Identifying spatially defined model representation templates in large data sets using MOTORS

Steven Woods[1]

[1]Defense Research Establishment Valcartier, Courcelette, Quebec, Canada, G1R 2L5

Abstract

A template is defined as an expected configuration of (possibly loosely) spatially related objects. Each object in a template can have constraints which describe it. In an object-rich environment it can be difficult to recognize these templates as a result of clutter, or as a result of the complex interactions of many object constraints. The problem becomes even more difficult if we consider many templates, or a dynamic environment. Our earlier work in this domain focused on providing a set of tools which would enable a human to orient a template defined in terms of a set of expectations upon a spatial workspace and then to be notified if a certain degree of correspondence was detected between the template components and the environment. A method requiring less human interaction would be desirable, particularly in cases where many templates could exist or co-exist. This paper is intended as an evaluation of the applicability of known constraint satisfaction and model pattern recognition methods to the solution of the template recognition problem. The system MOTOR (MOdel Template Object Recognition system) presented in this paper has been constructed by merging constraint propagation techniques with search strategies, and controlled via an interactive preference structure based on abstraction.

Contents

List of Tables

List of Figures

Chapter 1

Executive Summary

1.1 Project Identification

During January in conjunction with the members of my group and group leader Marlene Gauvin, I identified "Recognition of Spatial Deployment Templates from Temporal Data" as the preliminary topic for my research during my time at DREV. The following is an extract of this work proposal of February 3, 1993.

> "A deployment template represents information about a particular, pre-determined, set of relative unit positions. Each element of a template has a particular (ideal) constitution and relationship (spatial or otherwise) with the other units of the template. One method intelligence analysts use in the effort to construct a better "hypothesis" of an enemy situation is recognition of these common patterns or templates. The temporal nature of the messages describing sightings further complicates the analyst's task. I propose to investigate the possibility of implementing a decision aid subsystem that would be capable of assisting an intelligence analyst recognize the existence of these templates by examining the past positions, past movements, and constitution of units described in intelligence messages. Of particular interest in my research will be the movement of sets of units over time and in determining whether or not any pattern can be detected indicating evidence for or against a particular template or set of templates.

1.2 Work Summary

Initially, I researched existing DREV documents and made use of DREV personnel with knowledge of templates in order to form a better model in my mind of what the "essential" elements of a doctrinal template were in the view of an intelligence analyst. The result of this research is captured in the simplified model of templates I make use of throughout this document. Essentially I adopted a constraint based system where individual enemy elements are composed into a template of a particular type based on inter-element relationships of various types.

As the next major step of my preliminary work, I undertook a literature survey across several broad areas searching for models or discussions of systems of a similar nature. In particular I was interested in determining what approach to use in finding a practical, implementable system which could deliver the minimum requirements of serving as a useful tool for the analyst in the pursuit of identifying templates. I focused primarily on military related materials and conference proceedings for specific mention of this problem type, on prominent artificial intelligence publications for discussion of practical applications of modelling and reasoning with constraints in this fashion, and as well attempted to broaden my outlook by examining several alternative solution approaches such as massively parallel model recognition which demonstrates similarity in a restricted manner. Each of these tasks is discussed later in this paper, however, I was unable to find published discussion of similar work or implementations. Published documentation on constraint satisfaction implementation lead me to believe that a combined search and constraint propagation approach could solve the problem if a model could be found which appropriately handled analyst interaction in the search process.

At this time I established several major goals that I believed would be critical in the development of a working subsystem. These goals are listed below, along with a comment summarizing the state of the work toward these each goal.

1. Restrict the scope of the entire problem statement even further so as to manageable handle a static model first, before attempting to address the more complex temporal aspects. To this end, identify the critical assumptions and processes that must be made to time-standardize the message data.

   I have assumed throughout my work a "snapshot" model of a particular point in time. Templates as conceived by the military currently represent a picture of unit deployment at one time point. Identification of these unit patterns in message data is itself a difficult problem, and a strategy towards this static identification needs to be understood more fully before a temporal approach could be attempted. 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. In attempting to organize hundreds of these messages at some still later time T4 into a coherent picture of the world at T4, the analyst faces a daunting task. Any perception of the world at T1 decays over time is such a dynamic situation, and as such, sightings lose their information value over time. 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 will be considered to be the input to our system, essentially a blurry snapshot of a moment in the past.

2. Identify major limiting assumptions about the nature of the incoming message traffic itself so as to better understand the complexity of the problem. In particular, the spatial scope of messages needs to be better characterized in terms of the analyst's own perceptions, and "typical" volume and distribution values need to be determined.

   Since one major aspect of the constraint satisfaction problem of template recognition is spatial relationships of sighted units, it is necessary to characterize what locational information is considered implicit in a sighting message for some particular unit with type, size and other attributes. During May of 1993 I was assigned a summer student, and after investing much time explaining the problem itself to him, I instructed him to visit Valcartier directly and discuss the problem with an intelligence officer. Some progress was made in characterizing the scope of certain messages, and these results are captured in this paper. However, more work is required in this area, in particular in characterizing the way in which the analysts themselves perceive and use the information they receive.

   Volume and distribution of information is very important and in fact represents "noise" that confuses the situation and makes the discovery of template instances difficult. While both myself and my summer student have spent time examining the sample situation model present in the TIF demo, I believe that further examination of real situation message data is necessary to better characterize "typical" message distribution in terms of the location and type of sightings. Throughout the course of our experiments we have assumed largely random distributions of message sightings over the field of focus.

3. Model the problem solving subsystem as a stand-alone, non-interactive tool which could identify the templates in "typical" data. From this experiment, determine the deficiencies and highlight the value of the analyst experience in solving these problems with the intent of capturing the analyst's function in the complete, interactive model.

   We have modelled stand-alone doctrinal deployment template recognition as a *constraint satisfaction* problem (CSP). In this paper a toolset is described called MOTORS (MOdel Template Object Recognition System)which provides for the solution of CSPs utilizing a configurable hybrid search and constraint propagation strategy. Some very promising results have been obtained by experimenting with various heuristics and search strategies, and in fact the nature of spatial templates as modelled has allowed for discovery of all instances of a particular example template in very noisy data in only a few seconds. Specific results are detailed later in this paper.

4. From the stand-alone model, identify problem-dependent heuristics that may simplify the problem solution. In addition, attempt to identify problem characteristics that will suggest what type or mode of hybrid is appropriate. In all constraint satisfaction approaches, a trade-off must be made between search and constraint propagation

techniques during search, and the problem properties which might indicate the balance need to be examined. Earlier work in simplistic domains has suggested several possible approaches, but no work has yet been published with so complex a problem as template recognition.

Throughout this paper, work done by myself and by my summer student is presented which outlines the various approaches and tradeoffs. Implemented versions of several possible algorithms are available and may be easily integrated into the MOTORS stand-alone prototype. Additionally, some experiments have been carried out with spatial heuristics in an effort to restrict the size of the search space based on the concept of template-instance locality within the body of message data.

5. Develop a model accommodating the application of the analyst's experience and expertise together with the constraint reasoning approach of the stand-alone tool. Pay particular attention to forming a model that is capable of flexibly shifting between interaction and stand-alone so as to better suit a particular analyst's changing needs and available time. A best-so-far approach is required that will give some solution "anytime" if the analyst needs an answer immediately.

   In [69], a model has been developed which will allow for the user interaction in the search process for templates. This model is based upon the conception of expressing templates in terms of "key" or "critical" elements and additional "supporting" evidence which lends credence to the acceptance of a set of enemy sightings as a template instance. The model accounts for making explicit use of the new problem specification and change of representation information, is capable of providing best-so-far answers where unlimited time is unavailable, and has a configurable amount of user interaction.

6. Implement the full model as a prototype independent of the Tactical Information Fusion (TIF) system. In this way, the TIF experts can better identify exact application technology or interface requirements and take advantage of TIF specific data structures and implementation quirks of which I am not aware and cannot expect to master in so short a time period.

While the stand-alone MOTORS system is written in LISP and is fully operational, the interactive model prototype has not yet been implemented.

1.3 Future Work

Much of the implementation and experimentation work I have undertaken has been towards developing interesting constraint satisfaction algorithms which function as hybrids of search and constraint propagation. Examination of how to best dynamically carry out this search is not well understood, even for simpler problems, however, our implementation has suggested that model template recognition is restricted enough to allow for reasonable time solutions.

The model presented for interactive template recognition is consistent with other ongoing efforts in abstract constraint satisfaction work aimed at developing scheduling and planning systems elsewhere [71, 72]. I believe that the model presented in [69] can be utilized to create an interesting, useful and functional decision aid tool for application in an intelligence collation system such as TIF, and would provide a flexible testbed for new heuristics, propagation methods and search strategies.

There are many related interesting research issues that deserve attention. Problem-specific issues include:

- Message data characterization, particularly in terms of spatial distribution, and data point "resolution" indicating the area of possible location of a sighted element.
- Message volume characterization, and the time-range of data considered in template recognition must be determined more precisely.
- Integration of additional information such as terrain can significantly reduce the complexity of locating template instances by restricting even further the possible locations for enemy units.
- Extension of the model for template recognition to include temporal message values and extended fused data (such as tracks and merges).
- Make use of partial template matches supported by the model to identify probable or possible locations of unsighted enemy units based on acceptance of proposed templates. This application was discussed at some length at the TIF demo in Ottawa and was universally seen by the participants as a valuable additional tool. Further work needs to be undertaken to better characterize how the analysts themselves view these hypotheses in terms of likelihood and acceptance.

Additionally, constraint satisfaction provides a rich environment for future domain independent work which can significantly improve solution time for template recognition as well as other problems such as scheduling and planning:

- Template recognition is a very spatial problem, and more though must be given to how to best take advantage of this spatial nature in terms of limiting the scope of search for certain templates. In fact, all constraint satisfaction problems of a spatial nature can benefit from new approaches to propagation using spatial heuristics.
- Further expand upon the hybrid model so as to better understand where to make use of propagation effort and where to take advantage of search of certain types. "Smarter" hybrid search strategies such as that shown in Chapter 5 can make significant improvements in overall performance.

- Model constraint satisfaction so as to take advantage of parallel implementations of some aspects of constraint propagation where applicable.

- The tradeoffs between finding "one", "some" and "all" solutions are many, particularly in an interactive system. While we have adopted the "all" approach with an eye towards providing them each as quickly as possibly (one at a time), other slightly different requirements would suggest a different approach altogether. For instance, a scheduling problem requires only one solution of a certain "goodness". Similar work can be applied to template recognition in attempting to find a template instance that has a certain degree of "confidence".

- Expression of constraint satisfaction problems in terms of abstraction hierarchies with explicit representation change information as presented in our model suggests further work in profitably applying these representation changes during search. It is well known that problem representation has a direct and dramatic affect on search performance, and this new representation is only understood at a very high level currently. Other representations may exist for template recognition and other constraint satisfaction problems that suggest modified or entirely new solution approaches.

1.4 Looking Ahead

Commencing September, 1993 I will be commencing my Ph.D. work under the supervision of Dr. Qiang Yang at the University of Waterloo as a member of the Logic Programming and Artificial Intelligence Group. The exact nature and topic of my Ph.D. thesis is not completely determined at this time, however, I have agreed in principle with Dr. Yang that the problem of military doctrinal template recognition is sufficiently interesting and complex to base a thesis upon. I intend to focus on some aspects of these spatial and temporal constraint satisfaction problems and various solution possibilities. There are many domains for these problems, and the template recognition one is a rich and interesting one with good clear application possibilities.

Chapter 2

Model Template Recognition (MTR)

2.1 Introduction

In our previous work [68], we presented a system for allowing a user to dynamically record his/her reasoning process about the configurations of military units in the field. Of particular note in this work was the ability to place spatial EXPECTATIONS on the field. Each EXPECTATION actively checks existing or new data items to see if the EXPECTATION is met. When an EXPECTATION is met, the analyst is notified, and can integrate this new information into his/her current world view as he/she chooses. Essentially this system provided an automatic highlighting system for bringing future information or events previously determined to be "important" to the attention of the analyst. A simple extension of this capability would be to relate a set of EXPECTATIONS together into a TEMPLATE, and then define a matching procedure for a TEMPLATE based on integrating each of its individual EXPECTATION matches. Thus the analyst is able to place a template upon the field of battle simply as a set of EXPECTATIONS, where the EXPECTATIONS are related with one another according to spatial or other constraints as desired by the analyst.

The object we are referring to here as a template is essentially a group of potential sightings arranged according to a model which defines the way in which the enemy deploys the units within that particular template. The template we speak of could potentially exist at almost any level of detail - for example a regimental template would consist mostly of battalion elements, while a batallion template would consist primarily of company elements. A template specifies how lower level elements are arranged spatially according to the doctrine of the higher formation.

We make the assumption that the enemy deploys his elements according to his own templates, and therefore, if we can take the individual element sightings and "recognize" the pattern constituting a particular template, we will have gone a long way towards understanding the enemy intent, and quite possibly being able to predict the presence of units we have not sighted on the basis of unfilled template elements.

Similarly, we could hypothesize the existence of a particular template in the situation at a specific point according to our own analysis, and then essentially "drop" the related set of expectations into the situation according to the template relationships. In such a framework, the process of recognizing an entire template is one of first matching each EXPECTATION against the field of battle environment and basically matching one (or possibly more than one) of the military units in the environment to correspond with each EXPECTATION in the template. Care must be taken that during the matching process that the constraints are not violated. If we find an assignment of a military unit to each EXPECTATION to the satisfaction of the restrictions imposed by the TEMPLATE, than we have a successful template match. Other issues will arise in performing such a matching process, including dealing appropriately with multiple competing assignments to individual EXPECTATIONS, or handling cases where constraints are only partially or incompletely satisfied by a certain set of assignments.

A problem arises in that to hypothesize the presence of a template may be difficult given the confused or sparse situational information in the first place. In this type of situation we would be more interested in actually attempting to locate a template or set of templates in the confused world situation from scratch, possibly with analyst assistance during the searching process. This problem is related to the work described above in that automatic recognition of the existence of a template may suggest a match that is incomplete or only partially satisfied the problem description. If some partial solution were determined to have a measure of certainty deemed "satisfactory" or "high enough", this solution could be used a strong basis for inferring the existence of the elements that would "complete" the template match. In [11], we are told:

> Doctrinal enemy deployments are used as reference material by the (intelligence) analyst to help identify possible enemy units.

Essentially we may now make use of expectations in order to "fill in the holes" of a matched template. This approach is the one that this paper will focus on primarily.

Figure 2.2 on page 20 shows a representation of a simple example military template. This template shows a set of military units arranged spatially in a particular formation. For simplicity and clarity I shall attempt to refrain from excessive use of specific military terminology where possible.

A template will consist of a set of constituent units (or slots) of a common magnitude or scale, and also a set of relationships among the slots. Each of the template's constituent units shown may possess certain attributes that describe that unit's make-up or activity. For the purpose of this template construction example, we describe each unit as possessing individual attributes type, orientation, activity, and optionally an absolute location range. The inter-unit relationships in this example will include minimum and maximum spatial separation, the spatial relationships left-of, right-of, ahead-of, behind-of, a construct defining "echelons" which are essentially allowable corridors of presence for units not at the front of movement or position, and same-attribute relationships which indicate that a given pair of units must have matching individual unit attributes of a particular type. Many other types of attributes or relationships may be defined, but these will serve for the purposes of this example.

In [22], pages 4-34, we are told:

> The standard components of a unit are put together in specific patterns representing a basic method for organizing a division. The doctrinal standard, however, is not inviolate.

It is important that we realize that a template as defined in military doctrine is only "typical" of the deployment patterns, and does not represent an inviolable set of rules. As a result we will wish to incorporate flexibility in both the specification of templates, and also in the process of matching these templates to situations. Additionally, it is important that an analyst using a tool that performs this matching is able to dynamically adjust or specialize the search for a template.

As we have stated, the analyst will want to specialize typical templates at times. Of interest will be the application of templates to formations that have additional intelligence known, such as commander history. In [22] we are told:

> ... doctrinal patterns are modified based on the actual deployment of specific units. The enemy unit's history, training, and key personalities information from the data base are assessed to determine how the unit operates differently from (standard) doctrine."

Of concern in producing specific templates from general templates are a number of other factors, including specific unit behaviour, specific information about analysis of the area of influence and interest, and weather and terrain. Essentially once an enemy position is identified, the template is "stretched" by the analyst so that it accommodates the specific occurrence of these units to fit terrain and other factors. The key is that these "specific" templates can be used to track known existing enemy positions more accurately.

In our primary focus of identifying the existence of templates based on enemy element identification through messages, the analyst basically plots these message sightings *before* the template recognizer is given access to the message data. In such a methodology, the situational message data has already been "fit" to the battlefield and terrain as part of the intelligence analyst's task on the basis of only local locational and attribute information. Since this information has already been terrain-fit, we may attempt to fit a template without immense consideration of possible template modifications in advance. If we have defined an adequate "flexibility" into our template specification, it should "fit" the message data. If not, we must be able to allow the analyst to recognize promising partial fits which simply fall outside the complete "typical" template specification we have originally given.

The common military view of what constitutes a spatially represented doctrinal template has been obtained though discussion with intelligence officers, as well as by referring to military-oriented documents [21, 13, 11, 30, 12, 29, 55, 56, 2, 14, 1, 57].

Some of the key questions that identify the attributes represented in templates are outlined in [22]:

- What is the standard doctrinal pattern for an element in attack ?
- How much spacing is there between echelons ?
- What are doctrinal rates of movement ?
- What frontages and depths can we be expected to see ?

The 'static" nature of these templates (often they are physically made of plastic and overlaid directly on real battle maps), indicates that they are only an approximation and only represent a "snapshot" at a single moment in time. In using them, the analyst is always aware that they only represent an approximation, and further that the approximation may be "stretched" far since the actual relationships being modelled are much more complex than can be accommodated in a static, fixed template of this type. Templates and their specification will be discussed further later in the paper.

Frequently in discussing templates, reference is made to inter-element distance constraints (minimum and maximum). These types of flexiblet inter-element constraints suggests a model that is dramatically different than the typical "plastic" overlay might suggest. If we are representing the relative positions of three elements where each element has an inter-element range with the other elements, it is not possible to draw a uniformly scalable template representation that can be simply overlaid on a map in an attempt to recognize a template instance. The regions in which each element possibly exist are in fact difficult to describe simply even given two of three absolute positions. The third position could be determined by intersecting arc of range from the known elements, giving an area bounded by four curves[1]. This curved area would have been of different shape and size given different acceptable locations of the first two elements. Essentially the positions of the elements are heavily interdependent. A more flexible model is required for identification of templates with these types of spatial relationships.

2.2 Template representation and use

Template recognition is an important and difficult part of the intelligence analyst's task. Spatial relationships between sighted enemy elements can be matched with known doctrinal template definitions indicating the presence of a particular higher enemy activity formation. A "sufficient" accepted match of a known template suggests "holes" in the current situation understanding where previously unsighted enemy elements possibly exist.

Figure 2.2 shows an example of a doctrinal template as often depicted in intelligence analysis documents. Spatial relationships are inherent and only vaguely specified to some extent. Distance ranges are often given for echelons or partitions, whereas inter-element relationships such as "in-front-of" or "to-the-left-of" are depicted only roughly. In Figure 2.3

[1]Note that this bounding can be problematic also. This problem is discussed on page 126.

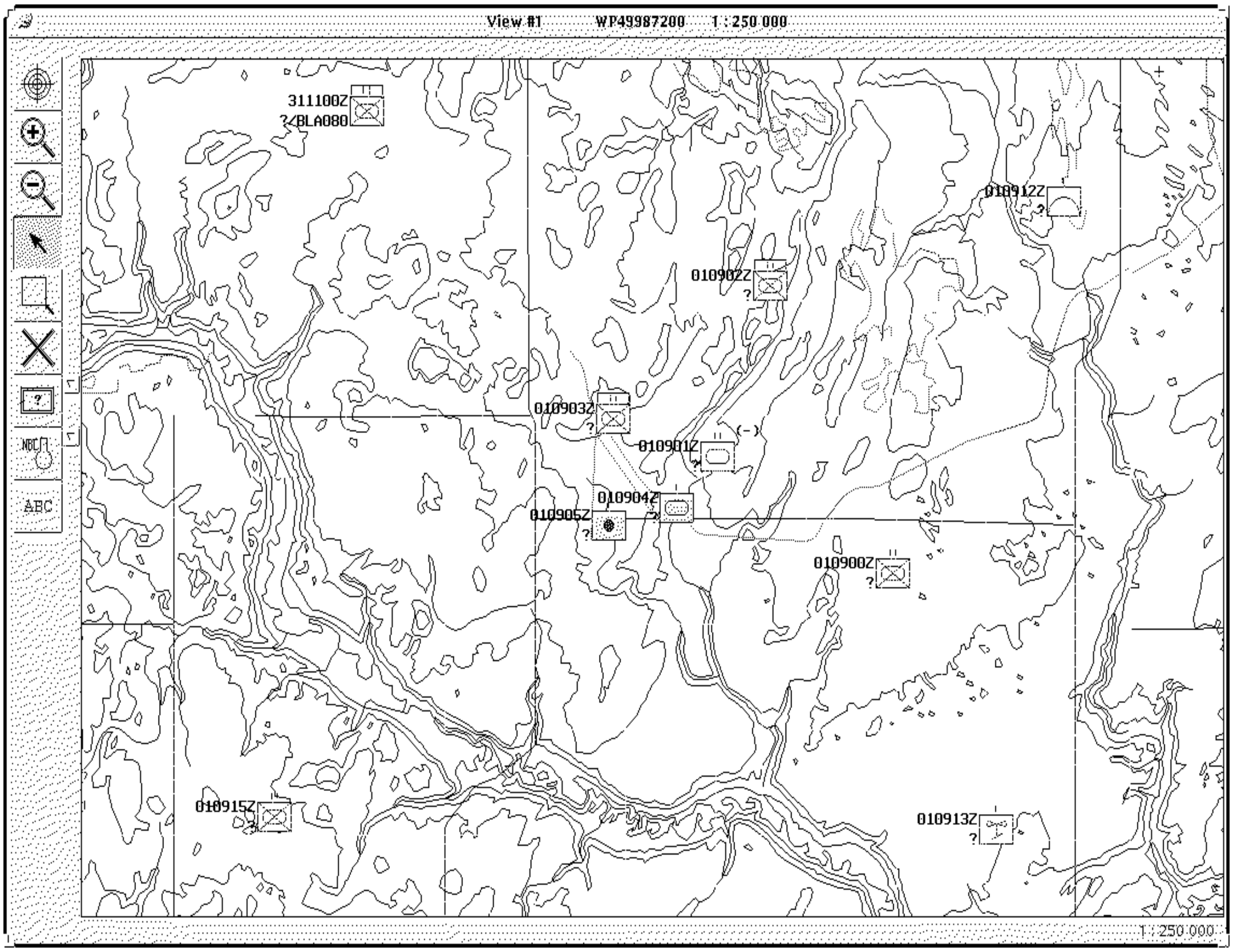


Figure 2.1: A deployment situation as seen in TIF.

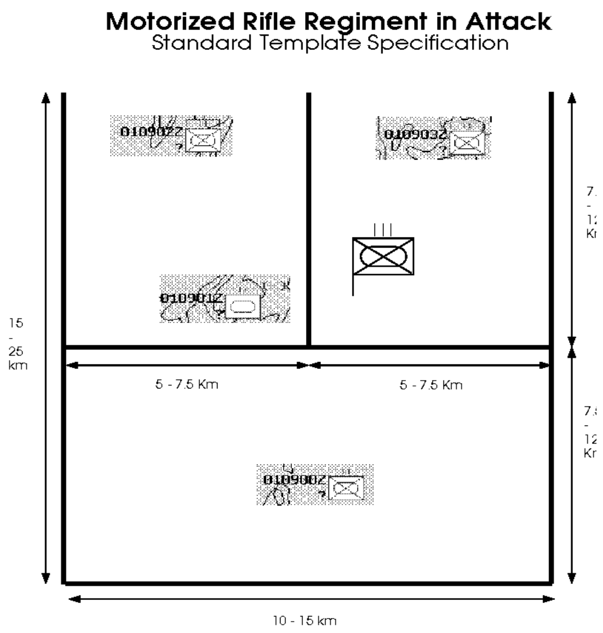


Figure 2.2: Standard deployment template

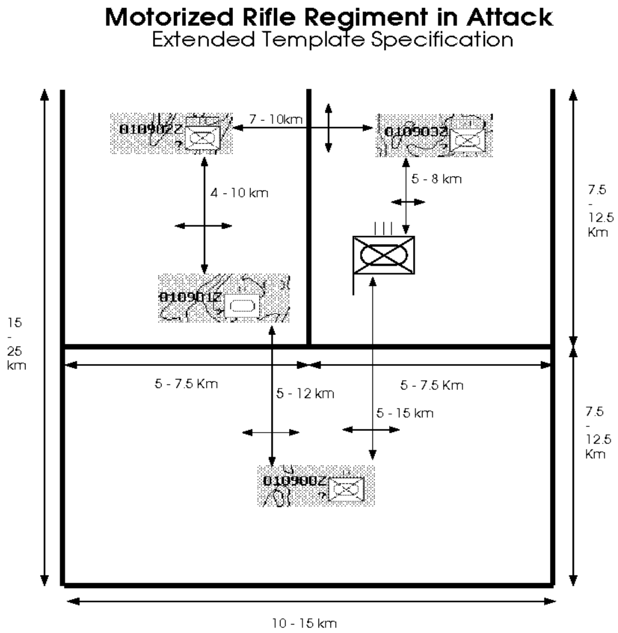


Figure 2.3: Inter-element range Template

we show the same template with additional annotation making the inter-element relationships more explicit. The example constraints shown by the arrowhead lines indicate example relationships such as "ahead/behind", "left/right" and "separation".

In fact, we can capture the entire template model in terms of a set of relationships between template elements. Figure 2.4 shows an alternate template model in terms of only inter-element relationships. Figure 2.5 shows the template reduced to a set of slots which the analyst will wish to "fit" in the current situational view.

