why a certain partial match qualified or failed to qualify at the next level is a simple matter of noting which additional constraint eliminated the solution.

Noise	Whole	Abstract
100	0.13	0.28
200	0.26	0.67
300	0.63	1.65
400	1.46	4.45
500	1.98	7.41
600	2.19	14.18
700	4.45	17.08
800	5.0	28.04
900	5.85	27.97
1000	9.6	45.51

Table 1: Maximum search time, two template models.

Experiments

Currently, all experiments have been undertaken with a limited implementation of the algorithm described which only functions at a single level of abstraction. However, this limited implementation representing steps 8 through 16 of our algorithm of Figure 6, has provided us the opportunity to explore several backtracking options, specific spatial heuristics, and varying degrees of constraint propagation before and during search. While detailed discussion of these results is left to a later publication and is described further in [10], Table 1 shows the CPU search time required to locate all occurrences of the template described in Figure 1, and an abstracted version of 4 Slots described in Section 3, in varying amounts of sighting noise. Each entry in Table 1 represents the worst CPU performance in 10 experiments. The "Whole" template had one occurrence placed in the data, while the "Abstracted" template had two occurrences placed in the data. Random noise resulted in none or 1 spurious solution for "Whole", and as many as 6 for the "Abstract".

We can conclude from this simple experiment that we must carefully determine the nature of our most abstract template so as to not overburden ourselves with solutions. Care must be taken that the elements of the most abstract template truly represent "critical" elements. Use of abstract templates which are under-constrained can result in far too many partial solutions at early search stages. One important aspect of our future work is to attempt to better characterize actual situational information to determine distribution and frequency of various sightings more accurately.

Conclusion

In this paper we have described a model for interactively recognizing spatial deployment templates in noisy and complex situational information. This model of interaction with the expert analyst is based on increasing confidence over time in the veracity of certain identified template instances. The ability of an implemented non-interactive system to provide single-level solutions in a reasonable time frame suggests that a strategy of most-abstract-solutions-first can be undertaken in a limited and predictable period of time.

Acknowledgements

Thanks to Dr. Qiang Yang of the University of Waterloo for providing comments on early aspects of this work, and for providing a skeleton lisp implementation of simple CSP search. Thanks also to Luc Lamontagne and Stephane Lapointe of DREV and to Kirsten Wehner for providing valuable comments.

References

[1] L. Des Groseilliers. Tactical information fusion prototype: Basic intelligence. Memorandum 3093/92, Defense Research Establishment Valcartier, Valcartier, Quebec, February 1992. Unclassified.

[2] E. Freuder and J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, December 1992.

[3] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[4] Vipin Kumar. Algorithms for constraint-satisfaction problems. *AI Magazine*, pages 32–44, Spring 1992.

[5] Steven Minton, Mark Johnston, Andrew Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[6] Bernard Nadel. Representation selection for constraint satisfaction: a case study using n-queens. *IEEE Expert*, pages 16–23, June 1990.

[7] Bernard A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[8] NATO. Nato glossary of terms and definitions. Technical Report AAP-6(R), NATO, 1988. Unclassified.

[9] Graham Williams and Steven Woods. Representing expectations in spatial information systems: A case study. In *Proceedings of the 3rd International Conference on Large Spatial Databases*, June 1993.

[10] Steven Woods. Interactive recognition of spatially defined model deployment templates. Research memorandum, Combat Intelligence Automation Group, Defence Research Establishment Valcartier, Courcelette, Quebec CANADA, August 1993. unclassified.

[11] Steven G. Woods. An implementation and evaluation of a hierarchical non-linear planner. Technical report cs-91-17, Computer Science Department, University of Waterloo, 1991.

[12] Qiang Yang and Philip Fong. Solving partial constraint satisfaction problems using local search and abstraction. Technical Report cs-92-50, University of Waterloo, 1992.

[13] Qiang Yang, Josh Tenenberg, and Steven Woods. Abstraction in nonlinear planning. Technical Report cs-91-65, University of Waterloo, 1992.