A template defined in terms of these spatial relationships has many possible "instances". For example, the constraints could be "stretched" to their maximum extents giving a template such as shown in Figure 2.6, or "compressed" to their minimum extents giving a template such as shown in Figure 2.7.

Figures 2.8 and 2.9 show an instance of this set of template constraints that "fits" the known situation. One interesting decision aid problem is in determining a valid approach for automatically finding such "fits" as suggestions for the analyst to consider.

The "fit" shown in Figure 2.9 actually matches only 4 of 5 template slots with situational elements. If the analyst "accepts" this template as a hypothetical explanation of the situation, then we can "expect" that a situation element exists that would fill the slot as shown in Figure 2.10. The predicted location of this "expected" element can be suggested by utilizing the doctrinal template information such as inter-element distance shown in Figure 2.11.

Figure 2.12 shows the overlap of two inter-element distance constraints which combine to

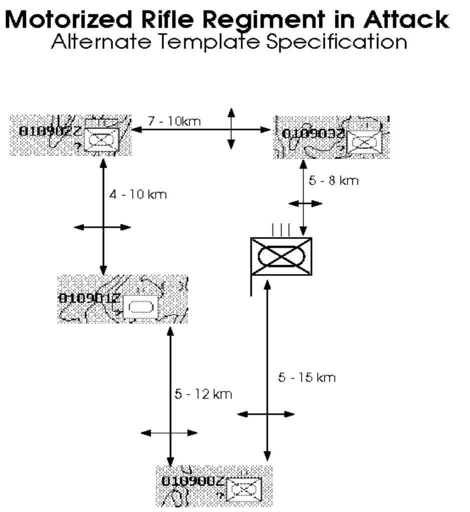


Figure 2.4: Simplified Inter-element range

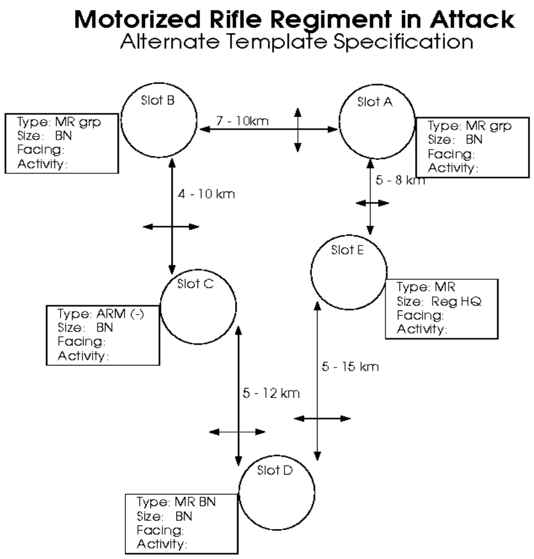


Figure 2.5: Template as a set of slots

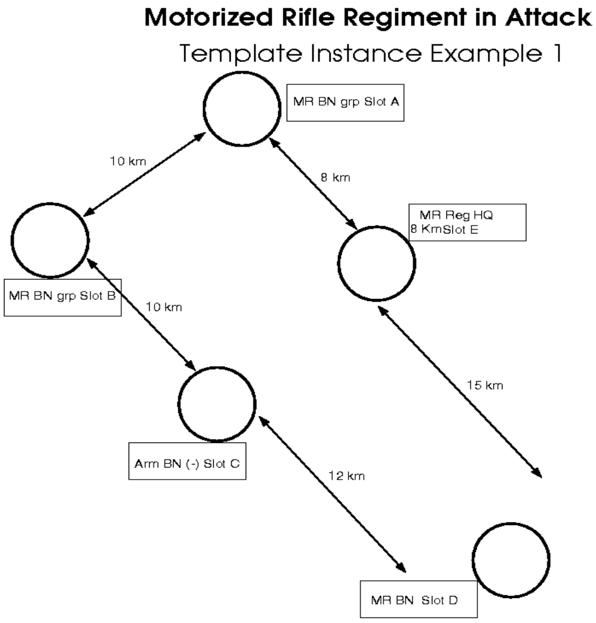


Figure 2.6: A extended template instance

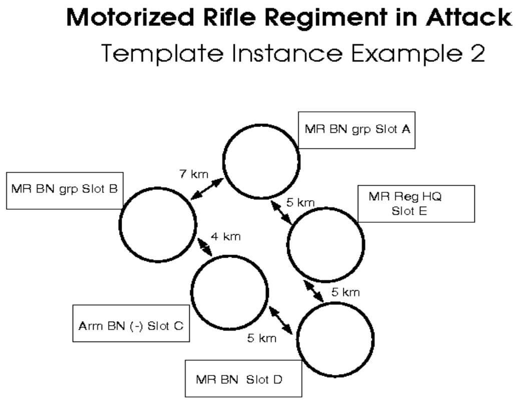


Figure 2.7: Another template instance

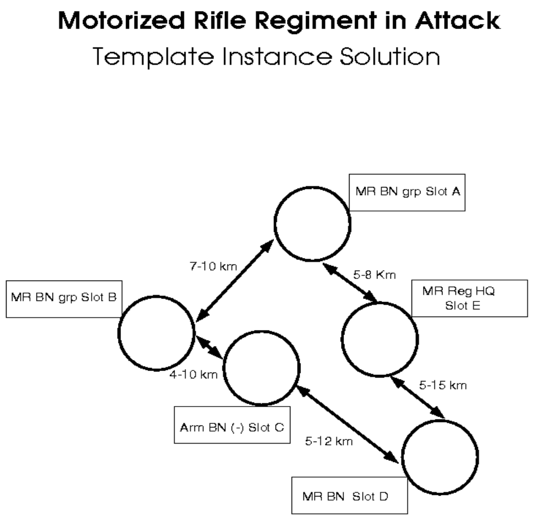


Figure 2.8: A solution template instance

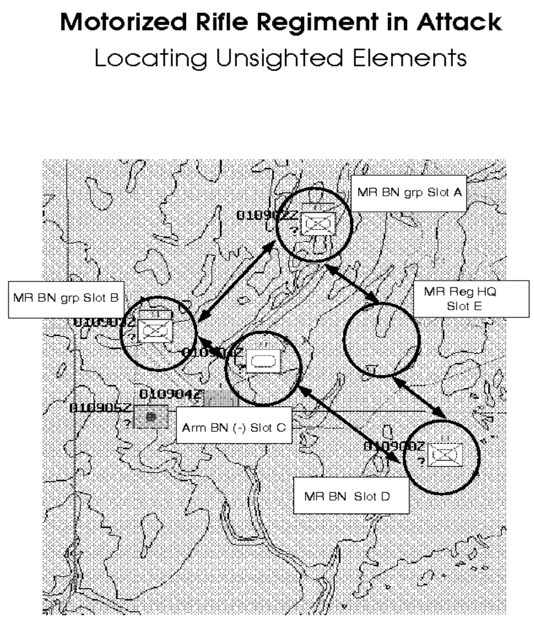


Figure 2.9: Overlay of solution on situation

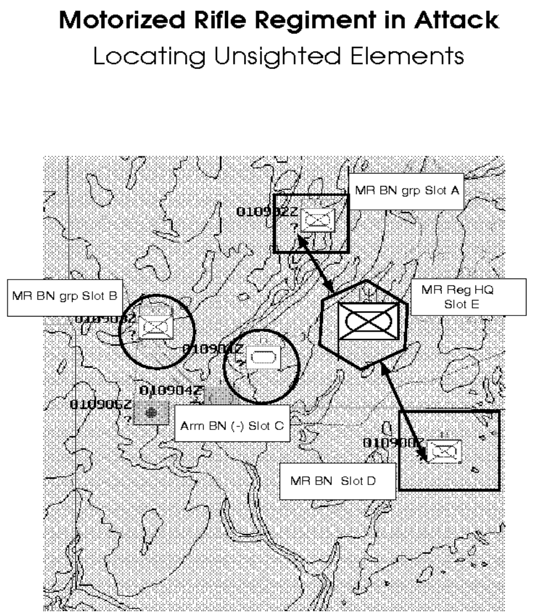


Figure 2.10: Empty slot implies element position

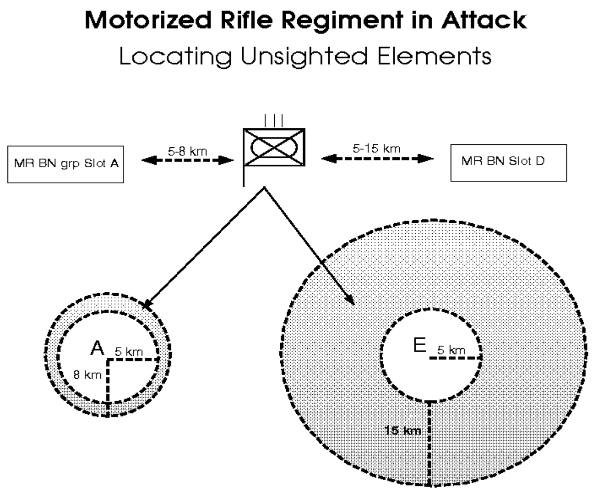
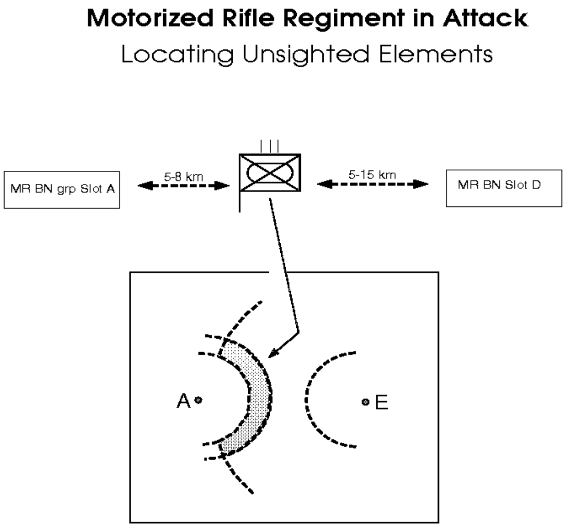


Figure 2.11: Locate element based on ranges



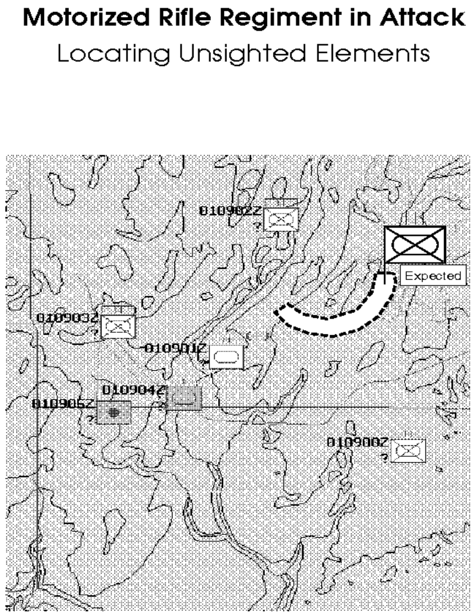


Figure 2.12: Overlapping inter-element ranges Figure 2.13: Mapping element expectation

limit the predicted location of the unsighted element. This overlap is depicted in Figure 2.13 as an overlay on the analyst's hypothetical situation view which could then be accepted, rejected, or modified.

Chapter 3

Constraint satisfaction

We wish to find a framework in which to express our templates, and a corresponding approach for attempting to recognize them in our base of situational information. One possible method of expression for this problem would be as a constraint satisfaction problem (CSP). CSPs have been studied extensively, and a variety of problem domains have been formulated in this framework. To list even a representative cross section of these papers would be difficult, however, some recent applications are discussed in [62, 71, 54, 65, 23]. Constraint satisfaction is outlined and evaluated in many papers, however, one of the most prolific implementation-based authors has been Nadel, as represented in [49, 48]. Many authors have been noted elsewhere for both broad discussions of CSP and specific theoretical and practical contributions, among these would be Mackworth [38, 39, 41, 42, 40, 37], Dechter [6, 7, 10, 4, 5, 8, 9], and Freuder [40, 42, 28, 19, 16, 17, 20].

In this section we begin with a brief description of just what constraint satisfaction problems are, and of how they are typically solved with a combination of heuristic search and constraint propagation (CP) techniques. In a later section we will reformulate our TEMPLATE recognition example as a particular CSP and discuss how the issues of constraint propagation and search affect our specific domain. Of particular interest in subsequent work will be the performance of solution strategies in this domain, and possible ways in which interaction with the user of such a system might be engineered.

3.1 Introduction to constraint satisfaction

A good introduction to CSP can be found in [35], where we are presented with a general overview of the formulation of many AI problems as CSP including those in machine vision, belief maintenance, scheduling, temporal reasoning, graph problems, floor plan design, genetic experiment planning, and the satisfiability problem. Kumar outlines several different approaches to solving these problems such as backtracking search, constraint propagation, and some hybrid combinations of the two.

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 instance all possible satisfying assignments may be of interest. Consider the following simple example from the map colouring domain. We have three variables $V1$, $V2$, and $V3$ which represent each of three countries to be coloured on a map. Each of the three countries may be coloured one of three colours *Red*, *Green*, or *Blue*. A set of constraints indicates which countries are beside one another, and further allows us to represent the rule that no pair of adjoining countries may be the same colour. Figure 3.1 shows the map we are modelling with a CSP, and Figure 3.2 shows one corresponding constraint graph reflecting the aforementioned variables, domains, and constraints.

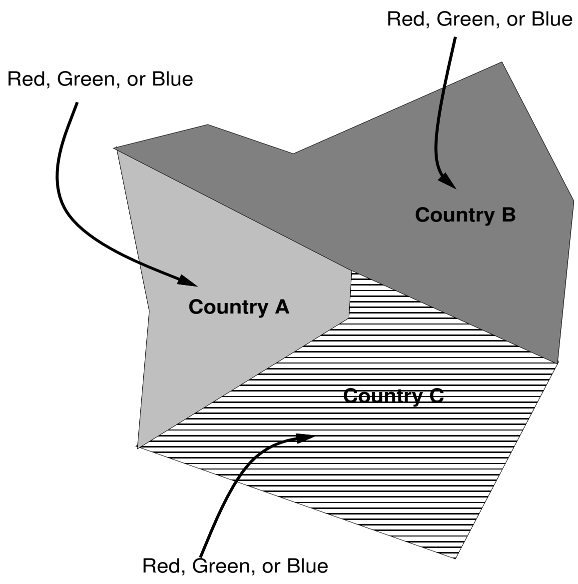


Figure 3.1: Example Map Problem.

3.2 A naive solution: Generate and Test

This example is simple as specified, since clearly we have a wide enough domain (or loose enough constraints) so that a particular assignment can be easily arrived at. We now can proceed to find solutions using some simplistic method such as generate and test. We simply search through the space of all possible assignments of colours to variables, and each assignment found to be consistent with our set of constraints is a solution. Of course, such

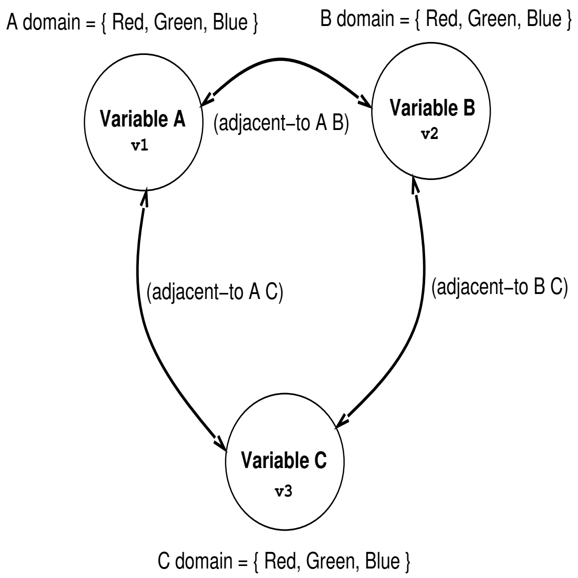


Figure 3.2: Example Map Problem as a CSP graph.

a method is quite inefficient and will not be adequate for more complex examples involving more variables, larger domain sizes, and more elaborate constraint networks. Specific search methods and heuristics that have been shown to reduce the overall search for certain types of problems will be discussed later in this paper.

Our example exhibits constraints between pairs of variables indicating that certain pairings of assignments are not acceptable. The constraints are known as *Arc constraints.* We could have also added constraints on a particular individual variable or node, that restricted the set of allowable domain values which that particular variable could be assigned. Constraints such as these are known as *Node constraints.* Figure 3.3 shows the previous problem formulation with the addition of a node constraint indicating that the country represented by variable $V1$ may not receive a colour assignment of *Red*.

3.3 Propagation of constraints

An alternate approach to solving CSPs is known as constraint propagation and is an attempt to take advantage of the limiting characteristics of constraints between the variables in the CSP. Essentially, any solution of a CSP has the quality that domain assignments to the variables must be *consistent* with one another. Any inconsistency discovered with a domain assignment and a constraint signals that that particular assignment is invalid and may be discarded. An inconsistency discovered as a result of a constraint between two variables indicates that that a particular pairwise assignment is invalid and that pair of assignments may be discarded.

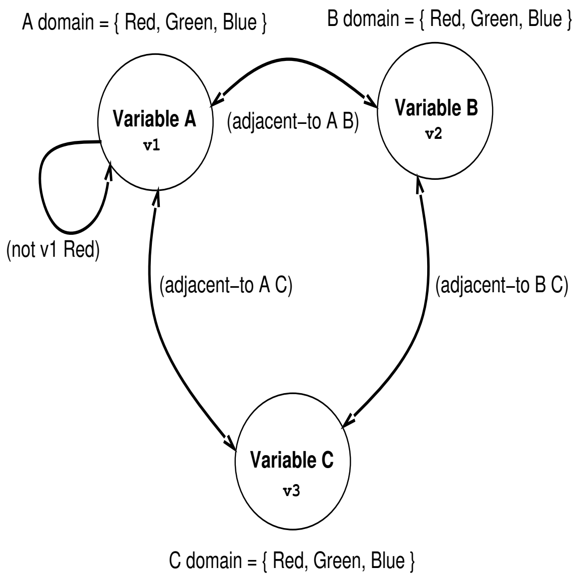


Figure 3.3: Map CSP graph with node constraints.

We will first re-state how a CSP is structured in slightly more formal terms so as to make the explanation of constraint propagation easier. Consider that we are given a set of variables $V_1 \ldots V_n$, and each of the variables V_i has a domain D_i size m. Now, consider that we have a set of constraints C which constrains which values can be assigned from each domain to each variable, and which also constrains assignments based upon pairs of variables. For instance a constraint could state that some V_k must have an assignment d from D_k such that d was a factor of 2. This is a node constraint and is expressed graphically in a CSP graph as an arc originating and ending with V_k. Another constraint might state that for some V_k and another V_j, that the assignment d_k from D_k must be greater than the assignment d_j from D_j. This type of constraint is an arc constraint, and is expressed graphically as an arc between V_k and V_j. The problem of finding an assignment of domain values to each variable such that all constraints are satisfied is the constraint satisfaction problem. A variation as stated earlier, is where we are interested in all possible assignments which satisfy the constraints.

3.3.1 Node consistency

If for each variable V_k we were to enforce any node constraints pertaining to its domain D_k by removing elements d_i from D_k where d_i violates a node constraint of V_k, then our resulting constraint graph would exhibit *node consistency.* If one or more of our variables were to have a resulting domain which is empty D_k, then our CSP has no solution possible. Algorithms for making a CSP node consistent are straightforward in that each potential domain value d_i for a particular variable V_j must be checked for all node constraints nc applicable to V_j. The worst case complexity of this process can be determined simply by taking the factor of the number of variables n, the largest number of domain values A of any of the n variables, and the largest number of node constraints applicable to any of the n variables. Of course, the actual performance will be considerably better than this since each variable v will have fewer than the maximum number of constraints on average, and fewer than the maximum domain size also. In addition, once a domain value for a particular variable fails one node constraint there is no need to check that value against subsequent node constraints for that variable. This fact suggests a simple heuristic in the ordering of node constraint checking, where those constraints most strict, or most likely to fail are checked first for a particular variable and domain value.

We know that if a solution can be found via search for the original CSP problem, than one can be found for the problem formulated in a node consistent fashion, CSP_{nodeC}, since the CSP_nc is nothing more than a version of the original CSP with many possible dead ends and their subsequent derivation paths removed from the search space in advance. Essentially we would expect the effect on search performance of making a CSP graph node consistent to be quite noticeable as we have essentially limited the branching factor of the domain values with each removal.

3.3.2 Arc consistency

Similarly, if for each *pair* of variables V_i and V_j we were to enforce all constraints c_{i_j} from V_i to V_j, and remove each domain element d_j of D_j where d_j is not consistent for c_{i_j} with any assignment of d_i from D_i to V_i, then our resulting constraint graph would exhibit *arc consistency.* Once again completeness is retained and a any solution reachable in the original CSP can be found in arc consistent CSP, CSP_{arcC}.

Several algorithms have been published that can convert a general CSP graph into an arc consistent CSP graph. Mackworth's algorithm [37] AC-3 is one of the most well known. AC-3 and other variations of arc consistency algorithms are discussed and presented in a common framework in [49]. An optimal algorithm, AC-4, for arc-consistency is presented in [46] where the worst case complexity of AC-4 is shown to be $O(ed^2)$ where e is the number of arcs (constraints) in a CSP graph, and d is the size of the largest variable domain in the problem. This algorithm is generalized further into AC-5 in [64], where AC-5 can be instantiated to produce an $O(ed)$ algorithm for a number of classes of constraints.

3.3.3 "N" consistency

[35] introduces quite proficiently the issues underlying discussions of the relative use of constraint propagation in reducing or eliminating search for particular domains. A natural question at this point is whether it is conceivable that we may achieve higher levels of consistency in our CSP in some manner such that we can save even more search effort. We must be aware that node and arc consistency are only two possible degrees of consistency. A further extension is to make a CSP graph *Path consistent* [47, 37]. In a most general

framework, arc and node consistency may be seen as level 1 and level 2 of what is *N-consistency* in general. It has been shown in [17] and is also outlined in [35] that any n variable CSP can be solved by achieving n-consistency in the CSP graph, without search. Of course, achieving this n-consistency can be at least as expensive in general as straightforward search. The cost of achieving n-consistency is high, and is avoided in most applications since it has been found to be less efficient than a lower degree of consistency application combined with search. An optimal algorithm for computing n-consistency where n is an integer Cooper in [3].

3.3.4 Utility of constraint propagation

In each of the cases of enforcing *node* and *arc consistency* we are propagating some of the constraints through the graph. Essentially we are just creating a simpler CSP from the original CSP. We might say that given an original problem $CSP_{original}$ we can derive first CSP_{nodeC} and then CSP_{arcC}. Intuitively it would now seem that the simpler CSP should be able to be solved using some heuristic search approach in less computational effort than the more complex original CSP problem, while achieving the same solution. The real issue to resolve though, lies in determining what level of advance consistency computation should be done on a CSP in order to minimize the overall effort of combined preprocessing and search ? This question have been addressed empirically in recent work [49, 24, 10], although most frequently for toy problem domains such as the N-queens and Confused-N-queens problems which involved arranging many queens on a chess board so that either they do not challenge each other (N-queens) or they all challenge others (Confused-N-queens). These problems are detailed further in many papers, and good descriptions appear in [49, 48].

3.3.5 Partial Arc consistency

In some of the experimental work [49, 10], it has been observed that a preprocessing phase accomplishing node consistency for a CSP has been cost effective in terms of overall computational savings for finding ALL solutions in some domains, but also that the cost of achieving arc consistency has been found to be not cost effective. As a result of this work, efforts have been made in [49] to see if the actual optimal benefit point for consistency actually falls somewhere between level 1 (node) and level 2 (arc) consistency. The result has been several algorithms which achieve j-consistency (partial arc consistency), where j is between 1 and 2. Results in some experimental domains have shown that these partial arc consistency algorithms combined with search outperform those utilizing only node consistency as a pre-search step [49].

Several partial arc consistency algorithms are detailed in [49], with each representing a progressively higher degree of consistency. Their names reflect the approximate amount (as a fraction) of arc consistency checking performed by each. The algorithms discussed and evaluated in [49] for the Confused-N-queens problem are listed below. These algorithms can be incorporated into backtracking search to form hybrid versions which guarantee differing levels of arc consistency at each search node. These hybrid algorithms will be discussed further in a separate chapter, Chapter 4, along with a discussion of some extensions.

- AC1, AC3, AC3, AC4, AC5 Full arc consistency Several algorithms are well known for achieving full arc consistency. Nadel in [49] describes the three most well known in common terms. These three are now known as "AC-1" [37], "AC-2" [67], and "AC-3" [37]. AC-1 differs from the other two in that the others attempt to avoid unnecessary arc checks performed in AC-1. This difference is elaborated in [49]. AC-3 differs from AC-2 in that the order in which arcs are each in turn checked for consistency is different. Nadel[49] points out that the issue of avoiding checks is deserving of more research, and observes that it is indeed an analog of the issue of constraint-check ordering we discuss briefly later in this paper in Section 3.13. Other full arc consistency algorithms are described in other work, particularly AC-4 [46] and AC-5 [64]. An optimal k consistency algorithm is presented in [3] as a generalization of AC-4.
- AC 1/5 arc consistency, apparently arrived at independently in [49, 44].
- AC 1/4 arc consistency, essentially this algorithm is "Check Forward" as outlined in [24], and as also presented in [44].
- AC 1/3 arc consistency, essentially "Partial Look Future" as described in [24].
- AC 1/2 arc consistency, essentially "Look Future" described in [24].

3.4 Hybridization of search and constraint propagation

We have so far considered two approaches to solving constraint satisfaction problems. The first, generate and test, is inherently simplistic, and the size of the search space is clearly going to be enormous in any complex domain. The second, constraint propagation can be either an incomplete solution as in the case of those algorithms which insure a degree of consistency short of total n-consistency for a problem of size n, or extremely inefficient, as in the case of total n-consistency algorithms. Several options present themselves for immediate improvement of this situation. First, generate and test can be replaced by a complete and more efficient heuristically directed depth first approach. Second, we can take advantage of the simplification properties of arc consistency algorithms in several different ways. An obvious approach is to use some degree of arc consistency propagation as a preprocessing phase, simplifying the original CSP and thus limiting much of the search required to solve the problem. This preprocessing behaviour is discussed in [10]. We have already mentioned this option, and will discuss it further through this paper. Still another option is to consider a slightly different hybridization of search and constraint propagation which has been suggested and has been the focus of experimental analysis by several authors including [49].

We will consider the nodes of a search tree as each representing a new CSP which is simpler in that one variable from the original CSP has already been instantiated. At each level of the tree, we instantiate another variable as in normal search, but each node is considered independently now, and some level of consistency algorithm may be first applied to that node before attempting a solution or refinement from that level. Essentially we are interleaving search and constraint propagation so as to limit the branching options further at each search node and the particular CSP problem that node represents. Several search algorithms and variations of search algorithms behave much as if there were a certain amount of constraint processing being performed at each node, even if this fact is not stated explicitly. Nadel [49] attempts to standardize this behaviour in order to determine how much constraint processing is of benefit during search. It is for this reason that the fractional AC algorithms were developed and are listed in the previous section.

We could possibly locate a solution (if the variable domains become all size 1 and the CSP is arc-consistent) or prune the node (if a variable domain becomes empty), or generate successor nodes in our search space (for some selected variable we generate all possible assignments to that variable). Each new successor CSP will be progressively "smaller" than its parent since the variable chosen at the succession point will never be selected again, and we now have one less variable in our CSP.

Experiments by other researchers [24, 49] have indicate that it seems best performance is obtained across a variety of domains when inherent constraint propagation is only applied in a limited form. Results with hybrid algorithms are discussed in the context of partial constraint propagation in [49], and many other papers give experimental results for particular search strategies that exhibit partial constraint propagation without explicitly stating their behaviour as such. In the next section we will discuss various search approaches often utilized and presented in conjunction with constraint satisfaction problems.

3.4.1 Sharing AC work in hybrid search

An interesting issue to consider in hyrbidized search and constraint propagation is whether or not work spent in one part of the search space in reducing a particular CSP remaining can be shared to other CSPs in the space, either completely or partially.

For example, since instantiation of a particular variable in n ways results in n remaining CSPs, each one variable smaller than the original CSP. Each CSP graph remaining is similar to its siblings in the still uninstantiated variables, and only those arc constraints connecting the newly instantiated variable and the still uninstantiated variables define the difference. Perhaps if some partial arc-consistency algorithm were utilized which acted only on the similarities between the CSP siblings, the work and simplification of one CSP could be shared amongst the others.

Later in Chapter 5, we describe one possible sharing of propagation during depth first search.

3.5 Search in CSP

CSPs can be solved using search alone, using consistency propagation alone, with search and using consistency propagation as a preprocessing phase, or by using some hybrid approach interleaving constraint propagation and search. As we have stated, the CSP search space consists of nodes where each node consists of a set of particular instantiations of variables with domain values. The initial or start node represents the least constrained version of the problem. The search space is generated by selecting a variable V to instantiate at each *level* (the variable order), selecting an order for the selection of domain values for each variable V (the domain orders), and proceeding to search the space with an appropriate search strategy. If we have n variables, then all solution nodes will exist as leaves at depth n. Since all solutions are leaves, and the tree is bounded in depth by n, clearly a depth first approach will be complete and we will eventually find all solutions that exist if we desire. Selection of variables to instantiate, and determination of which domain values to assign to a variable at each point in the search space are key issues in controlling the search.

Selection of a search strategy can be highly problem dependent. For instance, if one requires all solutions, then some form of complete search will be required, likely with the addition of heuristics to shorten the search time, if any single solution will satisfy the requirements in a particular case, then some type of quick but incomplete search may be what is required in order to more efficiently find the single answer. Possibly some combination of these two search strategies will be appropriate in cases where a single solution may be inadequate but several solutions may suffice, or where a particular solution must meet some outside (possibly manual) criteria to determine it's appropriateness. If possible, an interactive search utilizing an algorithm for *ALL* solutions may be appropriate if it can be stopped after a "satisfactory" solution has been detected. In such cases where a complete strategy is used, the order of arrival at various solutions can be accommodated directly into the search strategy in the form of control heuristics for the selection of variables to instantiate, the order of domain values to attempt to assign, and for the control of expanding the search strategy deeper or wider in our quest for a "good enough" solution. A later section of this paper will be dedicated to the discussion of this type of interaction and with the presentation of a particular overall heuristic to add to complete search that facilitates this ongoing interaction in search of appropriate or satisfactory solutions.

One single solution strategy that has shown great promise in various domains is referred to as "Local Search". Local Search works by performing a random assignment of domain values to each of the variables in the CSP, and then by attempting to repair the assignment according to some heuristic function. One common repair heuristic is to minimize the conflicts that arise with a particular assignment using a hill climbing approach. Local Search has been shown in [45, 60] to be orders of magnitude faster in finding single solutions quickly for difficult problems involving large variable and domain sets with many constraints. Recent ongoing work in the university course scheduling domain has also been discussed with similar results [71].

For the purposes of search in CSPs, if we make the assumption that each successive search layer results in one more variable instantiated, then each node towards a leaf represents a simpler CSP left to solve. So if there are n variables in some CSP, then the search space will be exactly n levels deep. We know all solution are leaf nodes, and for this reason, directed depth first search will be complete. The algorithms often cited in constraint propagation search literature include *Simple Backtracking*, *Forward Checking*, and *Partial Lookahead*, *Full Lookahead*, *Really Full Lookahead* [24]. Kumar [35] describes Nadel's work [49] which shows an empirical comparison of the performance of these algorithms in a unified framework constructed by Nadel. In this comparison, these algorithms are described primarily in terms of the degrees of arc consistency which is performed at the nodes of the search tree, essentially what level of hybridization they each represent. The definitions are show below with a depiction of each algorithm as a combination of pure tree search and algorithms which guarantee at each backtrack node varying fractions of arc consistency as described earlier.

- Generate and Test (GT)
- Simple Backtracking (BT = GT + AC 1/5)
- Forward Checking (FC = GT + AC 1/4)
- Partial Lookahead (PL = FC + AC 1/3)
- Full Lookahead (FL = FC + AC 1/2)
- Really Full Lookahead (RFL = FC + AC)

We will look at each strategy in more detail in subsequent subsections. We are interested in how each method combines various consistency checking with search, and in later experiments in determining which is most appropriate and effective with our specific domain.

3.6 BT: Simple Backtracking

BT incorporates some consistency checking implicitly since whenever a new variable is selected for instantiation, any values which are inconsistent with previous instantiations indicates a successor that may be pruned. This differs from Generate and Test in that GT would only determine consistency or inconsistency at the leaf nodes. All possible combinations would be generated, and then tested.

3.7 FC: Backtracking with Forward Checking

FC utilizes consistency checking more than BT by filtering all yet uninstantiated variables for consistency with previously instantiated variables values at successor creation time. In this way, we avoid generating some successor nodes that will fail against previous instantiations.

3.8 FCDR: Backtracking with Forward Checking and Dynamic Rearrangement

The Dynamic Rearrangement heuristic is basically an adjustment to any Forward Checking algorithm so that after a particular node is filtered removing any domain values for the remaining variables that are inconsistent with previous instantiated variable assignments, the next variable selected for instantiation is the one with the smallest domain size. This is essentially an additional attempt to limit the branching factor of the tree by selecting the next variable to instantiated more intelligently.

3.9 RFL: Really Full Lookahead

Really Full Lookahead, Full Lookahead, and Partial Lookahead are essentially augmented versions of FC in that they perform additional arc consistency checks even between uninstantiated variables. RFL essentially limits all remaining domains by insuring full arc consistency before selecting the next variable for instantiation using the Dynamic Rearrangement or some other method.

3.10 FL: Full Lookahead

Full Lookahead differs from RFL in that we utilize the partial consistency algorithm "AC 1/2" described by Nadel in [49]. Essentially, this algorithm is the same as Haralick's "Look_Future" described in [24].

3.11 PL: Partial Lookahead

Partial Lookahead, essentially Haralick's as Partial_Look_Future, is described by Nadel in terms achieving a degree of consistency suggested by his algorithm "AC 1/3".

3.12 Intelligent Backtracking

In previous results including [49, 24], it has been observed that during the constraint application in consistency checking, some constraints are repeatedly checked between pairs of variables despite no possibility of a different result in each subsequent case. Also, portions of the search space that have been discarded in earlier search as a result of arc inconsistency are revisited in later search. This behaviour is referred to in the literature as *thrashing*. Modifications have been suggested by several authors to try to avoid this redundant constraint checking and search. Two methods known as BackJumping and BackMarking are discussed below. Other more complex and expensive methods are described in [35], some of which minimize redundant checking even further, but at a much greater computational, space and complexity cost.

3.12.1 BackJumping

Consider the case where we have three variables for assignment and consequently a search tree of depth 3. Assume that the variables are selected in the order $V1$, $V2$ and $V3$. Also assume that there is a constraint between $V1$ and $V3$, call it $C_{1.3}$. If $V1$ has a domain of $(1, 2, 3)$, and $V2$ has a domain of (A, B, C), and $V3$ has a domain of $(X1, X2, X3)$, then consider the section of the tree shown in Figure 3.4 after we instantiate $V1$ to 1. On the "first" instantiation of $V2$ to A, there are no constraints, so we descend to $V3$, and attempt to instantiate $V3$ to its domain values one at a time. If the constraint $C_{1.3}$ fails $C_{1.3}$ for each domain value of $V3$, a normal backtracking method would retreat and attempt a new value for $V2$. Now, clearly this doesn't make any sense, since the conflict causing the backtrack is unrelated to the current instantiation of $V2$. We can infer now that the assignment of 1 to $V1$, is incompatible with any domain value of $V3$. In a sense we have learned something about arc inconsistency in the CSP graph. We can take advantage of this knowledge now by backtracking to the incompatibility, namely $V1$, and assigning a new domain value to $V1$, and bypassing any other attempts to instantiate $V2$. So not only have we saved ourselves the useless work of re-checking redundant constraints, but we have pruned a considerable portion of the search space involving each domain value of $V2$.

This approach will only be beneficial when there is arc inconsistency present in the CSP we are attempting to solve.

3.12.2 BackMarking

BackMarking is explained in some depth in [49], however to put it simply, BackMarking attempts to avoid repeating redundant consistency checks where possible. Whereas BackJumping avoids some cases (and some search space) where a set of consistency checks fail, BackMarking searches the same space as Backtracking, but with fewer consistency checks.

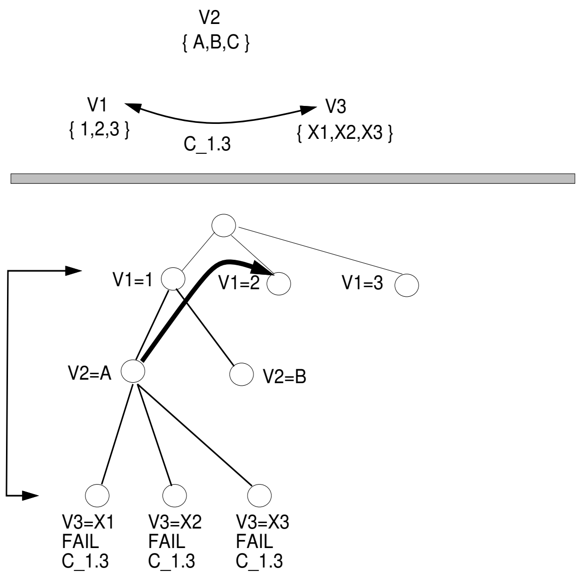


Figure 3.4: Example of BackJumping Behaviour.

3.13 Heuristic extensions to the search process

3.13.1 Preprocessing with advance consistency checking

We have seen in earlier sections how CSPs can be solved using various search approaches alone, using consistency algorithms without search, or using a hybrid combination of search and some degree of partial consistency checking during search. A more simplistic approach is possible in which some degree of consistency checking is performed prior to search only, and then some search strategy is performed on the simplified version of the CSP resulting.

3.13.2 Preprocessing with variable ordering

We know that the order in which variables are selected for instantiation directly determines the shape and structure of the search tree for a particular CSP. Dynamic rearrangement is one such heuristic improvement to Forward Checking search which attempt to control the shape of the search tree by selecting the variable with the smallest domain for next instantiation.

We know that if given a particular CSP, we can either attempt to solve it or other simplified versions obtained through advance consistency checking by some search method. Since the search space is determined by the order in which the variables are selected for instantiation, it is conceivable that some heuristic for ordering these exists in a particular domain that may be of benefit. Certainly one option is to simply order the variables from smallest domain size to largest which is similar to the approach taken by Dynamic Rearrangement, except that this would be more accurately labelled Static Advance Rearrangement.

3.13.3 Constraint application ordering

As a consequence of backtracking search, each domain value selected as a variable assignment at some branching point in the search space must be checked to see if it is consistent with each variable assignment made at previously visited search levels. As a result, any constraints between this variable and already instantiated variables must be checked. As with node constraints, a single such arc failure is enough to signal a backtracking point, and consequently it would be advantageous to attempt evaluation of these constraints in an order which would maximize the chances or finding a failure earlier. In other words, satisfying the "tightest" constraints first would seem to be an appropriate heuristic for avoiding unnecessary constraint checks.

3.13.4 During hybrid search

We have identified several interesting areas for development of appropriate heuristics during search for this domain. These areas include:

- Propagation only certain types of constraints after variable instantiation. Specifically, consider propagating spatial constraints that limit the field of vision of remaining variables at little cost. Perhaps some form of spatial index could be used to quickly filter variable domain sets, or possibly this filtering could be done via some kind of call to a data parallel machine, passing sets of variables and requesting the same elimination process applied to all sets simultaneously[1].

- Dynamic selection of variables to instantiate can dramatically reduce the branching factor of search. Whether or not to undertake this action would depend on understanding better the amount of work that results from more or less constraint checking. This is an open area of research.

- During search, even in depth-first search, the question of which domain value to attempt to instantiate next (or the order), is an issue of heuristics. Abstraction attempts to order the order of variable selection somewhat across levels, however, the question of instance selection is not addressed. Conceivably it might be beneficial to choose instances with close proximity to some other instance already assigned. At any rate, some ordering might be imposed on domain instances according to space or other method.

- As discussed in Chapter 5, it may be possible to propagate certain constraint reductions of a problem through the entire search space in limited instances. Determination of what heuristics might identify when this effort is justified is an open area of research.

- The order in which constraints are applied during constraint propagation or verification of variable instance consistency during search is a matter of heuristic. Selection of expensive constraints which seldom yield a negative response can result in much wasted computation. In some fashion we would want to order the application of these such that if a negative response was anticipated, the most "cost-effective" and "restrictive" constraints would be applied first, perhaps according to some ratio measuring their chance of returning a negative value to cost of application.

3.14 Different CSP representations of the same problem

It is important to note here that a particular problem instance such as the map colouring problem may be represented in many different ways, each a different constraint satisfaction problem with different numbers of variables, domains, and constraints. The particular representation chosen directly affects the structure of the problem search space and consequently the performance of various search methods applied to this space. In [48],the relative costs

[1]This concept was mentioned to Dr. Vezina of DREV in a discussion.

and benefits of selecting different representations for the same problem are discussed. In addition the performance trade-offs are determined both experimentally and theoretically.

Chapter 4

Partial arc consistency examined

The work presented in the sections of this chapter has been undertaken and is presented by Denis McGonigal[1] under the supervision of Steven Woods. Each subsection of this section was originally intended to "standalone", and for this reason the introductory portions of each subsection may be repetitive.

We are interested in interleaving constraint propagation and search. Arc consistent problems have considerably less redundant constraint checking, and substantially reduced search spaces than the same non-consistent problem. Making a graph originally node or arc consistency requires work. Searching for solutions requires work. Combining search and constraint propagation results in some total amount of work to find all solutions. We are interested in reducing this total work as much as possible. Experiments have suggested that making a problem representation fully arc consistent in advance of search is not cost effective for overall work in this domain with typical problems. However, making a problem node consistent is effective since many impossible options are discarded at little cost. If node consistency represents 1-consistency, and arc consistency represents 2-consistency, we are interested if some partial consistency greater than 1 and less than 2 for the whole problem will be beneficial. Essentially some portion of the graph will have its consistency increased by removing some inconsistent domain values.

The identification of graphs which could benefit from this application is an ongoing research interest. In addition, interleaving search and constraint propagation algorithms can result in search that demonstrates certain "Look-ahead" qualities. We intend to experiment in this domain with such hybrid algorithms and attempt to identify problem aspects that can help in the design of heuristically improved algorithms.

Thus, active research areas include selection of problems in which to apply partial arc consistency, identification of graph portions to apply the algorithms to, and determining how much effort should be spent before or during search.

[1] A summer student at DREV between June and August, 1993.

4.1 Overview of existing fractional arc consistency algorithms

Introduction

A large number of problems in AI and other areas of computer science can be viewed as special cases of constraint satisfaction problems (CSP). Arc consistency algorithms are an important class of algorithms used for solving CSP as they can simplify the problem.

The main focus of this section is on existing (published) partial or fractional arc consistency algorithms. It resumes the behavior of these algorithms and presents an example of a CSP and traces the algorithms as they work on the problem.

Constraint Satisfaction Problem (CSP)

The goal of a CSP is to find assignments of the variables to their candidate values such that all the constraints are satisfied.

A CSP is defined by three components :

- A finite set of variables V, where $V = \{V_i : 1 \leq i \leq n\}$,
- A finite and discrete domain D, where $D = \{(d_1, d_2, ..., d_n) : d_i \in D_i, 1 \leq i \leq n\}$, and $D_i = \{$ possible instances of $V_i\}$, and
- A set of constraints C, where $C = \{C_{ij} : 1 \leq i, j \leq n\}$ and C_{ij} is a constraint[2] between V_i and V_j.

Classes of Algorithms Used in Solving CSP

Three important classes of algorithms have been developed to find the solutions of CSP's. These are tree search, arc consistency (AC), and hybrid tree search/arc consistency algorithms.

Tree Search Algorithms

The tree search algorithms deal with searching through a tree of instances of the variables and then determining if these instances satisfy all the constraints.

Arc Consistency Algorithms

Arc consistency algorithms are an important class of algorithms used for solving CSP as they are used simplify the problem. These algorithms are the main focus of this section, in particular what are called fractional arc consistency algorithms.

[2]It assumed that all constraints are binary.

Hybrid Algorithms

It has been found that by doing some arc consistency processing combined with tree search can improve the efficiencies of the above methods if they are implemented properly together. Hybrid algorithms marry the arc consistency and tree search algorithms together.

Arc Consistency

Various tree search algorithms have been developed for finding solutions to CSP. All of these algorithms have shown inefficiencies that have spawned the need for preprocessing to reduce the original CSP into simpler CSP.

Preprocessing attempts to reduce the number of nonsolution instances encountered in a tree search without affecting the number of solutions to the problems. This simplification is attempted by what are known as consistency algorithms.

Consistency

Definition: Consistency A set of variables are said to be consistent if all the constraints among these variables can be satisfied.

The notion of consistency exists in varying degrees determined roughly by the number of variables within the given set. It has been shown that in general, it is most efficient to perform consistency preprocessing by what is called arc consistency.

Definition: Arc Consistency

Let V_i and V_j be variables from a CSP. An arc (V_i, V_j) is arc consistent if for every value χ in the current domain of V_i, there is some value γ in the domain of V_j such that $V_i = \chi$ and $V_j = \gamma$ are permitted by the binary constraint between 'V_i' and 'V_j'.

Note that the concept of arc consistency is directional (and not always a reflexive relation); that is if an arc (V_i, V_j) is consistent, then this does not imply that the arc (V_j, V_i) is also consistent.

Revision

An arc (source, target) can be made consistent by simply deleting those values from the domain of 'source' for which the previous condition is not true. Arc consistency algorithms use this method of revision typically implemented through a function called revise[3].

Because of this revision, an AC algorithm can be thought of as a simplification algorithm which transforms the original problem into a simpler version that has the same solutions.

[3]Once REVISE reduces the domain of some variable V_i, each previously revised arc (V_j, V_i) has to be revised again to obtain arc consistency because some of the members of the domain of V_j might no longer be compatible with any remaining members of the revised domain of V_i

In some cases the resulting problem is so simple that the solutions (or lack thereof) become manifest and the original problem is solved.

A call to the revise function may appear as the following:

revise (source, target, VAR d, VAR empty_source_domain, VAR change).

The reference array parameter d contains current domains. The variables empty_source_domain and change are included as reference parameters to inform the calling routine respectively whether the domain of source was totally depleted and whether a deletion occurred at all.

There is also another function called rrevise that revises a source over a set of targets. For example, a call to rrevise may appear as the following :

rrevise (source, target_set, VAR d, VAR empty_source_domain, VAR change).

Full Arc Consistency Algorithms

A variable network is said to be arc consistent when each arc is consistent. An algorithm that guarantees arc consistency of the (sub)network on the nodes to which it applies is said to be a full arc consistency algorithm for that (sub)network.

Many full arc consistency algorithms exist, however research has concluded that it is more efficient to use hybrid tree search/arc consistency algorithms that use fractional arc consistency algorithms to solve CSP[4].

Fractional Arc Consistency Algorithms

Arc consistency algorithms that revise various subsets of arcs in a variable network are called fractional arc consistency (ACq) algorithms. The suffix q represents a fraction, and is roughly proportional to the degree of fractional arc consistency that it achieves.

This section introduces several fractional arc consistency algorithms for revising various combinations of arcs in the constraint network $G_{1...n}$. A table is included to summarize and allow comparisons of the algorithms.

All of the arc consistency algorithms use some type of a revise function. The letters s and t are used to denote source and target variables introduced in the revision section.

AC 1/5 PROCEDURE AC 1/5 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
t = 1
WHILE ((T < k) AND (not empty_source_domain)) DO
   revise(k, t, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   t = t + 1
```

[4]Nadel, p.214, see references, has determined that an algorithm called Forward Check (or TSAC1/4) is the most efficient hybrid algorithm to use in solving CSP

```
END;
```

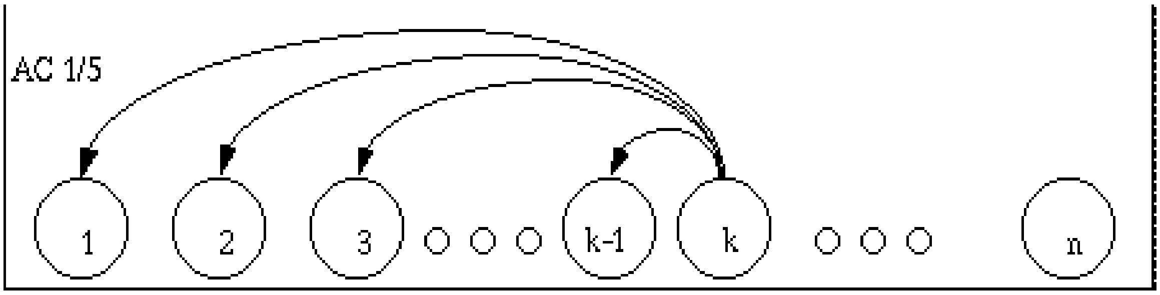


Figure 4.1: Fractional AC algorithm 1/5.

AC1/5 revises the arcs (k, t), $1 \leq t \leq$ k-1, once each. In other words, given a source node k, it revises the arcs with targets indexed with a smaller number than k, i.e. arcs (k, 1), (k, 2), ... , (k, k-1) are revised.

AC 1/4 PROCEDURE AC 1/4 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = k
WHILE ((s < n) AND (not empty_source_domain)) DO
   revise(s, k-1, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

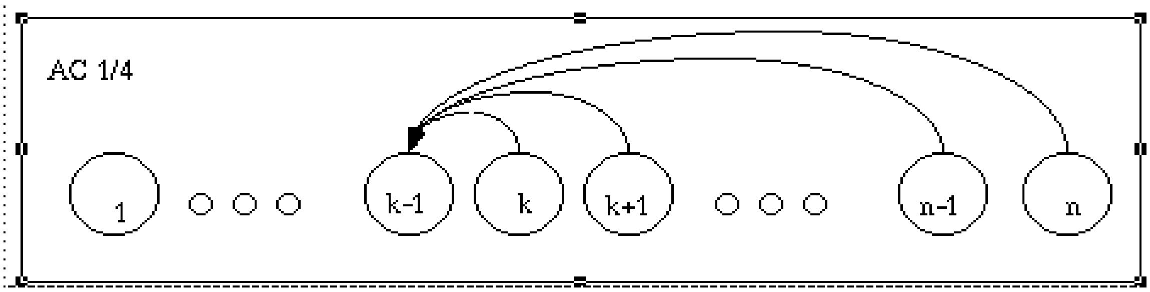


Figure 4.2: Fractional AC algorithm 1/4.

AC1/4 revises the arcs (s, k-1), $k \leq s \leq n$, once each. In other words, given a target node k-1, it revises the arcs with sources indexed with an index greater than k-1, i.e. arcs (k-1, k), (k-1, k+1), ... , (k-1, n).

AC 1/3 PROCEDURE AC 1/3 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = k
WHILE ((s < n) AND (not empty_source_domain)) DO
   rrevise(s, s+1 to n, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

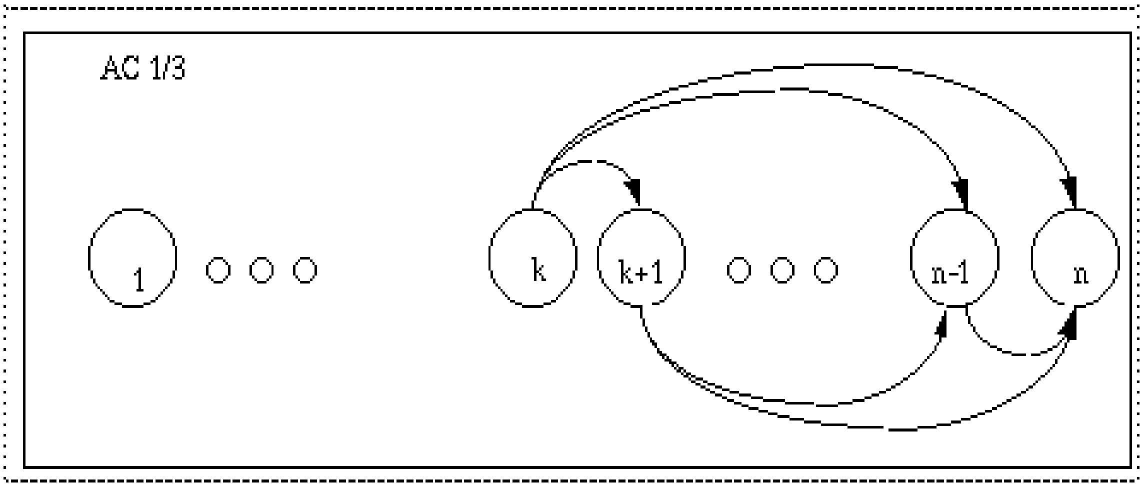


Figure 4.3: Fractional AC algorithm 1/3.

AC 1/3 revises the arcs (s, t), k ≤ s ¡ t ≤ n, once each. In other words, given k, it revises all the arcs such that source and target indexes are between k and n and such that the target index is greater than the source index, i.e. arcs (k, k+1), ... , (k, n), (k+1, k+2), ... , (k+1, n), ..., (n-1, n).

AC 1/2 PROCEDURE AC 1/2 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = k
WHILE ((s $\leq$ k) AND (not empty_source_domain)) DO
   rrevise(s, k to n but not s, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

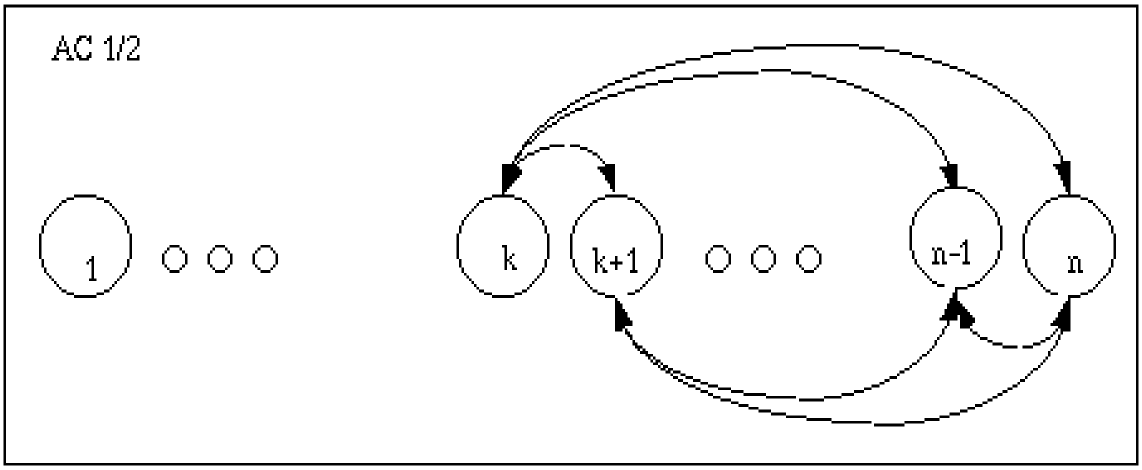


Figure 4.4: Fractional AC algorithm 1/2.

AC 1/2 revises the arcs (s, t), k $\leq$ s $\neq$ t $\leq$ n, once each. In other words, given k, it revises all the arcs such that source and target indexes are between k and n, i.e. arcs (k, k+1), ... , (k, n), (k+1, k), (k+1, k+2), ... , (k+1, n), ..., (n, k), ... (n, n-1).

ACq Summary In Figure 4.5 summarizing fractional AC algorithms, k is a parameter passed to the ACq function and does not change in value.

Representation of CSP

Constraint Networks

A binary CSP is a CSP which has binary constraints, in other words a CSP where the constraints involve at most two variables.

Constraint networks can be used to represent and solve binary CSP.

A constraint network is a labeled graph with a node for each problem variable and an arc between each pair of nodes for which there is constraint between the corresponding two variables. The nodes and (optionally) the arcs are labeled respectively with the names of the corresponding variables and consraints.

Not all the information about the CSP is captured in the constraint network as information on the domains are not included in the representation. This can be handled by expanding each network node to include one subnode for each domain value of the corresponding variable, and each pair of values that is consistent for two constrained variables is joined by its own subarc. Two variables that have no problem constraint between them can be thought of as being constrained by an implicit universal constraint.

ACq Algorithm	Revised Arcs	Ranges	Arc Nodes that are Varied	Relation Between Source abd Target
1/5	(k, t)	$1 \leq t < k$	targets	target index less than source index
1/4	(s, k-1)	$k \leq s \leq n$	sources	target index greater than source index
1/3	(s, t)	$k \leq s < t \leq n$	both	source index less than target index
1/2	(s, t)	$k \leq s \neq t \leq n$	both	source index not equal to target index

Figure 4.5: Fractional AC algorithm Overview.

Notation Suppose that there are n nodes representing problem variables and that they can be indexed by the numbers 1 to n. A constraint network with n nodes is denoted by $G_{1...n}$. G is used to denote a graph and 1...n the indexed nodes V(1) to V(n). Using similar notation, a subnetwork is defined as G_A, where A is a subset of the numbers 1 to n.

Domain Array

A different graphical representation that is sometimes convenient is what is called a domain array. A domain array has a row for each problem variable. The rows therefore correspond to nodes in the constraint network representation. Each row is labeled by the corresponding variable name (unless some standard ordering is implicit). The i^{th} cell of a row corresponds to the i^{th} domain value for the corresponding variable; array rows will therefore be of different lengths when variables have different domain sizes.

Example Problem : Relative Position

Problem Definition

Consider a grid of size 8 by 8. There is a set of variables $V = \{V_i : 1 \leq i \leq 4\} = \{A, B, C, D\}$. Here we have replaced the variable array by the first letters of the alphabet. Each variable takes on a coordinate value (x, y).

The constraint network and domain array representations are abandoned for the following method to give a better visual representation of the problems domain.

The domains are shown on the grid map below. Rather than assigning a value (instance) to a variable, variables are assigned to the values.

This method is chosen since the problem variables take on coordinate values from a small region in the plane.

8								
7					B	B	B	B
6			A					
5	A	A		A				
4	D							C
3		D					C	
2			D			C		
1				D	C			
	1	2	3	4	5	6	7	8

Figure 4.6: Variable domains for Relative Position example.

The constraints are shown in Figure 4.7.

$$C(A,B) = (3 \leq (Bx - Ax) \leq 4) \wedge (-1 \leq (By - Ay) \leq 1)$$

$$C(A,C) = (3 \leq (Cx - Ax) \leq 6) \wedge (3 \leq (Ay - Cy) \leq 6)$$

$$C(A,D) = (3 \leq (Ay - Dy) \leq 6) \wedge (-1 \leq (Dy - Ay) \leq 1)$$

$$C(B,C) = (3 \leq (By - Cy) \leq 5) \wedge (-1 \leq (Bx - Cx) \leq 1)$$

$$C(D,C) = (3 \leq (Cx - Dx) \leq 5) \wedge (-1 \leq (Dy - Cy) \leq 1)$$

Figure 4.7: Constraints for Relative Position example.

Traces

The traces of the ACq algorithms are shown below in Figure 4.8 to demonstrate the work they do to the original domains.

The tables in Figures 4.9 to 4.12 show the domains left over after the ACq algorithms have been applied to the original domains. The lower case letters represent variable instances that have been deleted. Upper case letters represent variable instances that can still exist.

ACq Algorithm	K	Revision of Arcs (s,t)	Deletions from Source Domain
		(D,A)	(1,4)
		(D,B)	NIL
1/5	4	(D,C)	(4,1)
		(B,A)	(5,7), (8,7)
		(C,A)	(8,4)
1/4	2	(D,A)	(1,4)
		(B,C)	NIL
		(B,D)	NIL
1/3	2	(C,D)	(5,1), (8, 4)
		(B,C)	NIL
		(B,D)	NIL
		(C,B)	(5,1)
		(C,D)	(8,4)
		(D,B)	NIL
1/2	2	(D,C)	(1,4), (4,1)

Figure 4.8: Fractional AC results in Relative Position examples.

Interesting Note

Suppose that there existed an intelligent AC algorithm such that it would be able to do the following revises respectively : (A, B), (C, D), (D, C), (B, A). These arc revisions would leave the following domain values shown in Figure 4.13. All of these instances of the variables happen to be solutions.

4.2 Many more fractional arc consistency algorithms

Introduction

Arc consistency (AC) algorithms are an important class of algorithms used for solving CSP as they are used simplify the problem. There are currently two classes of arc consistency

8								
7					B	B	B	B
6			A					
5	A	A		A				
4	d							C
3		D					C	
2			D			C		
1				d	C			
	1	2	3	4	5	6	7	8

Figure 4.9: AC1/5(4) Relative Position example call.

8								
7					b	B	B	b
6			A					
5	A	A		A				
4	d							c
3		D					C	
2			D			C		
1				D	C			
	1	2	3	4	5	6	7	8

Figure 4.10: AC1/4(2) Relative Position example call.

algorithms. These are full and fractional (or partial) arc consistency.

A variable network is said to be arc consistent when each arc is consistent. An algorithm that guarantees arc consistency of the (sub)network on the nodes to which it applies is said to be a full arc consistency algorithm for that (sub)network.

Arc consistency algorithms that guarantees arc consistency on various subsets of arcs in the network are called fractional arc consistency (ACq[5]) algorithms. Most of the ACq algorithms can be seen as 'dumb' AC algorithms as they simply perform single iterations of revisions on an arbitrary subset of arcs. These 'dumb' algortithms can be cleverly used to construct 'intelligent' algorithms such as the AC-3 algorithm that iteratively uses the AC-1/2 algorithm [49]. By defining more of these 'dumb' ACq algorithms, other more efficient AC algorithms may be cleverly constructed.

The main purpose of this section is to describe these 'dumb' ACq algorithms. Some of these are new while most are based on those already introduced in [49].

[5]The suffix q represents a fraction, and is roughly proportional to the degree of fractional arc consistency that it achieves.

8								
7					B	B	B	B
6			A					
5	A	A		A				
4	D							c
3		D					C	
2			D			C		
1				D	c			
	1	2	3	4	5	6	7	8

Figure 4.11: AC1/3(2) Relative Position example call.

8								
7					B	B	B	B
6			A					
5	A	A		A				
4	d							c
3		D					C	
2			D			C		
1				d	c			
	1	2	3	4	5	6	7	8

Figure 4.12: AC1/2(2) Relative Position example call.

Revision

An arc (source, target) can made consistent by simply deleting those values from the domain of 'source' for which the previous condition is not true. Arc consistency algorithms use this method of revision typically implemented through a function called revise.

Because of this revision, an AC algorithm can be thought of as a simplification algorithm which transforms the original problem into a simpler version that has the same solutions. In some cases the resulting problem is so simple that the solutions (or lack thereof) become manifest and the original problem is solved.

A call to the revise function may appear as the following:

revise (source, target, VAR d, VAR empty_source_domain, VAR change).

The reference array parameter d contains current domains. The variables empty_source_domain and change are included as reference parameters to inform the calling routine respectively whether the domain of source was totally depleted and whether a deletion occurred at all.

There is also another function called rrevise that revises a source over a set of targets.

8								
7						B	B	
6			A					
5								
4								
3		D					C	
2			D			C		
1								
	1	2	3	4	5	6	7	8

Figure 4.13: Ideal partial AC result domains.

For example a call to rrevise may appear as the following :
rrevise (source, target_set, VAR d, VAR empty_source_domain, VAR change).

Fractional Arc Consistency (ACq) Algorithms

Some of the algorithms in this section are postfixed with the word 'in' or 'out'. The postfix 'out' is used to denote that the source is static and that revisions "are performed out of that source". The postfix 'in' is used to denote that the target is static and that revisions "are performed into that target".

The algorithms AC 1/5 OUT, AC 1/4 IN and AC 1/2 are found in [49] as AC 1/5, AC 1/4 and AC 1/2 respectively. AC 1/3 FORWARD is found in Section 4.4 as NEW AC 1/3. All other algorithms are introduced for the first time in this section.

AC 1/5

PROCEDURE AC 1/5 OUT(k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
t = 1
WHILE ((T < k) AND (not empty_source_domain)) DO
   revise(k, t, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   t = t + 1

END;
```

AC1/5 (see Figure 4.14) revises the arcs (k, t), $1 \leq t$ ¡ k, once each. In other words, given a source node k, it revises the arcs with targets indexed with a smaller number than k, i.e. arcs (k, 1), (k, 2), ... , (k, k-1) are revised.

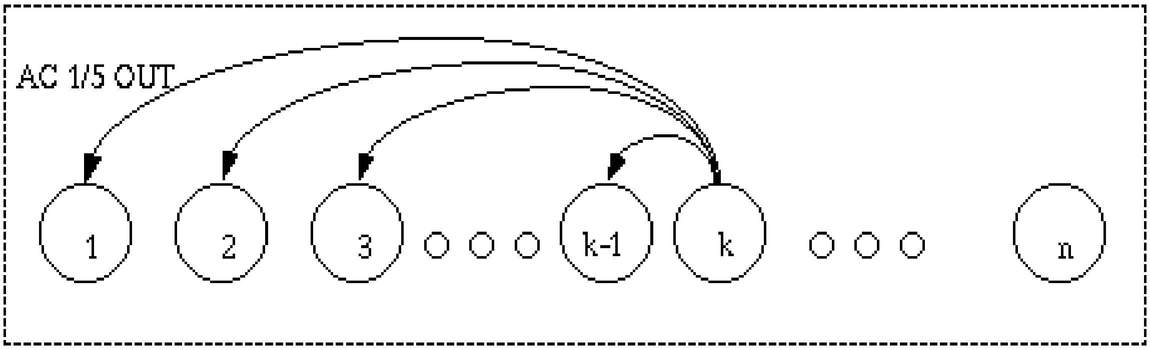


Figure 4.14: AC1/5 behaviour.

PROCEDURE AC 1/5 IN(k, VAR d, VAR empty_source_domain, VAR deletion_occurred)

This algorithm has the same structure as that of 'AC 1/5 OUT', however the 'source' and 'target' in the revise call have been swapped. The result is demonstrated in the figure above, with the arrows reversed.

Note that in AC 1/5 OUT, the code inside the procedure could have been replaced by a call to rrevise with target-set containing the indices 1 to k-1.

AC 1/4

PROCEDURE AC 1/4 IN(k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = k
WHILE ((s < n) AND (not empty_source_domain)) DO
   revise(s, k-1, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

AC1/4 (see Figure 4.15 revises the arcs (s, k-1), $k \le s \le n$, once each. In other words, given a target node k-1, it revises the arcs with sources indexed with an index greater than k-1, i.e. arcs (k-1, k), (k-1, k+1), ... , (k-1, n).

PROCEDURE AC 1/4 OUT(k, VAR d, VAR empty_source_domain, VAR deletion_occurred)

Again, this algorithm has the same structure as that of 'AC 1/4 IN', however the 'source' and 'target' in the revise call are swapped. The result is demonstrated in the figure above, with the arrows reversed.

Just as in AC 1/5 OUT one may show that the code inside the procedure AC 1/4 OUT can be replaced by a call to rrevise with target-set containing the indices k to n.

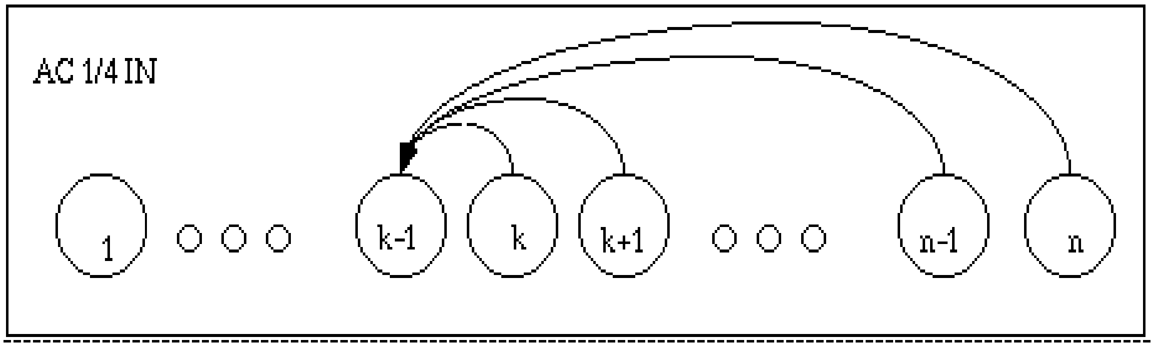


Figure 4.15: AC1/4 behaviour.

AC 9/20

The work done by a AC 1/4 and a AC 1/5 algorithm may be sequenced appropriately to construct what are called the AC 9/20 algorithms. By summing 1/4 and 1/5, we obtain 9/20, thus the name AC 9/20.

For example, if AC 1/5 OUT(k) and AC 1/4 OUT(k+1) are performed we could obtain arc consistency between the node k and all other nodes.

AC 1/3

PROCEDURE AC 1/3 FORWARD(k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = n - 1
WHILE ((s > k-1) AND (not empty_source_domain)) DO
   rrevise(s, s+1 to n, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s - 1

END;
```

AC 1/3 FORWARD (see Figure 4.16) revises the arcs (s, t), $k \leq s$ ¡ $t \leq n$, once each. In other words, given k, it revises all the arcs such that source and target indexes are between k and n and such that the target index is greater than the source index, i.e. in order of revision the arcs (n-1, n), ... , (n-2, n-1), ..., (n-2, n),(k+1, k+2), ... , (k+1, n), (k, k+1), ..., (k, n).

PROCEDURE AC 1/3 BACKWARD (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = k
```

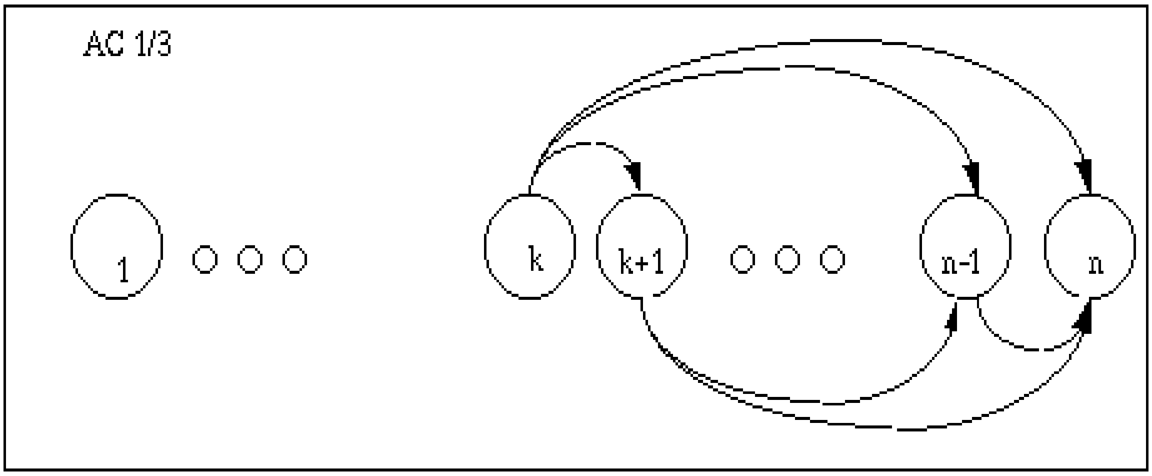


Figure 4.16: AC1/3 behaviour.

```
WHILE ((s < n) AND (not empty_source_domain)) DO
   rrevise(s+1, k to s, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

AC 1/3 BACKWARD (see Figure 4.16) revises the arcs (s,t), k $\leq$ t ¡ s $\leq$ n, once each. In other words, given k, it revises all the arcs such that source and target indexes are between k and n and such that the target index is smaller than the source index, i.e. in order of revision, the arcs (k+1, k), (k+2, k), (k+2, k+1), ... , (n, k), , ..., (n, n-1).

One may notice that the code for the AC 1/3 FORWARD and BACKWARD algorithms can be replaced by a set of calls to the AC 1/4 IN or OUT algorithms.

AC 1/2

PROCEDURE AC 1/2 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred $\leftarrow$ false

```
s = k
WHILE ((s $\leq$ n) AND (not empty_source_domain)) DO
   rrevise(s, k to n but not s, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

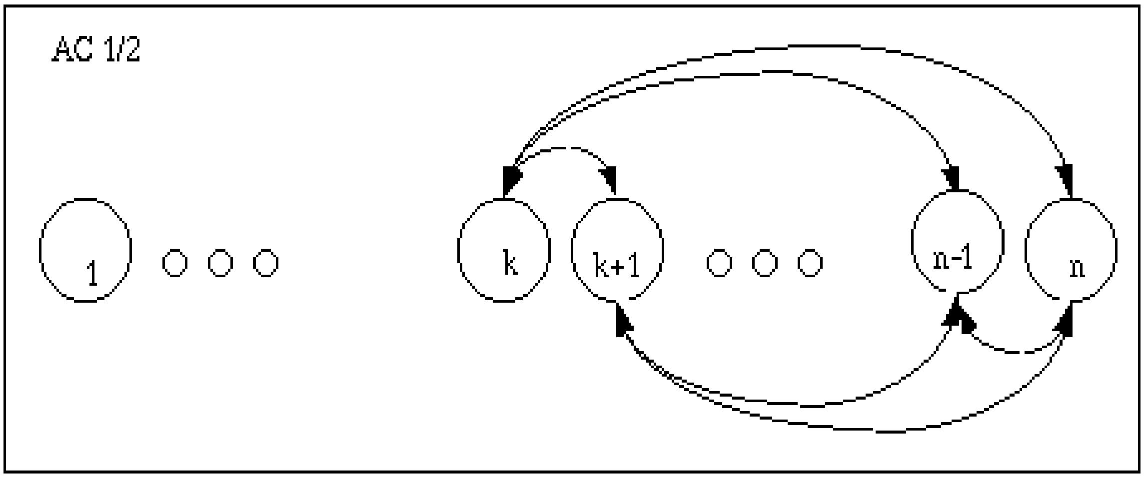


Figure 4.17: AC1/2 behaviour.

AC 1/2 (see Figure 4.17 revises the arcs (s, t), k $\leq$ s $\neq$ t $\leq$ n, once each. In other words, given k, it revises all the arcs such that source and target indexes are between k and n, i.e. arcs (k, k+1), ... , (k, n), (k+1, k), (k+1, k+2), ... , (k+1, n), ..., (n, k), ... (n, n-1).

Again here in the code for AC 1/2, one can show that its code can be replaced by a set of calls to the AC 1/4 or AC 1/3 algorithms. Depending on the type of CSP to be solved the order of the calls to these algorithms could be determined to optimize the amount of domain value deletions in a single call to AC 1/2.

4.3 Full arc consistency algorithms as functions of partials

Introduction

In [49], Nadel showed that the most straightforward way to insure arc consistency (AC) of a subnetwork $G_{k:n}$ is to simply repeat the AC1/2(k) algorithm until it returns false on the deletion_occurred field. Essentially, Nadel shows that a full AC algorithm can be defined as a function of the AC1/2 algorithm.

The purpose of this section to show that other partial AC algorithms are in fact functions of variations of other AC algorithms. The section concludes by proving that a full AC algorithm can be defined as a function of the simplest partial AC algorithms, the AC1/4 and AC1/5 algorithms.

It is assumed that the reader is familiar with the fractional AC algorithms introduced in [49] and Section 4.3 of this report, and Constraint Satisfaction Problems (CSPs) in general.

Notation

Suppose we have a CSP with n variables and that they are indexed 1 to n. In calls to Fractional AC algorithms, k is a parameter that defines an index of a variable. For example, AC 1/5 (k, other parameters) or simply AC 1/5(k) is a call to a fractional AC algorithm were the variable indexed k plays an essential role in the set of revisions performed by the algorithm.

A revision of an arc (s, t), is a revision of the variable indexed s (source) with the variable indexed t (target).

When Nadel defines the work done by the algorithms defined in [49], he always mentions that the arcs are revised in "lexographical order". This means that the pairs (i, j) appear left to right in the arc revision list with j ranging over all its values (except j = i) in order before the next i value is used.

In this paper, "lexographical order" is ignored for the more efficient ordering described in Section 4.4. In other words, the algorithms described in [49] may be replaced by a similar more efficient algorithm performing the same set of revisions but in a more efficient order.

Proposition : AC 1/2 = f(AC 1/3)

The function call AC 1/2(k) revises the set of arcs $\{(s, t) : k \leq s \neq t \leq n\}$, once each. The function call AC 1/3 FORWARD(k) revises the set of arcs $\{(s, t) : k \leq s \text{ ¡ } t \leq n\}$, once each. This algorithm is defined in Section 4.4 which essentially does the same revisions as AC 1/3 in [49], but in a more efficient manner. The function call AC 1/3 BACKWARD(k) revises the set of arcs $\{(s, t) : k \leq t \text{ ¡ } s \leq n\}$, once each. Taking the union of the two latter sets, we obtain the set described for AC 1/2(k). Therefore, a call to AC 1/2(k) can be replaced by calls to AC 1/3 BACKWARD(k) and AC 1/3 FORWARD(k).

Propostion : AC 1/3 = f(AC 1/4)

The AC 1/4 IN(t) algorithm (AC 1/4 in Section 4.3) can be defined to revise the set of arcs $\{(s, t) : t \text{ ¡ } s \leq n\}$, once each. By letting t range over values k to n-1, we see that the union of the sets $\{(s, t) : t \text{ ¡ } s \leq n\}$, $k \leq t \leq n$ is equal to the set defined for AC 1/3 BACKWARD(k). Therefore, a call to AC 1/3 BACKWARD(k) can be replaced by calls to AC 1/4 IN.

We may obtain a similar result with AC 1/3 FORWARD and AC 1/4 OUT.

Corollary : AC 1/2 = f(AC 1/4)

AC 1/2 can be replaced by calls to AC 1/4 IN and AC 1/4 OUT.

Corollary : AC1 = f(AC 1/4)

From the results above, we obtain the following :

AC1 = f(AC 1/2) = f(f(AC 1/3)) = f(AC 1/3) = f(f(AC 1/4)) = f(AC 1/4)

where f(Z) denotes 'function of Z' and is not meant to be a consistent function.

Therefore, AC1 can be replaced by a repetition of iterations of calls to AC 1/4 IN and AC 1/4 OUT. The algorithm AC1 is redefined in the following algorithm. The algorithm guarantees full AC on the subnetwork $G_{k:n}$.

```
NEW AC1 (k, VAR domain, VAR empty_domain)

repeat
   for i = n-1 downto k
       AC1/4 OUT(i, VAR domain, VAR empty_domain, VAR deletion_occurred)
   end for
   for i = k to n-1
       AC 1/4 IN(i, VAR domain, VAR empty_domain, VAR deletion_occurred)
   end for
until not deletion_occurred
```

Corollary : AC1 = f(AC 1/5)

In a similar way, if we were to suppose that AC1 guaranteed full AC on the subnetwork $G_{1:k}$, then it can easily be shown that AC1 can be replaced by calls to AC 1/5 IN and AC 1/5 OUT.

Discussion

These findings allow us to see AC algorithms in a different way. It has been shown that AC1 can represented by the most primitive of fractional AC algorithms.

One can take advantage of these findings to increase the efficiency of a full arc consistency algorithm. This result is very implicit. By using the findings in Section 4.3, one can reorder the revisions done by the original AC1 algorithm to increase the efficiency of the algorithm. The algorithm above (NEW AC1) was designed to do this.

4.4 Revision of inefficiency in a specific arc consistency algorithm

Introduction

A large number of problems in Artificial Intelligence and other areas of Computer Science can be viewed as special cases of constraint satisfaction problems (CSP). Arc consistency algorithms are an important class of algorithms used for solving CSP as they can simplify the problem.

This section reviews a Fractional Arc Consistency algorithm called AC1/3 found in [49]. An inefficiency within this algorithm is noted and solution to eliminate this problem is explained.

Constraint Satisfaction Problem (CSP)

A CSP is defined by three components :

- a finite set of variables V, where $V = \{V_i : 1 \leq i \leq n\}$,
- a finite and discrete domain D, where $D = \{(d_1, d_2, ..., d_n) : d_i \in D_i, 1 \leq i \leq n\}$, and $D_i = \{\text{possible instances of } V_i\}$, and
- a set of constraints C, where $C = \{C_{ij} : 1 \leq \mathrm{i}, \mathrm{j} \leq \mathrm{n}\}$ and C_{ij} is a constraint[6] between V_i and V_j.

The goal of a CSP is to find assignments of the variables to their candidate values such that all the constraints are satisfied.

Consistency

Definition: Consistency

A set of variables are said to be consistent if all the constraints among these variables can be satisfied.

The notion of consistency exists in varying degrees determined roughly by the number of variables within the given set.

Definition: Arc Consistency

Let V_i and V_j be variables from a CSP. An arc (V_i, V_j) is arc consistent if for every value χ in the current domain of V_i, there is some value γ in the domain of V_j such that $V_i = \chi$ and $V_j = \gamma$ are permitted by the binary constraint between 'V_i' and 'V_j'.

[6] it is assumed that all constraints are binary.

Note that the concept of arc consistency is directional (and not always a reflexive relation); that is if an arc (V_i, V_j) is consistent, then this does not imply that the arc (V_j, V_i) is also consistent.

Revision

An arc (source, target) can made consistent by simply deleting those values from the domain of 'source' for which the previous condition is not true. Arc consistency algorithms use this method of revision typically implemented through a function called revise.

Because of this revision, an AC algorithm can be thought of as a simplification algorithm which transforms the original problem into a simpler version that has the same solutions. In some cases the resulting problem is so simple that the solutions (or lack thereof) become manifest and the original problem is solved.

A call to the revise function may appear as the following:

revise (source, target, VAR d, VAR empty_source_domain, VAR change).

The reference array parameter d contains current domains. The variables empty_source_domain and change are included as reference parameters to inform the calling routine respectively whether the domain of source was totally depleted and whether a deletion occurred at all.

There is also another function called rrevise that revises a source over a set of targets. For example a call to rrevise may appear as the following :

rrevise (source, target_set, VAR d, VAR empty_source_domain, VAR change).

Arc Consistency Algorithms

There are currently two types of arc consistency algorithms.; these are full and fractional (or partial) arc consistency.

Full Arc Consistency Algorithms

A variable network is said to be arc consistent when each arc is consistent. An algorithm that guarantees arc consistency of the (sub)network on the nodes to which it applies is said to be a full arc consistency algorithm for that (sub)network. Many full arc consistency algorithms exist, however research has concluded that it is more efficient to use hybrid tree search/AC algorithms that use fractional AC to solve CSP.

Fractional Arc Consistency Algorithms

Arc consistency algorithms that revise various subsets of arcs in a variable network are called fractional arc consistency (ACq) algorithms. The suffix q represents a fraction, and is roughly proportional to the degree of fractional arc consistency that it achieves.

An inefficiency within the AC1/3 algorithm presented in [49] is noted and a solution to eliminate this problem is explained.

Inefficiency

Ordering of Revisions

Consider the subset of variables $V_1, V_2, V_3, ..., V_n$ of a CSP. Now suppose that the revisions from the following set are to be performed : $(V_1, V_2), (V_2, V_3), ..., (V_n - 1, V_n)$.

These revisions may be performed in the orders demonstrated in following diagrams, Figures 4.18 and 4.19. The numbers on the arc represent the order in which the arcs were revised.

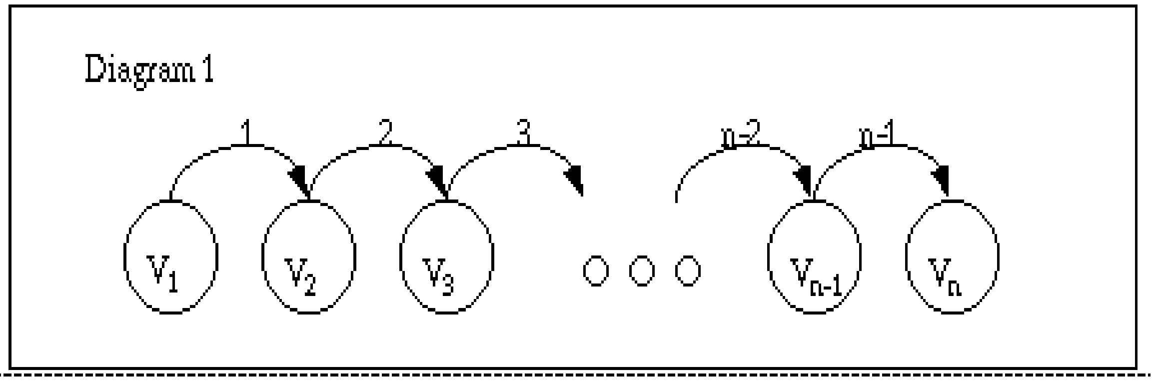


Figure 4.18: Inefficient revision ordering.

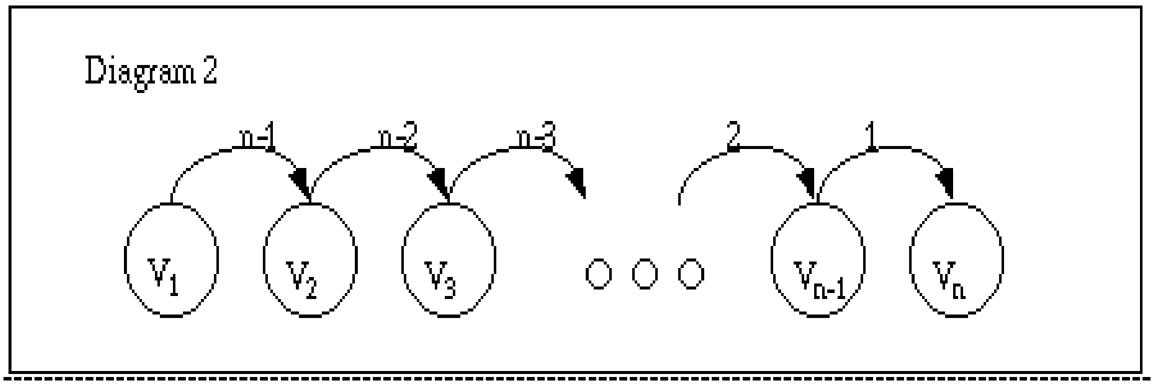


Figure 4.19: Efficient revision ordering.

In Figure 4.18, the revision of the arc $(V_k, V_k + 1)$ comes before that of $(V_k + 1, V_k + 2)$, 1 $\leq$ k $\leq$ n-2. Once revise reduces the domain of some variable V_j, then the previously revised arcs $(V_i - 1, V_i)$, i $\leq$ j, may no longer be consistent.

The ordering of revisions in Figure 4.14 avoids the inefficiency of the revisions in Figure 4.18. This ordering can either eliminate more redundant revisions from continued iterations of these revisions, or result in a higher degree of arc consistency in one iteration.

This method also can also reduce the work done in each revision as the size of the target domain for each call may be reduced by the previous revision. For example the revision of (V_k+1, V_k+2) may reduce the source domain V_k+1, which then becomes the target domain for the revision of (V_k, V_k+1).

Algorithms

The savings obtained from this method are demonstrated again in the algorithm [49] :

PROCEDURE AC 1/3 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = k
WHILE ((s < n) AND (not empty_source_domain)) DO
   rrevise(s, s+1 to n, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s + 1

END;
```

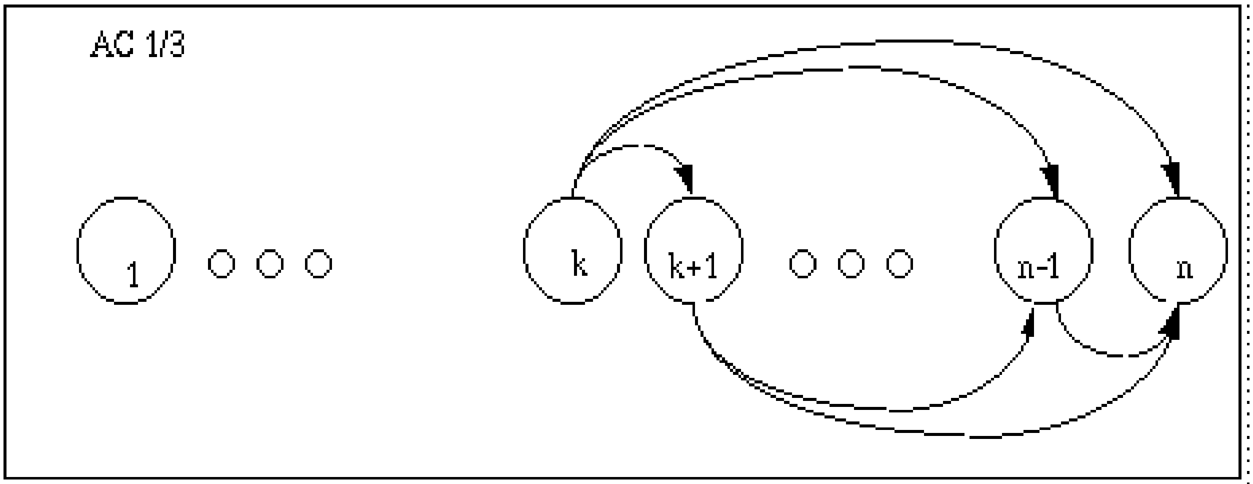


Figure 4.20: AC1/3 behaviour.

AC 1/3 (see Figure 4.20) revises the arcs (s, t), k ≤ s ≤ t ≤ n, once each in the following order : (k, k+1), ... , (k, n), (k+1, k+2), ... , (k+1, n), ..., (n-1, n). The procedure has been optimized by reordering the values that the source counter takes on.

PROCEDURE NEW AC 1/3 (k, VAR d, VAR empty_source_domain, VAR deletion_occurred);
empty_source_domain, deletion_occurred ← false

```
s = n - 1
```

```
WHILE ((s > k-1) AND (not empty_source_domain)) DO
   rrevise(s, s+1 to n, d, empty_source_domain, deletion)
   deletion_occurred = deletion_occurred or deletion
   s = s - 1

END;
```

AC 1/3 (See Figure 4.20) revises the arcs (s, t), $k \leq s \leq t \leq n$, once each in the following order : (n-1, n), ... , (n-2, n-1), ..., (n-2, n),(k+1, k+2), ... , (k+1, n), (k, k+1), ..., (k, n).

Conclusion

The solution presented to this problem is good in the sense that if the AC1/3 algorithm were to be used in an application. It is unlikely that this algorithm will ever be implemented as it has been shown that other algorithms, such as the FC (or TSAC1/4) hybrid tree search/arc consistency algorithm also found in [49], avoid this problem by using another fractional arc consistency algorithm called AC1/4.

4.5 Effective fractional arc consistency example calls in "4-Queens" problem

Introduction

This section gives examples of calls to fractional arc consistency algorithms that (given proper parameters) reduce the domain of a variable. In other words, these calls actually delete values from a domain due to carefully chosen domains and input parameters.

The purpose of highlighting these examples is to show what kind of fractional arc consistency work is useful during preprocessing of a CSP.

For simplicity, the CSP used is that of the Confused 4-Queens Puzzle. Proper parameters for effective arc revisions are also very easy to find using this CSP.

The definitions of the arc consistency algorithms are found in Section 4.2.

Notation

The domain array for the Confused 4-Queens Puzzle will be expressed as a list that is used for the fractional arc consistency functions implemented in LISP. An example of such a list with the original unrevised domains is given in the following:

```
(((row 1) 1 2 3 4)
 ((row 2) 1 2 3 4)
```

```
 ((row 3) 1 2 3 4)
 ((row 4) 1 2 3 4) )
```

After each example of an AC call, results are presented to summarize the number of calls to the revise algorithm, the number of constraint checks and the number of deletions.

Constraint

Suppose (row A) and (row B) are variables of a Confused 4-Queens Puzzle and A ¡¿ B. These two variables must meet the following constraint :

abs(A - B) = abs(DV(A) - DV(B))

where and DV(A) and DV(B) are domain values.

Hence in a call to the 'revise' function during a call to an arc consistency function, with A as the source and B as the target, if any domain value of A cannot find at least one domain value in of B that satisfies the constraint then this domain value of A will be deleted from A's domain.

Effective Calls to ACq Algorithms

Calls to fractional arc consistency algorithms are presented in the following sections. The input and output domain array are shown as well as statistics that show the number of variable revisions, constraint checks and deletions of domain values were performed.

AC 1/5 IN

The input for the domain array is given in the following lines.

```
(((row 1) 1 2 3 4)
 ((row 2) 1 2 3 4)
 ((row 3) 1 2 3 4)
 ((row 4) 4       ))
```

The effective call to the algorithm uses (row 4) as the index parameter and is given here: AC 1/5 IN ((row 4), Domain Array).

The output is shown in the following.

```
(((row 1) 1 4)
 ((row 2) 2 4)
```

```
 ((row 3) 3 4)
 ((row 4) 4  ))
```

Statistics are presented in the following lines.

```
Calls to Revise : 3
Checks  : 12
Deletions : 6
```

AC 1/5 IN revised the first 3 (unrevised and uninstantiated) variables with respect to the instantiated variable (row 4). The fact that (row 4) was instantiated allowed for many values to be deleted from the domains.

If a second value, say 3, had been included in the domain of (row 4), then no deletions would have occurred.

AC 1/5 OUT

The input for the domain array is given in the following lines.

```
(((row 1) 1       )
 ((row 2) 2       )
 ((row 3) 3       )
 ((row 4) 1 2 3 4))
```

The effective call to the algorithm uses (row 4) as the index parameter and is given here: AC 1/5 OUT ((row 4), Domain Array).

The output is shown in the following.

```
(((row 1) 1)
 ((row 2) 2)
 ((row 3) 3)
 ((row 4) 4))
```

Statistics are presented in the following lines.

```
Calls to Revise : 3
Checks  : 9
Deletions : 3
```

AC 1/5 OUT revised the (unrevised and uninstantiated) variable (row 4) with respect to the instantiated variables (row 1), (row 2) and (row 3). The fact that variables preceding (row 4) were instantiated allowed for the three values to be deleted from the domain of (row 4).

All variables are now instantiated, consistent and yield a solution to the puzzle.

AC 1/4 IN

The input for the domain array is given in the following lines.

```
(((row 1) 1      )
 ((row 2) 1 2 3 4)
 ((row 3) 1 2 3 4)
 ((row 4) 1 2 3 4))
```

The effective call to the algorithm uses (row 1) as the index parameter and is given here: AC 1/4 IN((row 1), Domain Array).

The output is shown in the following.

```
(((row 1) 4  )
 ((row 2) 3 4)
 ((row 3) 2 4)
 ((row 4) 1 4))
```

Statistics are presented in the following lines.

```
Calls to Revise : 3
Checks  : 12
Deletions : 6
```

AC 1/4 IN revised the (unrevised and uninstantiated) variables (row 2), (row 3) and (row 4) with respect to the instantiated variable (row 1). This is very similar to AC 1/5 IN but with the order variables reversed.

A different instantiation of (row 1) would have yielded a different result. For example if (row 1) had 3 for its domain, then the result would be:

```
(((row 1) 3     )
 ((row 2) 2 3 4)
 ((row 3) 1 3  )
 ((row 4) 3    ))
```

AC 1/4 OUT

The input for the domain array is given in the following lines.

```
(((row 1) 1 2 3 4)
 ((row 2) 2      )
 ((row 3) 3      )
 ((row 4) 4      ))
```

The effective call to the algorithm uses (row 1) as the index parameter and is given here: AC 1/4 OUT((row 1), Domain Array).

The output is shown in the following.

```
(((row 1) 1)
 ((row 2) 2)
 ((row 3) 3)
 ((row 4) 4))
```

Statistics are presented in the following lines.

```
Calls to Revise : 3
Checks  : 9
Deletions : 3
```

AC 1/4 OUT revised the (unrevised and uninstantiated) variable (row 1) with respect to the instantiated variables (row 2), (row 3) and (row 4). The fact that variables following (row 1) were instantiated allowed for the three values to be deleted from the domain of (row 1).

All variables are now instantiated, consistent and yield a solution to the puzzle. This example is very similar to AC 1/5 OUT but with the order of the variables reversed.

AC 1/3 BACKWARD

The input for the domain array is given in the following lines.

```
(((row 1) 3 4  )
 ((row 2) 1 2  )
 ((row 3) 3 4  )
 ((row 4) 1 2 3))
```

The effective call to the algorithm uses (row 1) as the index parameter and is given here: AC 1/3 IN((row 1), Domain Array).

The output is shown in the following.

```
(((row 1) 3 4)
 ((row 2) 2  )
 ((row 3) 3  )
 ((row 4)    ))
```

Statistics are presented in the following lines.

```
Calls to Revise : 6
Checks  : 19
Deletions : 5
```

It can be shown that the AC 1/3 IN algorithm can be constructed by an iteration of calls to AC 1/4 IN. These calls are listed in order with the deleted domain values:

AC 1/4 IN((row 1), Domain Array), ((row 4) 2), ((row 2) 1)

AC 1/4 IN((row 2), Domain Array), ((row 4) 1 3), ((row 3) 4)

AC 1/4 IN((row 3), Domain Array), nil, domain of row 4 has been wiped out

If (row 1) had one less domain value, the number of checks would have been reduced from 19 to 9 with 5 deletions.

AC 1/3 FORWARD

The input for the domain array is given in the following lines.

```
(((row 1) 1 2 3)
 ((row 2) 3 4  )
 ((row 3) 1 2  )
 ((row 4) 3 4  ))
```

The effective call to the algorithm uses (row 1) as the index parameter and is given here: AC 1/3 OUT((row 1), Domain Array).

The output is shown in the following.

```
(((row 1)    )
 ((row 2) 3  )
 ((row 3) 2  )
 ((row 4) 3 4))
```

Statistics are presented in the following lines.

```
Calls to Revise : 6
Checks  : 19
Deletions : 5
```

It can be shown that the AC 1/3 IN algorithm can be constructed by an iteration of calls to AC 1/4 IN. These calls are listed in order with the deleted domain values:

AC 1/4 OUT((row 3), Domain Array), ((row 3) 1)
AC 1/4 OUT((row 2), Domain Array), ((row 2) 4)
AC 1/4 OUT((row 1), Domain Array), ((row 1) 1 2 3)

AC 1/2

The input for the domain array is given in the following lines.

```
(((row 1) 1 2 3)
 ((row 2) 3 4  )
 ((row 3) 1 2  )
 ((row 4) 3 4  ))
```

The effective call to the algorithm uses (row 4) as the index parameter and is given here: AC 1/2((row 4), Domain Array).

The output is shown in the following.

```
(((row 1))
 ((row 2))
 ((row 3))
 ((row 4)))
```

Statistics are presented in the following lines.

```
Calls to Revise : 12
Checks  : 23
Deletions : 7
```

The AC 1/2 algorithm can be expressed as a call to the two AC 1/3 algorithms. Using the same domain array as AC 1/3 OUT, we see that if we use the AC 1/3 OUT algorithm as the first AC 1/3 call, the domain is reduced to the following:

```
(((row 1)    )
 ((row 2) 3  )
 ((row 3) 2  )
 ((row 4) 3 4))
```

Applying the AC 1/3 IN algorithm we get a complete domain wipeout (all domain values are deleted).

4.6 Effective fractional arc consistency example calls in "Relative Position" problem

Introduction

This section gives examples of calls to each fractional arc consistency algorithm that (given proper parameters) reduce the domain(s) of a variable(s). In other words, these calls actually delete values from a domain due to carefully chosen domains and input parameters.

The purpose of highlighting these examples is to show what kind of fractional arc consistency work is effective during preprocessing of a CSP.

Notes

In this paper we use the Relative Position CSP defined in Section 4.1.

In the Relative Position CSP, the variables A, B, C and D are equivalent to 1, 2, 3, and 4 respectively.

The definitions of the arc consistency algorithms can be found in Section 4.2.

Definition of Relative Position CSP

Consider a grid of size 8 by 8. There is a set of variables $V = \{V_i : 1 \leq i \leq 4\} = \{A, B, C, D\}$. Here we have replaced the variable array by the first letters of the alphabet. Each variable takes on a coordinate value (x, y).

The constraint network and domain array representations are abandoned for the following method to give a better visual representation of the problems domain.

The domains are shown on the grid map in Figure 4.21. Rather than assigning a value (instance) to a variable, variables are assigned to the values.

8								
7					B	B	B	B
6			A					
5	A	A		A				
4	D							C
3		D					C	
2			D			C		
1				D	C			
	1	2	3	4	5	6	7	8

Figure 4.21: Variable domain in Relative Position problem.

The constraints are shown in Figure 4.22.

Effective Calls to ACq Algorithms

Calls to fractional arc consistency algorithms are presented in the following sections. The input and output domain array are shown as well as statistics that show the number of variable revisions, constraint checks and deletions of domain values were performed.

AC 1/5 IN

The input for the domain array is given in Figure 4.23.

$$C(A,B) = (3 \le (Bx - Ax) \le 4) \wedge (-1 \le (By - Ay) \le 1)$$

$$C(A,C) = (3 \le (Cx - Ax) \le 6) \wedge (3 \le (Ay - Cy) \le 6)$$

$$C(A,D) = (3 \le (Ay - Dy) \le 6) \wedge (-1 \le (Dy - Ay) \le 1)$$

$$C(B,C) = (3 \le (By - Cy) \le 5) \wedge (-1 \le (Bx - Cx) \le 1)$$

$$C(D,C) = (3 \le (Cx - Dx) \le 5) \wedge (-1 \le (Dy - Cy) \le 1)$$

Figure 4.22: Constraints in Relative Position problem.

The effective call to the algorithm uses D as the index parameter and is given here: AC 1/5 IN (D, Domain Array).

D is chosen as the call will revise all the lower indexed variables (A, B and C) which are unrevised and uninstantiated. The output is shown in Figure 4.24.

Statistics are presented in the following lines.

```
Calls to Revise : 2
Checks  : 8
Deletions  : 5
```

No values were deleted from B since there exists no constraint between B and D.

AC 1/5 OUT

The input for the domain array is given in Figure 4.25.

The effective call to the algorithm uses D as the index parameter and is given here: AC 1/5 OUT (D, Domain Array).

D is chosen as the call will revise D with all other lower indexed variables (A, B and C) which are instantiated. The output is shown in Figure 4.26.

The output is shown in Figure 4.26.

Statistics are presented in the following lines.

```
Calls to Revise : 2
```

8								
7					B	B	B	B
6			A					
5	A	A		A				
4								C
3		D					C	
2						C		
1					C			
	1	2	3	4	5	6	7	8

Figure 4.23: Variable domains before AC 1/5 IN call.

8								
7					B	B	B	B
6			A					
5								
4								
3		D					C	
2						C		
1								
	1	2	3	4	5	6	7	8

Figure 4.24: Variable domains after AC 1/5 IN call.

```
Checks  : 6
Deletions  : 2
```

All values of the variables are consistent and yield a solution to the problem.

AC 1/4 IN

The input for the domain array is given in Figure 4.27.

The effective call to the algorithm uses A as the index parameter and is given here: AC 1/4 IN(A, Domain Array).

A is chosen as the call will revise the greater indexed variables (B, C and D) (which are unrevised and uninstantiated) with the variable A. The output is shown in Figure 4.28.

8								
7					B	B	B	B
6			A					
5								
4	D							
3		D					C	
2			D					
1				D				
	1	2	3	4	5	6	7	8

Figure 4.25: Variable domains before AC 1/5 OUT call.

8								
7					B	B	B	B
6		A						
5								
4								
3		D					C	
2			D					
1								
	1	2	3	4	5	6	7	8

Figure 4.26: Variable domains after AC 1/5 OUT call.

Statistics are presented in the following lines.

```
Calls to Revise : 3
Checks  : 12
Deletions  : 5
```

AC 1/4 OUT

The input for the domain array is given in Figure 4.29.

The effective call to the algorithm uses A as the index parameter and is given here: AC 1/4 OUT(A, Domain Array).

8								
7					B	B	B	B
6			A					
5								
4	D							C
3		D					C	
2			D			C		
1				D	C			
	1	2	3	4	5	6	7	8

Figure 4.27: Variable domains before AC 1/4 IN call.

8								
7						B	B	
6			A					
5								
4								
3		D					C	
2			D			C		
1				D				
	1	2	3	4	5	6	7	8

Figure 4.28: Variable domains after AC 1/4 IN call.

A is chosen as the call will revise A with all other lower indexed variables (B, C and D) which are instantiated. The output is shown in Figure 4.30.

Statistics are presented in the following lines.

```
Calls to Revise : 3
Checks  : 6
Deletions  : 3
```

Note that the revision between A and B performed all the deletions and left the domain array consistent. The other revisions had no effect on the domains, but the AC1/4 call was still quite effective.

8								
7						B		
6			A					
5	A	A		A				
4								
3								
2			D			C		
1								
	1	2	3	4	5	6	7	8

Figure 4.29: Variable domains before AC 1/4 OUT call.

8								
7						B		
6			A					
5								
4								
3								
2			D			C		
1								
	1	2	3	4	5	6	7	8

Figure 4.30: Variable domains after AC 1/4 OUT call.

AC 1/3 BACKWARD

The input for the domain array is given in Figure 4.31.

The effective call to the algorithm uses A as the index parameter and is given here: AC 1/3 BACKWARDS(A, Domain Array).

The output is shown in Figure 4.32.

Statistics are presented in the following lines.

```
Calls to Revise : 5
Checks  : 17
Deletions  : 6
```

8								
7					B	B	B	B
6			A					
5								
4	D							C
3		D					C	
2			D			C		
1				D	C			
	1	2	3	4	5	6	7	8

Figure 4.31: Variable domains before AC 1/3 BACKWARDS call.

8								
7					a	B	B	a
6			A					
5								
4	a							a
3		D					C	
2			D			C		
1				c	a			
	1	2	3	4	5	6	7	8

Figure 4.32: Variable domains after AC 1/3 BACKWARDS call.

It can be shown that the AC 1/3 BACKWARDS algorithm can be constructed by an iteration of calls to AC 1/4. Rather than leaving an empty space, deletions are expressed by the variable parameter k used in calls to AC 1/4 IN.

AC 1/3 FORWARD

The input for the domain array is given in Figure 4.33.

The effective call to the algorithm uses A as the index parameter and is given here: AC 1/3 OUT(A, Domain Array).

The output is shown in Figure 4.37.

Statistics are presented in the following lines.

8								
7					B	B	B	B
6			A					
5	A	A		A				
4								C
3		D					C	
2						C		
1					C			
	1	2	3	4	5	6	7	8

Figure 4.33: Variable domains before AC 1/3 FORWARDS call.

8								
7					c	B	B	B
6			A					
5	d	d		d				
4								d
3		D					C	
2						C		
1					d			
	1	2	3	4	5	6	7	8

Figure 4.34: Variable domains after AC 1/3 FORWARDS call.

```
Calls to Revise : 5
Checks  : 19
Deletions  : 6
```

It can be shown that the AC 1/3 IN algorithm can be constructed by an iteration of calls to AC 1/4 IN. Again, rather than leaving an empty space, deletions are expressed by the variable parameter k used in calls to AC 1/4 OUT.

AC 1/2

The input for the domain array is given in Figure 4.35.

8								
7					B	B	B	B
6			A					
5	A	A		A				
4								C
3		D					C	
2						C		
1					C			
	1	2	3	4	5	6	7	8

Figure 4.35: Variable domains before AC 1/2 call.

The effective call to the algorithm uses A as the index parameter and is given here: AC 1/2(D, Domain Array).

The output is shown in Figure 4.36.

8								
7						B	B	
6			A					
5								
4								
3		D					C	
2						C		
1								
	1	2	3	4	5	6	7	8

Figure 4.36: Variable domains after AC 1/2 call.

Statistics are presented in the following lines.

```
Calls to Revise : 10
Checks  : 29
Deletions  : 7
```

The AC 1/2 algorithm can be expressed as a call to the two AC 1/3 algorithms. Using the same domain array as AC 1/3 FORWARD, we see that if we use the AC 1/3 FORWARD algorithm as the first AC 1/3 call, the domain is reduced to Figure 4.37.

8								
7					c	B	B	B
6			A					
5	d	d		d				
4								d
3		D					C	
2						C		
1					d			
	1	2	3	4	5	6	7	8

Figure 4.37: Variable domains after second AC 1/3 FORWARD call.

Applying the AC 1/3 IN algorithm we obtain the result in Figure 4.35.

Chapter 5

Upward exploitation of constraint propagation

5.1 Introduction

One method currently proposed for the solution of constraint satisfaction problems (CSPs) is the coupling of traditional breadth-first or depth-first search through the space of all possible variable assignments with some degree of constraint propagation before or during search. Hybrid strategies interleave the work spent on search and the work spent reducing variable domains through some kind of constraint propagation algorithm(s).

Hybrids have been constructed based upon well known search strategies including traditional backtracking, forward checking, backjumping, and backmarking. All of the hybrids utilizing these algorithms have been constructed typically by attempting to achieve some degree of arc consistency during search. For example, forward checking attempts to revise or reduce the domains of remaining variables based upon the most recent instantiated variable value.

It is quite often the case that the work spent in performing constraint propagation is duplicated at various points in the hybrid search space. Essentially identical domain values at disparate points in the search space are removed as a result of identical constraint checks being performed. We present here a brief description of a hybrid-based search strategy that can allow for the sharing of constraint propagation work under certain circumstances. In addition, this strategy can be easily encapsulated in a depth-first structure so as to minimize space requirements of search.

This work is based on ongoing research into constraint satisfaction search and hybrid application of search and constraint propagation techniques.

5.2 Upward Sharing

The general structure of hybrid search is selection of a variable, assignment of a domain value for that variable (instantiation), some constraint propagation intended to simplify the remaining problem, and repeat. This approach can be nicely formed as a complete depth first strategy which finds some solution as quickly as possible, and yet which is capable of finding all solutions. Forward checking for instance operates precisely in this manner, where the hybrid aspect appears in that all other variables are revised or have their domains "reviewed" with respect to the latest assignment, and any values no longer satisfying a constraint between a particular variable and the newly instantiated one are discarded.

If we were to take the decision point for the assignment to some variable, call it A, and consider the various subtrees resulting from different domain value selections for A, we can notice that the resulting problems under each subtree are quite similar to the parent (before A) problem. In fact, they differ only in that A no longer has a domain, but rather a single value. Each of the A subtrees differs only in the choice of domain value for A.

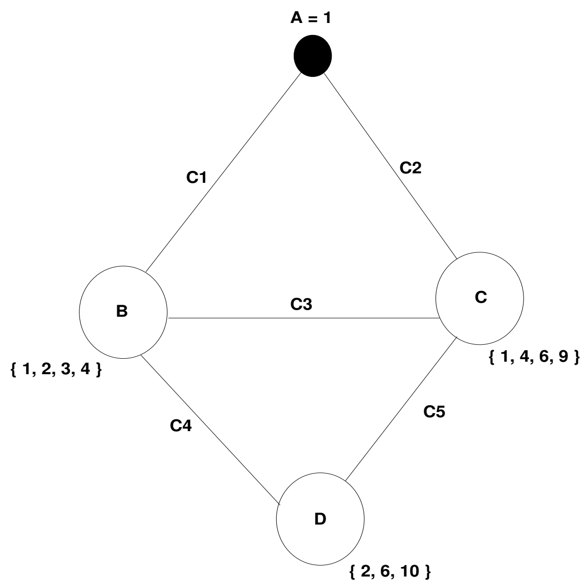


Figure 5.1: Partially Instantiated Constraint Graph.

Figure 5.1 shows a problem with four variables A, B, C and D in which A has been selected for instantiation, and in fact the value assigned to A is 1 in this subtree. If A has a domain of 1, 2 and 3 in the parent, then there will conceivably be three siblings of this nature. In particular, the subgraph consisting of nodes (and domains) B, C, and D plus their associated constraints C3, C4 and C5 will be identical in each of the three subtrees. Further, this subgraph will exist in earlier ancestors of the parent since in our search method we never add variables or domain values as we progressively instantiate variables.

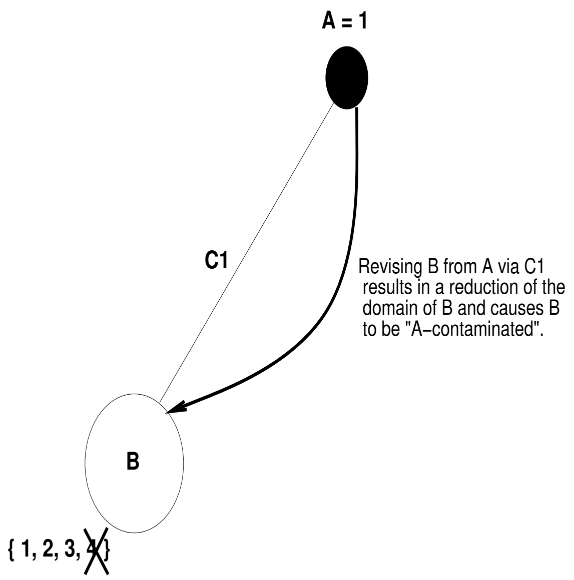


Figure 5.2: Contamination trickling through graph.

Now, if we were to perform some constraint propagation in which the domain of B is reduced based upon the newly selected value of A, we can say B's domain has become "A-contaminated". This is illustrated in Figure 5.2. B's domain has been reduced based upon our new "conjecture" that A's value is 1. Similarly, for any other variable's domain constrained by A, if that domain is reduced based upon an instantiation of A that domain is considered to be "A-contaminated". Now, if D's domain were to be reduced based upon either B or C (each A-contaminated), then D's domain would be said to be A-contaminated also. The basic intuition of contamination in this context is to allow us to be aware of what variable assignment conjectures have been used in our arrival at a particular domain for each variable.

Assume that the order of instantiation in our example is determined prior to any search. For instance, we will say that the order of variable instantiation is always A, B, C and finally D. Therefore, as we progress down our instantiation order and search space, A is instantiated first, and in all subtrees after that instantiation, the assignment of A is a "fact" that can be used in reasoning about other domain values. If we look at the search space as a whole, the variable commitment point spawns child subtrees, where in each subtree the value of that particular fact differs. The parent of these children has no "given" fact for A, only a set of possible domain values.

The concept of contamination allows us to keep track of which "facts" have been used in reducing some possible domain assignments. Since these assignments occur top-down and one at a time, we can see that the point in time where A is assigned always occurs before B's assignment, and further that at any point in the space after B's assignment, A must be assigned to some particular value also. In fact, when an arbitrary variable has just been assigned a particular domain value, all variables earlier in the order have already been assigned, and all variables earlier in the order have a set of possible values.

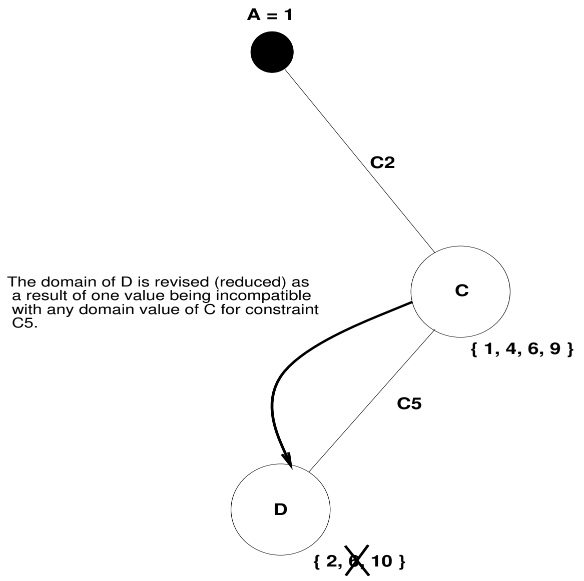


Figure 5.3: Constraint propagation between unbound variables.

If, in a particular instance of constraint propagation during search (say after assigning A=1), we reduced the domain of some variable (say D) based upon the current domain of some other unbound variable (say C) and the constraint set between C and D, then (assuming C is not contaminated) we should be able to "share" this reduction throughout our search space. Figure 5.3 details this reduction. The reasoning that resulted in a reduction of D's domain was not dependent on any instantiation explicitly or implicitly, and this reduction could have as easily been done *prior* to the start of any search.

In a backtracking paradigm, sharing this information can be thought of as propagating the domain reduction upwards in the search space. In this particular example, we have just assigned A=1, and the parent is the original (top) problem formulation without assignments. If we propagate the D reduction to each parent up to and including the top problem, any further successor creations through other instantiations will carry the reduction. The D reduction will therefore have been made locally in the ancestor tree of the node where the constraint propagation occurred, however, it will take global affect as search progresses.

The key to this upward propagation is in the lack of contamination of the domain being propagated. We cannot reduce a domain based upon some variable assignment (directly or indirectly), and propagate this result above the "highest" variable which has taken part in the "contamination". Figure 5.4 and 5.5 show the upward propagation to the highest level. Figure 5.5 shows how this revision of D's domain is later used in a different instantiation of A.

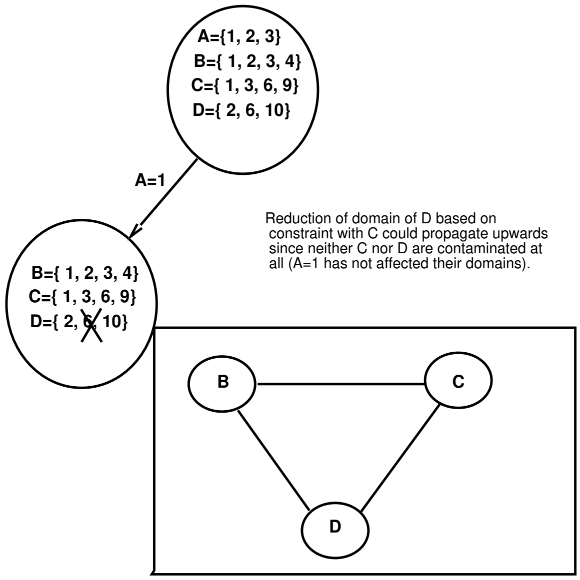


Figure 5.4: Problem space before upward propagation.

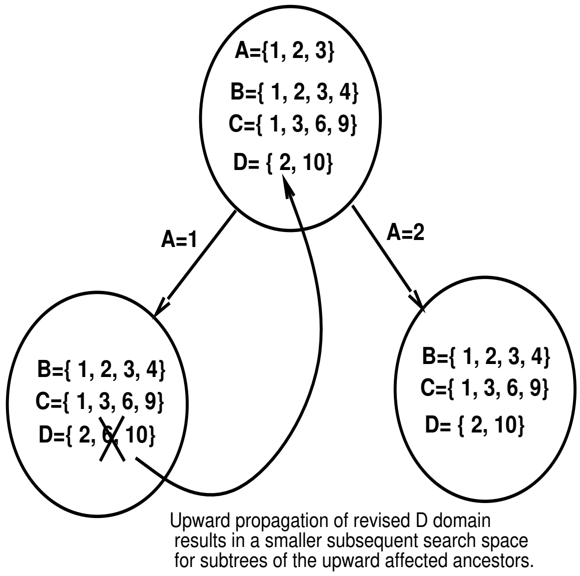


Figure 5.5: Problem space after upward constraint.

For example consider Figure 5.6. If we instantiated A=1 and B=2, then reduce the domain of D based only on C where neither D nor C have been contaminated by either A or B, then we may propagate the new D domain to all parents up to and including the whole problem. All subsequent children resulting from backtracking will include this reduction. If we instantiated A=1, and B=3, then reduced the domain of C based on A's value, then C has become A-contaminated. C's new domain may only propagate up to the level where A was instantiated. All subsequent children of the node where A was instantiated will now reflect the reduced C domain. We must be careful since if we attempt to propagate a contaminated reduction past its point of instantiation, we are essentially carrying inferences based upon assignments past where they occur, and thus would force a reduction in a domain based upon an erroneous assumption, and completeness will have been sacrificed.

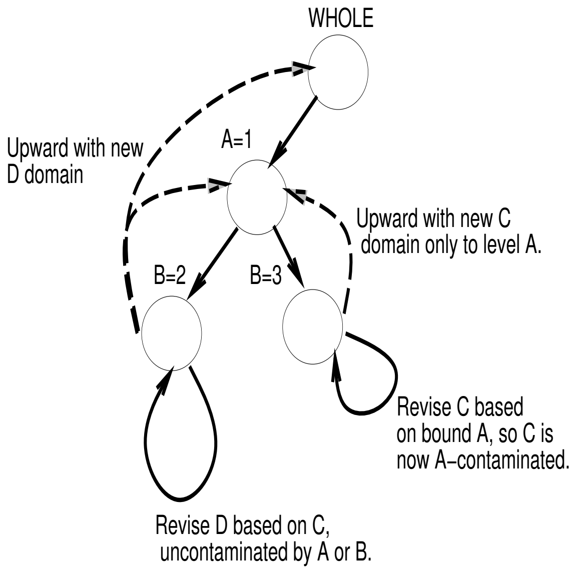


Figure 5.6: Problem space after upward constraint.

5.3 Implications

As a result of propagating this constraints upward in the depth first path where possible, we can reduce the size of the search space in subsequent instantiation paths. However, we should note that the work we are sharing could have been performed as part of more elaborate initial consistency checking prior to the start of search. The advantage of this approach is that we may be uncertain initially as to how much effort to spend on preliminary problem reduction. If some factors appear or are noticed during hybrid search, we can take advantage of these conditions locally via application of further consistency propagation, and achieve global improvement for only the cost of replacing ancestor domains where reduction occurs and where the domain contamination determines upward propagation is permissible.

This approach will result in savings only if:

1. A constraint propagation algorithm(s) is utilized which at some point successfully revises some domain based upon constraints between variables with uninstantiated variables. For this point, the tighter the constraints the more likely it is that at least one value will be removed. If no values are removed, there is nothing to propagate.

2. In at least where the above (1) is successful, we are able to propagate the reduction at least one level above the current instantiated variable. For instance, if we have just instantiated B, then in order for a revision to be useful, we must propagate it to at least A so that A's later children (the current node's siblings), reflect this improvement. Of course, the more levels that a removal is propagated, the greater the potential for savings as the scope of the removal is widened.

Please note also :

1. This method evolved from thinking about how to share constraint propagation *across* the search space in an A or breadth-first search. Further thought will have to be given to this original intent to see to what extent individual node constraint propagtion can be shared in a wider scope than simply depth-first ancestors.

2. Note also that since upward propagations occur along the ancestor path of the current node, that the upward propagation of several domain reductions can combine combinatorially within ancestor nodes to produce larger reduction in the size of the search space expanded following upward propagation. These reductions should form a multiplicative reduction in the search required.

3. Search strategies that do not pre-order the variable instantiations can still benefit from this upward ancestor propagation strategy. Since we are careful to keep track of the "contamination" level of each domain during search, we never propagate an erroneous inference above the level in the space where the "trigger instantiation" was first made.

4. This method seems to relate to propagating hypothetical inferences. If identical or highly similar work has been done, we have been unable to locate it.

5. At a later time we will test this constraint satisfaction hybrid algorithm empirically.

Chapter 6

Decomposing constraint satisfaction problems

In this section we will discuss several advantages and methods of decomposing constraint satisfaction problems, and also ways in which we may control or guiding the recomposition of these separate solutions with user interaction or/and by heuristics.

One drawback with the approach to solving constraint satisfaction problems based upon interleaving search and constraint propagation described in the preceding sections is that a solution or set of solutions is found for a particular CSP in a particular world situation or set of domain values, but those "near" misses for CSP solutions are ignored. Essentially, some particular sets of variable and domain pairs that fails a single constraint, or is unable to match a single variable to a domain value in the world situation does not appear to the user as a candidate solution. In many domains it is easy to see that this hard and fast "yes/no" response to a CSP solution is inadequate. It is quite conceivable that there is a certain degree of uncertainty in the formulation of the CSP itself, to say nothing of possible inaccuracies in the data expressed as domain values. Certainly some of this uncertainty may be addressed in the formulation of constraints themselves, but it seems obvious that some way of specifying the "degree" of match in terms of some predefined "idea" or "desired" match state would be desirable. An ordering or heuristic of this type could serve also as a means of focusing search effort on particularly promising candidates. In addition to uncertainty or error present in either the template specification or in the data, we must also recognize that we are attempting with many CSP problems to model the way in which some agent performs a particular task. The modelling process itself can easily have missed or been unable to capture certain aspects of the reasoning process underlying the solution strategy, and a method for allowing for interaction with the agent allowing for dynamic shaping of the search process would be particularly valuable in terms of both improved search performance and in terms of perceived utility of the problem solving system itself. In many problem instances, the amount of time allowed for searching for a potential solution is limited. In many cases, if a solution cannot be found in that time period, then a "best" partial solution or set of "best" candidates that could possibly be solutions, would be much more valuable than no solution at all.

In viewing constraint satisfaction as search, each progression down in the search tree is as a result of the instantiation of a variable with a particular domain value. Further, if we are performing consistency checking as we go, only consistent assignments will be reached at each level. As a result, any given node in the search space represents a partial solution. A total solution is reached only at a leaf node. In fact, if we were to consider stopping this search before ever reaching a leaf node successfully, many partial solutions were seen, some of which were "closer" to a total solution than others. It is possible to consider keeping track of these partial solutions and then suggesting them in the event of failure to find a total solution. Essentially what we have done in this case is "relax" the remaining constraints, suggesting that the resulting partial solution is "good enough". Ideally though, we would like to perform this "relaxation" in a well conceived and systematic fashion so as to obtain partial solutions that fulfill all of what we deem "crucial" constraints, and perhaps relax some of the other less important ones only.

Essentially, we need to shape the constraint satisfaction process so that it can:

- Provide a degree of modelling of uncertainty by ordering or allowing a certain subset of failed total solutions as "partial solutions".

- Provide an ability for interactive ordering or preferring of partial solutions DURING the search for total solutions.

- Allow for "anytime" behaviour, where a demand can be made of the problem solving system for its' best solution or set of solutions at an arbitrary point in time, BEFORE the search is completed. Ideally, the solutions given would be the "best" possible (in terms of some predetermined defining measure), given the time spent. Further, subsequent time spent would "improve" the quality of the solutions. It seems reasonable that if we are attempting to deal with uncertainty in some measure, that the "best" quality could be "certainty" or "confidence", thus we could achieve greater confidence in a particular solution or set of solutions by spending more time on search.

In subsequent subsections we will outline three different approaches to ordering and preferring potential partial solutions during the larger template matching process. The first, *partitioning*, will be concerned with attempting to break down CSPs into several component parts, solving each independently and then re-composing the solutions to each component into a solution or set of solutions to the total problem. The second, *abstraction*, will outline how we may arrange a problem hierarchically into progressively more constrained version of the problem, and attempt to search through the space of all simple or constrained solutions in a way in which we can achieve both timely solutions and are guaranteed increasing solution "quality" over time. The third subsection will discuss possible *hybrid approaches* for combining partitioning and abstraction. In addition, we will look specifically at how these approaches relate with an interactive problem solving paradigm, and also what kinds of heuristics will be of particular use to a decomposed problem solving approach.

6.1 Partitioning CSP

6.1.1 Simple partitions

Some problems seem to "naturally" decompose into parts that may be solved independently first, with the final solution being composed of the completed independent solutions. A simple example of this type of decomposition would be in solving mathematical expressions. Bracketed sections may be solved first in an expansion tree structure, with the results being accumulated upward from the leaves of the tree to the root, with the root being the ultimate solution. Other problems are less "structured" and yet can also be decomposed readily. For instance, the problem of constructing an airplane might be seen as too complex a task. Breaking the task down into manageable pieces might first suggest that we need to "construct a pair of wings", "construct a body", and only then assemble the wings and the body to form an airplane[1].

Similarly if we were attempting to recognize an airplane in an image, we might first consider trying to recognize portions of the plane first that are in some sense more "elemental", for instance the wings, and the body. Once we had recognized what we believed were wings and bodies in a picture, we could then attempt to put them together according to the known definition of what a plane looks like in terms of the relationships between wings and the body.

Partitioning implies a divide-and-conquer methodology which relies on the fact that the divided problems are each simpler than the original problem and in fact which are simpler to solve as a set than the original problem. Essentially this type of decomposition and recomposition could only be justified as a sensible approach if the amount of effort spent solving the individual subproblems plus the time spent decomposing and recomposing was less that the effort that would be spent in solving the entire problem all at once. In this paper, the focus of experimentation has been to demonstrate the utility of decomposition for one instance of CSP.

6.1.2 Embedded CSPs using partitions

We commonly look at a CSP as a problem in which we have a set of variables each with a domain set, and with a set of constraints on the assignment of domain values to each variable. Consider now that each "variable" in the CSP could represent a sub-problem, and each domain value for that variable a possible solution. Figure 6.1 outlines this "embedded" problem structure.

[1]Airplane analogy due to Qiang Yang

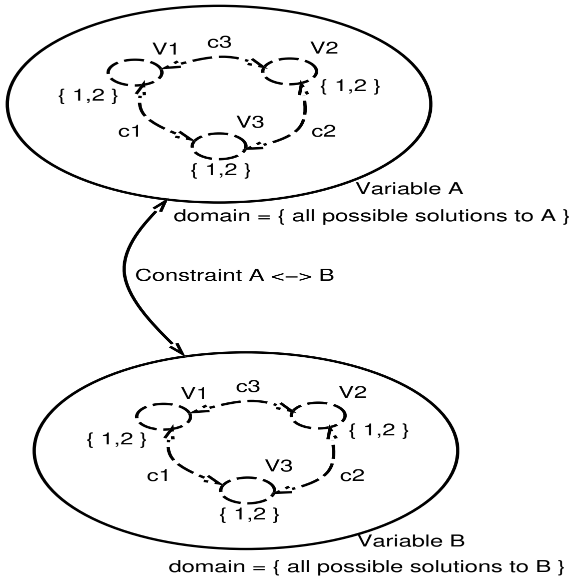


Figure 6.1: Embedded Constraint Satisfaction.

If we consider that the "outside" or "external" CSP is made up of the "rules" which constrain how the individual "internal" CSP solutions may be combined, we see a convenient representation for CSP decomposition.

6.2 Abstraction and CSP

6.2.1 What abstraction means in this context

In attempting to apply techniques of abstraction to a constraint satisfaction problem (CSP), we must first understand what abstraction means in the context of constraint satisfaction. Some basic questions must be addressed at this point, such as "How can abstraction improve problem solving efficiency ?", "How would abstraction be encapsulated into our problem specification ?", and "How must we modify the way we search for solutions in order to take this new specification into account ?".

`"How can abstraction improve problem solving efficiency ?"`

In order to be of any use, one would expect that abstraction would result in an overall simplification of either the problem itself, and in tighter control of the process of solving the problem. It is apparent that simplification of the problem could arise from encoding more domain information into our representation. Given this additional information, we would wish to modify our problem solving process so that we basically remove from consideration effort-consuming dead ends earlier in our search. The intuitive appeal of this use of abstraction lies in the fact that we are attempting to tailor our search for a problem solution according to specific problem attributes. The effectiveness of abstraction as a strategy is not well understood theoretically, and in the context of this work the utility of abstraction is addressed empirically only. The next obvious question in our specification of abstraction is what domain information can we encode, and how ?

`"How would abstraction be encapsulate into our problem specification ?"`

Typically abstraction is applied to a particular problem by creating what is known as abstraction hierarchies. Hierarchies basically attempt to identify for each aspect of a particular problem how "integral", "key", or "important" that aspect is. These structural problem aspects are organized into hierarchies from "most" to "least" "important". In terms of CSPs, problems are structures in terms of N variables, M constraints amongst those N variables, and a domain of some size for each of the N variables. Consequently, the specification of an abstraction hierarchy for a particular CSP would involve the identification of the "important" variables, and constraints. In addition, abstraction is often to taken to mean detailing the difference between "more specific" and "less specific" problem representation. In CSPs, constraints are essentially the guardians of specificity and so a less specific representation of a problem instance could in fact be adjusted to have less constraining constraints. We then have at least 3 ways of formulating the differences between levels of an abstraction hierarchy for, or of relaxing, a particular CSP.

1. Ordering Variables by "importance".

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

2. Ordering Constraints by "importance".

   As for variables, the intent is that a "high" level representation of a particular problem includes only "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 constraints 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 categorized in some other way by level for all variables. Arc constraints affect a set of variables and may be categorized with respect to either that set of variables or to the abstraction level.

3. Relaxing Constraints.

In addition to constraint ordering, constraints may themselves be "relaxed" or have their restrictiveness softened. As for constraint ordering, only constraints between variables which exist at a particular have any meaning in the representation.

In some manner we must formulate the problem representation hierarchy utilizing these tools. Work has been done in other domains in terms of attempting to automatically create these hierarchies [34, 33] but for now we will consider only the manual construction of these hierarchies. In conjunction with a domain expert we must identify the "key" features of the problem, and in some number of layers represent what that expert expects will be successive problems whose solution would suggest a progressively increasing degree of confidence in the accuracy of the solution. For instance, if a solution to the "most abstract" representation of a problem is considered to be "quite likely" true, then the solution of the next "least abstract" representation should result in an overall increase in confidence in that solution.

We have now identified one immediate benefit of applying abstraction to CSP - the ability to make use of 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-guesses" near-solutions. Further, these near-solutions are arrived at systematically in that the user has implicitly indicating what constitutes progressively improving confidence for a particular problem domain.

`"How must we modify the way we search for solutions in order to take this new specification into account ?"`

Essentially the hard work is done in the representation phase. What we now are presented with is a succession of CSPs which may be solved according to the techniques discussed earlier in this paper. The advantage now lies in that since the hierarchy specifies "increasing specificity", any solution to the most specific representation will also be a solution to any of the more abstract representations. Thus, once we have solved an abstract representation completely, we need only attempt to "filter" the abstract solution set between levels according to the changes we detail in terms of constraints or variables at that level change.

In fact, at each level of abstraction in the abstract solution space we essentially have the problem neatly partitioned. In Chapter 7, we describe these partitions in terms of representation changes. Since each abstract solution is a candidate to become a next least-abstract level solution, we may apply the constraints between levels to each candidate independently of the others. Failure of the new constraint set application indicates the failure of the candidate to become accepted as a solution at the next level.

Further, other heuristics essentially independent of the abstraction representation may be applied at each successive abstraction level shift. For instance, a heuristic specific to problems with spatial orientation (Spatial Cohesion) is described on page 97 and can be used as a means to simplify extended abstract solutions before applying the next level constraint set.

6.2.2 Abstraction as partial solution of CSP

We have until this point discussed the concept of "partial solutions" quite loosely, essentially labelling any consistent partial assignment of domain values to variables a partial solution. To some extent this is true, since the portion of the problem associated with those assigned variables has been solved. However, such a "partial solution" may be as useless as no solution at all. What would be valuable is a predefined portion of the whole problem, whose solution indicates something about the entire solution in itself. [16] introduce the concept of Partial Constraint Satisfaction Problems (PCSP) in order to systematically define the relaxation of constraint satisfaction problems. In essence, there are three ways in which a CSP may be "relaxed" or simplified:

1. The domain of a particular variable can be extended to include more domain values.

2. A variable or constraint may be removed from the original CSP.

3. The domain of a constraint may be enlarged.

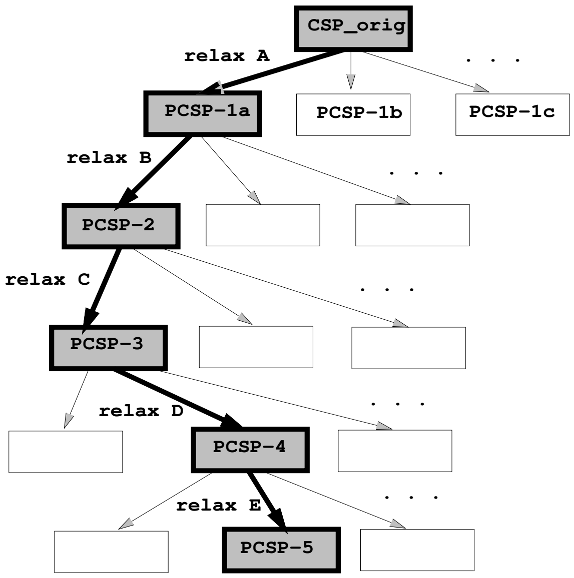


Figure 6.2: One abstraction hierarchy in a PCSP space.

In Figure 6.2 we consider the original CSP as the common root of a graph representing all possible relaxations of that CSP. Each node in the graph are 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. Since there is a very large space of possible applications of the three relaxation steps at any stage of creating a PCSP space, a domain dependent method is assumed to detail appropriate relaxations, and indicating bounds on how much relaxation is acceptable for this particular problem. If we now were to consider the problem as searching through this problem space attempting to find one PCSP that could be solved, a great deal of work must be expended in determining which ones could not be solved. An appropriate method might interleave search through the PCSP space, thus not overcommitting to either too strict or too relaxed versions of the CSP.

In a very real sense in building the 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.

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 previous three 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 6.2 on page 94 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 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.

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 [70], a 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. It is also possible that a user may wish to interleave the various levels of search and interaction.

6.3 Combining partitions and abstraction

We have outlined how CSPs could be decomposed through either partitioning or abstraction approaches. An interesting point for further investigation is the issue of interleaving these concepts in decomposing CSPs[2]. For instance, a complex problem abstracted into a hierarchy of several layers could have further decomposition within each level, in the form of a partition, or even further abstraction inside that level. This flexibility allows a very wide range of definition of a particular CSP, and suggests many possible options for the designer to consider in deciding how to decompose a particular problem.

This type of problem structuring has fairly broad implication to the manner in which search for solutions could be performed. A global strategy controlling where effort is spent across hierarchies and between partitions would be required. In addition, the very nature of partitioning suggests that multiple processor or distributed solutions may offer many more possible extensions to the control strategy.

6.4 Decomposition and user interaction

As we have stated earlier, one advantage of partitioning or abstracting a large CSP problem is that it provides a useful model for solving the CSP in which the user may interactively guide the search. This guidance can occur in several forms for both partitioning and abstraction. We will consider abstraction as an interaction strategy in detail in Chapter 7, and only touch on the important issues in this short summation.

During the search of the PCSP space detailed in the abstraction section, high level solutions are found for abstracted CSPs, and these candidates are then refined if possible to lower level solutions. This process is repeated until a solution set is found at the lowest or base level of abstraction. Justifications for this approach include:

1. Elimination of high level solutions from the set of solutions will simplify the search, and can also account for the ability of the user to understand the domain better than has been captured effectively in any pure search pruning heuristic. Basically, the user is allowed to view and remove candidates before they are passed between levels. This process can be modified to suit the user and the application in terms of the degree of interactiveness required or desired.

[2]Concept and implications arose during a discussion with Dr. Qiang Yang, University of Waterloo

2. Ordering of high level solutions by "likelihood" or some other user based preference heuristic that may not be easily captured in an automatic heuristic. If the user can see and evaluate the possible high level solutions based on his own experience or preference, the solutions that are eventually found at lower levels may suit better the user requirements or needs.

An interesting side-effect of this user interaction is that observation or analysis of the selections or ordering of the user may assist in the creation of more automatic methods of filtering that can be employed in future versions of the search process. These methods may become apparent to the user through his mechanized selection process, or perhaps to other independent analysts observing the filtering process as employed by the user.

Looking at CSP search as several disjoint CSP searches for separate partitions, and then re-composing these partitions into complete solutions allows for a similar level of user interaction in the process. Justifications for this approach include:

1. Elimination of elements from *any* of the partitions will reduce the number of recomposition attempts that need to be attempted without computational effort. Depending on the problem domain, the identification of some or even one part might be enough to dramatically reduce the possible explanations of that deployment. In this way partial solutions may be very valuable.

2. While many real world problems may often be solved using a form of reasoning similar to abstraction, others may have no easily identifiable form of abstraction, while they have several parts that fit together on a single level of conception. If we consider unit deployment at a particular level of dtail, such as company, a template might have several equally important branches or sections of deployment such as medical and artillery.

Just as abstraction suggested possible derivation of analyst expertise at reducing the possibility set, so too can we observe the combinations that are frequently filtered out in a search for a particular template, and the weight assigned to each in the determination of the existence of some template instance.

6.5 Spatial Cohesion Heuristic and Problem Partitioning

Spatial Cohesion (SCH)is basically a heuristic applicable to CSP problems that are grounded in spatial coordinates and which are partitionable either by abstraction or some other means. Essentially, we are attempting to encode rough boundary information for an entire solution based upon only partial variable assignment information. Of particular importance is that the boundary information suggested by SCH represent the extreme outer boundary and not preclude any solutions prematurely.

SCH seems especially applicable to abstract representations since the partial variable assignment information has been defined in the representation as being "important" or "key" and therefore we assume that this information is more relevant to determining boundaries. In problems partitioned without reference to abstraction but rather to portions at a single abstract level, the definition of the SCH area may be more difficult or impossible depending on the particular problem, however, it is easy to imagine a problem where the size or scope of one key part limits the possible scope or aspect of another part. Any problem in which symmetry or some kind of balance exists would appear to be candidates for this type of partitioning heuristic.

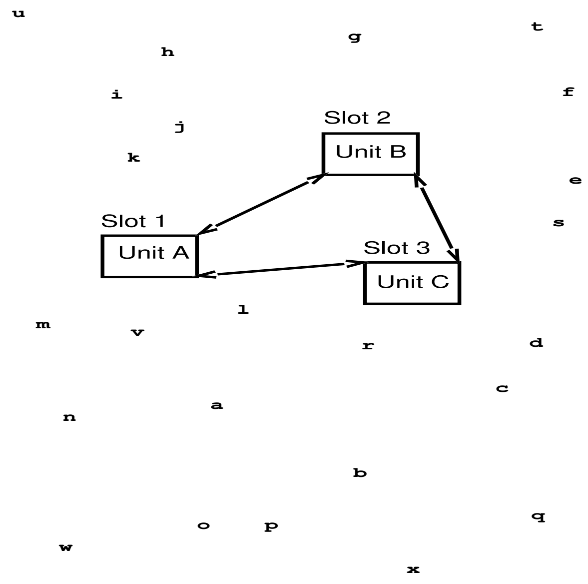


Figure 6.3: SCH Example with level 1 solution.

The advantage of determining boundaries is straightforward in that it allows for removal possible domain values which lie outside of the boundary in a particular problem instance. For instance, in Figure 6.3 and 6.4, SCH is sketched for a set of abstract problem solutions. Once assignments have been made to the variables so that the joined pairs of nodes represent abstract solutions, a boundary is created limiting the scope of that 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 abstract 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 6.3 shows the abstract solution where Slos 1, 2 and 3 are matched to Units A, B and C respectively. Before search continues at the next abstract level, we apply a spatial cohesion constraint derived from A, B and C instance positions which limits the area of search for the next level Slots as shown in the limiting circle. Thus, the only candidate instances in the selected area for the next level slots are *j, k, v, l, r, d s and e* .

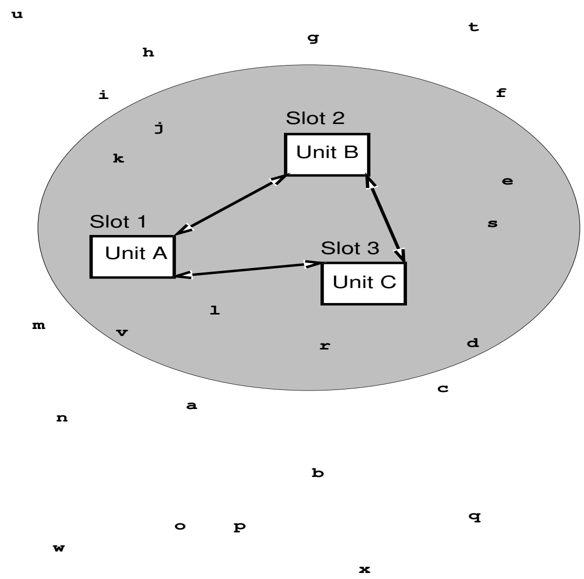


Figure 6.4: SCH Example limiting range of level 2 instances.

Chapter 7

Abstraction and Interaction

7.1 Introduction

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 [70, 73]. In this chapter 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. This chapter is self-constained to some extent, and effort is made to clarify briefly some of the concepts introduced earlier in this report before addressing the abstract model.

7.2 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 [68] 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 chapter 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 chapter 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.

7.3 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 chapter 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 [35], and are known as a Constraint Satisfaction Problems (CSPs). An example of a template constraint graph is shown in Figure 7.3 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) 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 [49] 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 [49] 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, 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 chapter is designed so as to provide a framework in which further research in appropriate combinations of heuristic search and propagation can be undertaken.

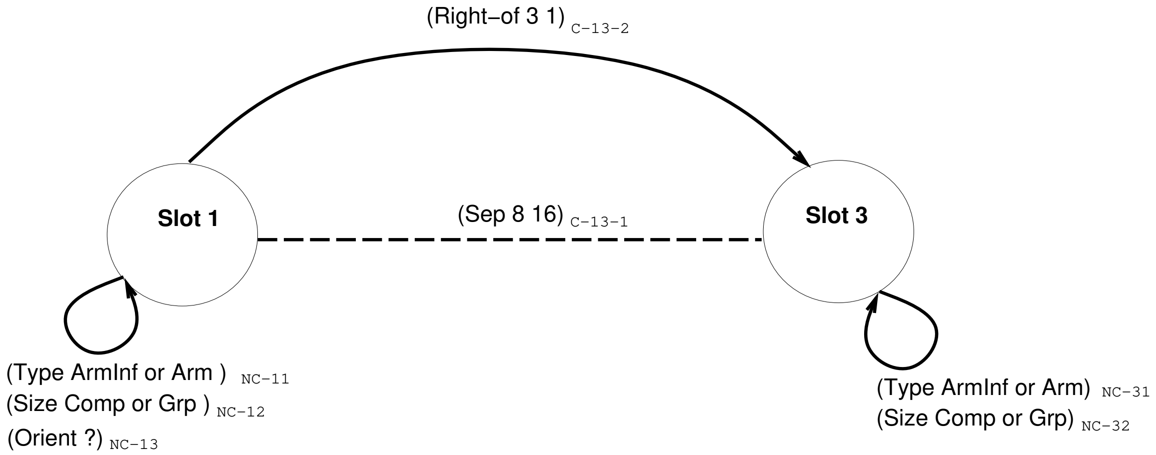


Figure 7.1: Simple PCSP 2 variables, 2 arc constraints.

Figure 7.4 details a situation with one embedded Figure 7.3 template instance amid 300 randomly created and positioned elements. In this situation, there is also one embedded instance of the template of Figure 7.2, and one embedded instance of the template of Figure 7.1. The time required by a simple backtracking configuration of non-interactive MOTORS 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 7.5 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

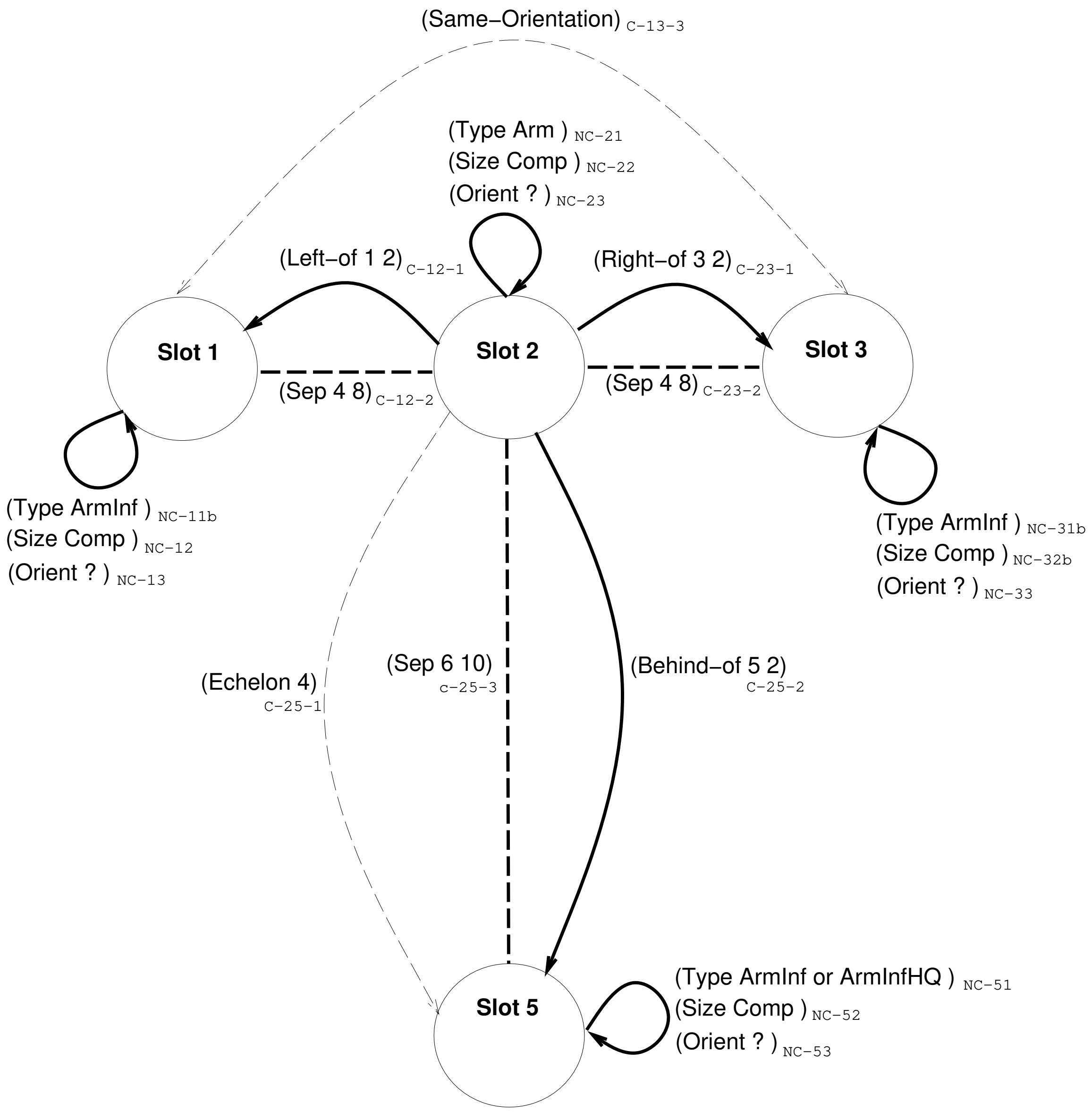


Figure 7.2: Medium PCSP 4 variables, 8 arc constraints.

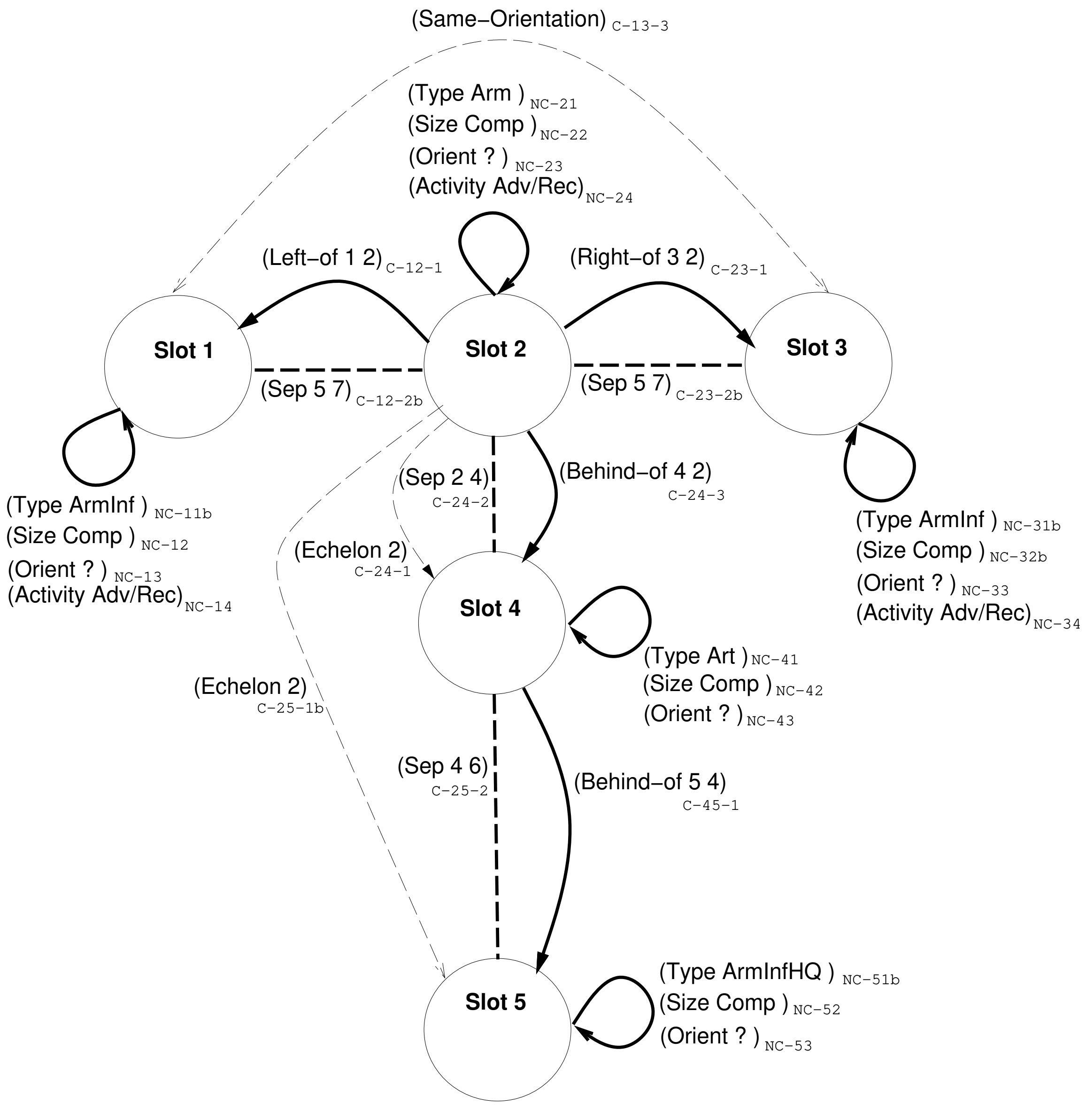


Figure 7.3: Whole CSP 5 variables, 11 arc constraints.

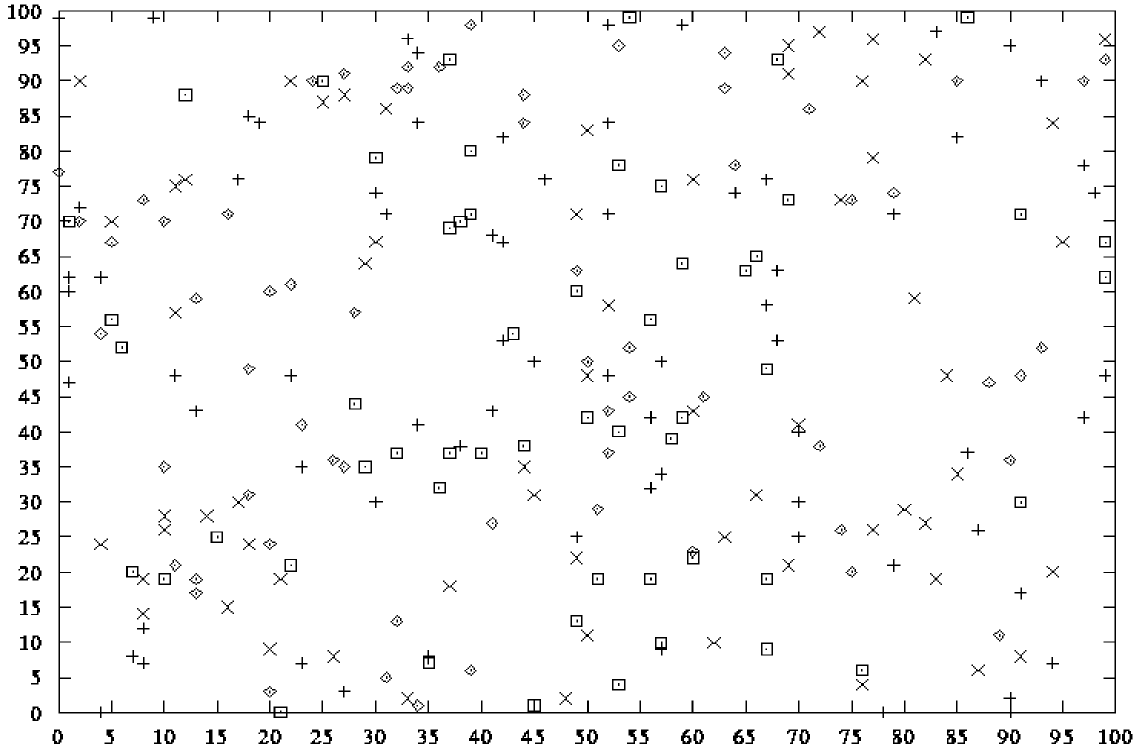


Figure 7.4: Situation with 300 unit sightings and 1 whole template instance.

during search, and domain-dependent spatial heuristics are discussed further elsewhere in this paper.

7.4 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

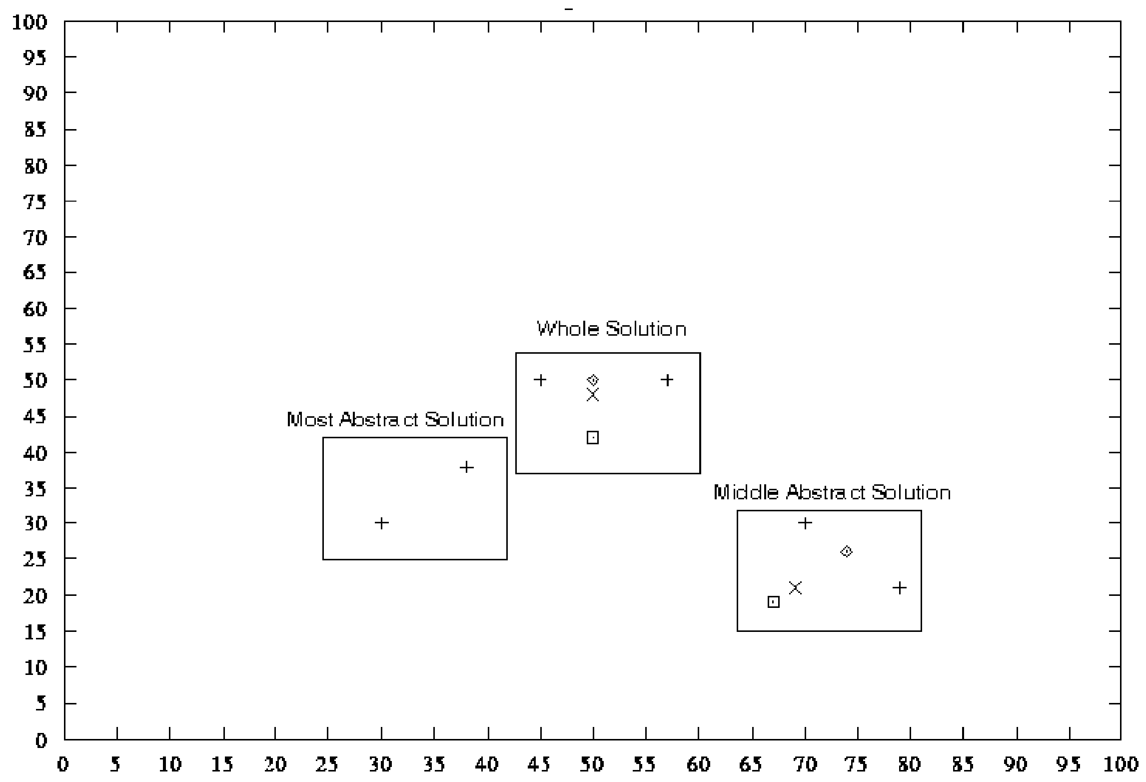


Figure 7.5: Template instances hidden in Situation.

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 abstracted 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. Earlier, we discussed partition of CSPs further, however, this chapter will concentrate on developing a model for the solution of CSPs based on solving abstract problem representations before more specific or detailed ones.

In [16], 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

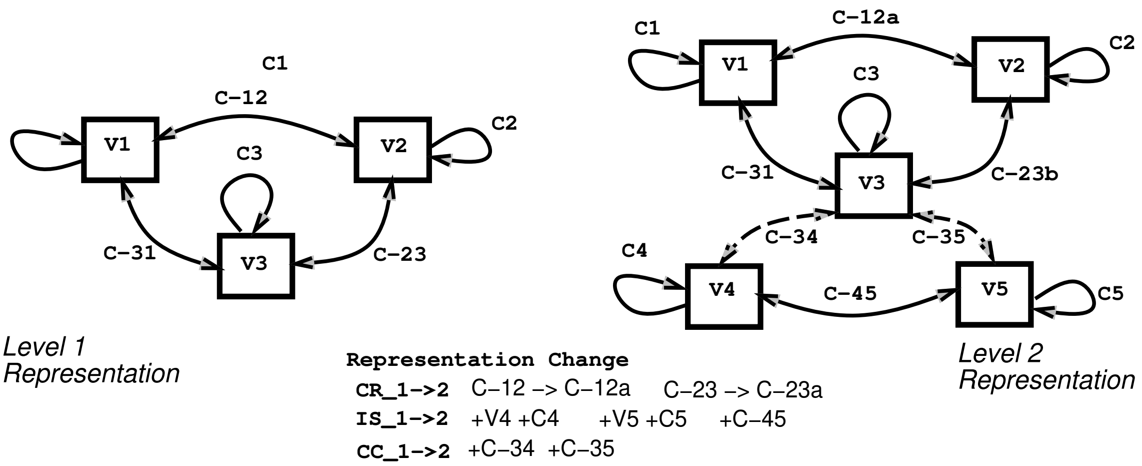


Figure 7.6: Representation Change between abstraction levels.

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 6.2, Chapter 6, on page 94.

The specification of an abstraction hierarchy for a particular CSP involves the identification of the most "important" variables and constraints. Abstraction is 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 important" 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[1], 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.

[1]One example of military critical element identification that supports this model is *Signature Equipment* [53, 11], defined as "any item of equipment which reveals the type and nature of the unit or formation to which it belongs".

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 by the analyst are the same as those for formulating the differences between levels of an abstraction hierarchy, as described in Chapter 6, page 92.

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 [34], 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.[2] 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 [71].

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 [71], from original CSP to least specific PCSP on the leaf. Figure 6.2 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

[2] In a related work, Nadel [48] 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.

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 [73] 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 7.3 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 shown in Figure 7.1 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, the result shown in Figure 7.2. Finally, the most detailed model of Figure 7.3 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.

7.5 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 [71, 45] 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 [70] 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.

7.6 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.

7.6.1 Algorithm

Figure 7.7 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 7.6) 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 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 earlier in this paper 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 solutions 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 [68].

7.7 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.

7.8 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 7.7, 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 section, Table 7.1 shows the CPU search time required to locate all occurrences of the template described in Figure 7.3, and an abstracted version of 4 Slots described in Section 3 and detailed in Figure 7.2, in varying amounts of sighting noise. Each entry in Table 7.1 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

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 7.1: Maximum search time, two template models.

better characterize actual situational information to determine distribution and frequency of various sightings more accurately.

7.9 Conclusion

In this chapter 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.

```
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 7.7: Abstracting Interactive Search Algorithm.

Chapter 8

Model Template Recognition (MTR) as CSP

In Section 2 we outlined the problem of recognizing templates in the military domain. In Chapter 7, Section 7.3 we outlined template recognition represented as a constraint satisfaction problem. Here, we discuss the implementation, selection and control of an appropriate search strategy or strategies, and outline the application of both domain independent and domain dependent heuristics towards achieving the best possible performance in this domain.

8.1 Decomposing Template Recognition

Why should we want to decompose the Template Recognition problem ? Several reasons can be seen. First, the problem of creating an adequate template model is a major point of concern since it completely determines which, if any, patterns will be detected. A particular worry is that a template will be defined too strictly in some sense so that a template is missed in the world situation. While this is no fault of the recognition procedure itself, it will quite often be the case that templates are defined based upon "best guess" approaches and do not represent absolutes in the mathematical sense. Other approaches to template recognition proposed have been based upon best-fit schemes where constraints are essentially relaxed until a certain number (possibly 1) of solutions are found. A second justification for decomposition is that it is conceivable that some partial solution right away would be better than some complete solution later. In fact, since the analyst can only process alternatives serially it is quite conceivable that some type of iterative improvement method would be of benefit. Further, an iterative refinement approach would allow for a readily seen level of analyst participation in the refinement process where he/she could prioritize, accept or reject partial solutions as they become available, thus directing later refinement to successively more complete solutions.

There are certainly several conceivable approaches to modifying the way in which MOTOR works that can accomplish these type of goals. These are discussed separately in greater detail below.

8.1.1 Partitioning

One possible decomposition method already alluded to would be to partition the original CSP (template) into separate sub-templates. For example, if we were attempting to recognize an object such as an airplane in a picture, one approach might be to subdivide an airplane into it's constituent major parts, say a wing object, and a body object. We thus define two templates, wing and object along with the set of constraints that mate the wing and object to form a plane. MOTOR would proceed serially (or possibly in parallel) to solve each of the object recognition CSPs separately and then merge the solutions according to the re-application of connecting constraints, and possibly according to analyst preference interactively.

Clearly this process can take on a multi-level feel where templates can be decomposed successively into smaller and simpler component pieces, and templates are constructed from the base parts of the previous level.

8.1.2 Abstraction

Abstraction as discussed earlier in this paper has an intuitive appeal for the problem of model template recognition. It would seem sensible that the determination between templates is triggered in the analyst's mind by "key" differences in the unit patterns between templates. These patterns can signify indicators to the analyst for how to restrict his set of possible determinations. If we can extract these types of patterns with the help of the analysts themselves, the patterns could be used as determinants for the recognition of templates. In particular, if an analyst could say "this pattern or group of equipment is *necessary* for the presence of a certain template, we could quickly eliminate cases where this was not the case. Specifically, we could only examine those situation cases where these key conditions are satisfied. Unfortunately, it is quite possible that a template containing "key" equipment or patterns is present, despite the fact that the key equipment has not been sighted. In these cases, where a partial template could be found in general, but where key equipment is missing, the problem is more general, and we can either fail to include such partial template occurrences from consideration or allow for more searching in these cases. 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, and the degree of difficulty of the process.

We must realize that there are several very different but related problems in template recognition. The primary problem is perhaps the determination amongst a set of templates for the best fit in a situation. Of course, more than one template may fit, possibly in different portions of the situation. A subproblem of this is simply to determine whether a certain template fits in a situation. In this case, a given template may have more than one instance in the situation. While the first problem is the larger and widest in scope, the subproblem of identifying a single template's occurrences in a given situation is the one which we are currently concerned with in this work. In light of this focus, we will be more interested in constructing template hierarchies which can indicate the presence of a single template than in creating some kind of "discrimination" system or hierarchy which would help us determine which template is represented in a particular data pattern. To a great extent this clarifies the distinction between this work and some other pattern recognition applications in computer vision related publications.

Constraint relaxation is a technique commonly used to find a solution to some CSP where all constraints cannot be adequately satisfied, but some solution is required nonetheless, and to this we have arrived at a definition of abstraction that allows us partial solutions. We will wish to categorize partial solutions strictly in terms of our particular domain keeping in mind our primary goal of determining all instances of a particular template in a data set. If we can identify "crucial" or "critical" portions of the template that in some sense are *required* by any partial solution, we can use this to define a strict set of relaxation operators that will simplify our original problem giving a simpler problem, and a solution to our simplified problem will constitute a partial solution.

In Section 6.2, we outlined how abstraction can be used to formulate a hierarchy for CSPs. Earlier work in Partial CSPs (PCSPs) focused on all possible ways in which a problem could be relaxed, resulting in a space of relaxed problems. Searching through this space to a leaf (least abstract) solution node resulted in a problem solution. We are interested in limiting this PCSP space dramatically by formulating the hierarchy according to the way template recognition is "typically" performed. We are in fact defining a particular hierarchical path through this larger space, and only the series of relaxations on this path with be considered in the matching process. The conception of how abstractions are related to partial solutions is discussed in detail in Chapter 7.

There are two main aspects to this work. First we must define the abstraction hierarchy that will determine the path through the entire PCSP space. Second, we must indicate how this search will progress.

8.1.2.1 Formulation of abstraction hierarchies

The process of formulating a particular template's abstraction hierarchy must be done in consultation with intelligence officers most familiar with the process of identifying these templates in intelligence message data. The example given in this paper is meant as an example of how these template hierarchies could be formed. In general, the main issue should be to identify what would constitute "critical" indicators of the existence of a particular template, and then supplement this with supporting evidence. Essentially we can view the progression between abstraction levels as progression between some degree of likelihood of a matching template instance. In fact, as we descend our abstraction hierarchy from most-relaxed problem to least-relaxed, we will be moving between partial solutions that have progressively more support in the data.

As defined in Section 6.2, there are three acceptable mappings or elements for performing relaxation that will constitute the change between abstract levels. These tools will allow us to define our particular abstraction or partial solution path for each template, and they are:

1. Extend a variable's domain.

2. Remove a variable or constraint.

3. Enlarge a constraint's domain.

Some set of these three modification strategies will formulate the "difference" between abstract levels. In Figures 7.3, 7.2, and 7.1 of Chapter 7, we outline an abstraction hierarchy with three levels. The first level (labelled "plausible"), indicates the initial or "key" indicators of the presence of this template. The second level (labelled "supported"), indicates that a match of this level would indicate that the earlier plausible match now has important additional supporting evidence. The third and final level in this hierarchy (labelled "base"), indicates that a match at this level is a match of the entire template.

In general, any number of abstraction hierarchy levels may be used. One key issue in determining the complexity of a hierarchy is that the search strategy we will be using must cross all levels, and also that the analyst will have to interact in the actual search at some point. This interaction should be configurable by the analyst, and will basically allow the analyst to act as a "pruning" heuristic at some step across abstraction levels. For instance, the analyst may wish to "approve" any possible solution at the "supported" level before additional work is done attempting to fit that solution at the "base" level. A different strategy might insist that the analyst only approve or remove solutions at the "plausible" stage, while yet another would require no analyst interaction at all.

In addition to "pruning" solutions at a particular level, certain strategies might indicate that all solutions at some level will be found before solutions at subsequent levels are considered. In cases such as these, the order in which abstract level solutions are considered may be of importance to the analyst. For instance, the analyst may be searching for only a single solution that fits adequately and is not interested in all solutions, and in this instance intervention in determining the order in which abstract solutions will be examined further can dramatically affect the search time required to find the "single" solution the analyst is searching for. Conversely, if the analyst is interested in ALL solutions, then ordering will only affect the ordering in which the solutions are determined.

8.1.2.2 Abstract search strategies

In [70], search in abstraction for planning is discussed in some detail. A complete search strategy is outlined (A*), and the use of this strategy in searching across levels of an abstraction hierarchy are examined. The key is that we are interested in searching for solutions to each level of abstraction in an interleaved fashion, Basically, A* can be adjusted with a preference heuristic to prefer solutions at a single level and act as a breadth-first search which finds all abstract solutions before attempting less abstract solutions, or as a depth-first search which attempts to find an abstract to least-abstract series of solutions, or as some combination of the two strategies. One composite strategy described is Left-Wedge which basically extends the search space rightwards and downwards at the same time and thus explores the space in a wedge-like manner. Work is expended extending the number of abstract solutions, and also in attempting to find less abstract solutions.

In any of these cases, the space examined is identical, and the key issue is the order of generation of solutions at each level. Since we are quite likely bounded in the amount of time allowed for the search, we wish that any chosen strategy will exhibit "anytime" properties, where a solution found in a finite amount of time is essentially the best possible solution available in that amount of time. The definition of "anytime" and which properties we are most interested in will determine the relative appropriateness. Essentially if we could somehow evaluate how "good" a particular solution at a particular abstract level is, we could structure our strategy accordingly. However, this "goodness" can be only vaguely defined - for instance an analyst may recognize an abstract level solution as being more reasonable than a more specific solution based on some measure or indicator of which the algorithm is unaware. The exact search strategy most appropriate to this type of problem and interactive application is a matter for debate.

8.2 Accommodating change over time

This area is one in which we shall be concentrating on during future work, however the following are several ideas for possible application.

8.2.1 Tracking solutions

Certainly it is true that in the more general sense the domain we are interested in applying MOTOR to is in fact quite dynamic. Positions and descriptions are continuously changing over time thus essentially changing the initial CSP problem description by changing the domain values for each template variable frequently. Since it would appear to be impractical to attempt to solve a CSP for each Template each time a change occurs in a domain value, it would be beneficial if we were able to either "track" changes in some way so as to filter out previous solutions as they become no longer valid, and in fact to recognize new solutions that have become valid through the change. The conception of selecting some set of solutions to "track" over time needs to be investigated further.

In addition to tracking changing sighted units, some consideration could be given to modelling movement over time based upon a knowledge base merging domain independent knowledge about unit movement ability and terrain, so as to "allow" for the decay of existing knowledge. For instance, information such as unit position is quite time dependent, and since the various sightings are made at various times, a static reading or interpretation of the combination of these sightings must be questioned. If for instance, an exact location sighting were "spread" over time (to certain limits) based upon terrain, unit mobility, and possibly other domain information, the possibility of a certain template match could be quite different than in the static time instance. Of course this type of modelling can make the problem virtually unsolvable when taken to extremes (since domain attributes could become overly loose), some degree of decay could provide useful flexibility. It should be noted that this conception is related to other parallel applications of unit mobility being discussed elsewhere [66].

8.2.2 Standardizing for a common time view

As stated in the previous section, sighting information is disjoint in time. If we are attempting to match temporally static spatial templates to this time-disjoint information there are going to be problems. One saving aspect is that the intelligence analyst is often attempting to construct a "current" best view of the situation taking time aspects into account manually. In addition, future analyst tools which make modelling unit movement easier will result in presumably more complete single-time models. In a simpler case, if we could assume that all information present in a situation was representative of actual sightings at actual past times, it would be conceivable that we would wish to "simulate" this information forward to a consistent time point before attempting any template matching. In this case, just as the analyst may use tools for modelling unit movement over time, our "simulator" could use these modules to make simple assessments of unit position at a future time.

This aspect of time-standardizing sighting information would present an interesting case for further study and experimentation, particularly in that the utility of this approach could be easily compared with matching performed using the time-disjoint information only. A particular issue in this type of enhancement would be the performance of the unit movement modelling, and for further information on this issue, refer to [66].

8.2.3 Modelling movement and prediction

Given information about some enemy element such as activity, direction, and mobility capability it is possible to make use of terrain or map information to "predict" a position in the future. Clearly the position would be based on information that is essentially decaying over time, and as a result the prediction would lower in accuracy over time. Essentially we can envisage a model where a sighting possess some kind of inertia when we predict its future location, and at some point this inertia (of movement or of other attribute "accuracy" is expended. The 'inertia' of a sighting could possibly be sustained further if more information is available about the activity of higher formations to which the lower level element belongs.

Time standardizing sightings in order to match templates based on some predictive locations must utilize some similar model of decay of information, and the intent of this discussion is simply to point out that a matched template essentially suggests activity of a higher formation and as such could serve as part of a lower element position predictive tool.

8.2.4 Merging time variant solutions

Alternatively, if we were to attempt to solve for a particular template at some t1 and then at some later time t2, then if we let $S1_t1$ represent a set of solutions to a problem we will call CSP-Template1(world at time t1), and $S1_t2$ represent the (possibly different) set of solutions of CSP-Template1(world at time t2), then both the intersection set $I_{1.2}(S1_t1, S1_t2)$, and the difference set $D_{1.2}(S1_t1, S1_t2)$ will be of interest. Specifically, $I_{1.2}$ could represent the set of solutions at time t1 which have persisted to time t2, and possibly are now more "likely" solutions than those that did not persist from t1 to t2. Similarly, $D_{1.2}$ could represent both those new solutions that have appeared at t2 that were absent at t1, possibly indicating the detection of new information successfully completing some expected template constraint, and also those solutions which have disappeared at t2 that were present at t1, indicating perhaps that these solutions are now less "likely" in some sense than those that are still present. In addition, a more detailed investigation of "why" certain new (t2) solutions have appeared, or "why" certain old (t1) solutions have disappeared can focus attention on critical unit attributes that may be of special interest for the analyst.

8.3 Partial fitting of templates to a situation

Whether one uses a CSP approach based upon an abstract representation of a template, or simply a single-level description, we must have a sensible way of dealing with "near" or "partial" solutions. As described at length earlier, abstraction provides a framework in which to categorize these partial solutions. If an abstract version V_i of the problem has a set of solutions S_i, and the next less abstract version $V_i - 1$ obtained through only the addition of some variables and corresponding constraints has an empty solution set $S_i - 1$, then we can easily identify that the slots added presented the difficulty, and in fact some of these will indicate "holes" where although no domain value fit a variable and its constraints, we might believe that the domain values given are in fact in error, and such a value might be supposed to exist.

Essentially, every failed solution path constitutes a "partial" solution, where some variable was found which had no domain values that satisfied some constraint or constraint set. We have discussed elsewhere the methodologies for relaxing these constraints in order to find a partial solution, and here we will mention another possibility - that some other currently unknown domain value exists which *would* satisfy the constraints. In terms of template fitting to known situation data, we are stating that in certain situations, a template which failed to satisfy *all* of its stated constraints or variable assignments might still be accepted as representing the situation it partially matched, and the resulting constraints be used to "narrow" or "limit" the range and definition of the missing elements that would have provided a total fit.

Essentially many types of partial solutions could exist, however, we will assume that the definition of what constitutes a partial solution is predefined. When using abstraction, an abstract solution can be seen as a partial solution, when using a single level, a definition indicating which constraints or variables may be relaxed or removed could serve this same purpose. In the next subsection we will look more closely on partial fits and how we can make use of them.

8.3.1 Defining "partial fit" using abstraction

We define more precisely what a partial fit really is in terms of partitioning or abstracting and solving less constrained versions of a CSP in Chapter 7. We must pay attention to partial fits involving simplified problems by removing constraints as this actually describes the slots or holes that we will wish to "fill". The constraints for the location of these slots are important also. In addition, we could relax constraints on the location of other slots and hypothesize that some particular placed unit should be moved slightly in order to fit a template. Essentially this would amount to correcting known sightings according to an accepted template. This is one area for future thought and consideration, particularly as to how it would fit in with the analyst's expectations and existing hypotheses.

One interesting issue is how to prefer one partial fit over another. In trying to solve a particular CSP P, we know that P can be relaxed in many ways, forming a set of relaxed P_i. In traditional CSP problem solving, each constraint c can be tested between variables assigned a domain value, and either *succeeds*, or *fails*. A relaxed constraint has a wider range for success, but in standard formulation either succeeds or fails. Since a partial fit by definition has "failed" in that all constraints are not satisfied, we must somehow be more flexible in determining preferability.

One approach would be to assign some degree of fit measurement to the evaluation of each constraint. Combining the individual constraint fit values according to some method which reflects the critical aspects of a problem would allow us then to evaluate and compare several partial fits. A heuristic based upon such a scoring method would have difficulties certainly, as it is conceivable some more critical constraints with little "fit" must not be allowed to be outweighed by the satisfaction of many "unimportant" ones.

8.3.2 Expectations in the "holes"

Given a solution set for some abstract template instance, a set of partial solutions is available to the analyst. Depending on time constraints (among others), the analyst may wish to forgo attempting to find more specific solutions by extending these abstract solutions, and may wish to adopt one of these partial solutions directly. Similarly, if a solution set exists at the abstract level, and no solutions were found at the more specific level, or if the solutions at the more specific level were unacceptable to the analyst, the analyst may wish to adopt an partial solution.

The partial solution is a solution to a simplified or partially specified version of a complete template. Essentially the simplified version would constitute a sub-part of the entire template. The remaining difference between the simplified version and the complete template forms the "hypothetical" or "possible" information that would be required in order that the entire template would "fit" the situation. The analyst may be interested in creating "hypotheses" about the existence of currently unsighted enemy units according to the slots in the "difference" between templates. If the slot were combined with local terrain information, a best-guess could be made as to the location of the hypothesis.

In earlier work [68], we have described a concept known as "expectations", which allow an analyst to insert suppositions about enemy existence or movements and which alert the analyst to the arrival of supporting information, conflicting information, or the expiry of a particular supposition in face of lack of evidence in a particular time interval.

Templates are defined in this paper as having enemy unit slots constrained both spatially and in terms of their specific identifying characteristics such as type, size or activity. Consequently, the insertion into a hypothetical situation of an expectation of an element corresponding to a currently unfilled template slot can be assumed to possess these "typical" or "default" attributes also. To some extent then, the acceptance of a template by an analyst could conceivably insert expected elements into a situation giving them identity, activity, and quite possibly other aspects such as direction and speed.

One interesting aspect of using holes as locations for future or current expectations or hypotheses is that the area in which we are posititiong the expectation may have implied certainty that is hard to justify. For instance, if based on the location of several other units we are expecting to see some unit between 8 and 10 km to the right of some object A within 45 degrees off parallel of A's orientation, then a valid question might be - where in the area defined by this explanation might the object be located. Further, since we are ultimately interested in fusing sighting information, where can another sighting be to be recognizes as "possibly" fulfilling this expectation ? The point is that we must be careful in attributing hard and fast definitions to the area indicated for locating the expectation, and realize that the flexible nature of the system we are examining and the approximate nature of the template definitions themselves suggests that this area itself is only an approximation. Other work[1] has attempted to define these areas in possibilistic terms where instead of an area of expectations, we might consider an area with height representing the possibility of the position of the expectation in reality. For instance, the highest point might represent the centre of the area, and the lowest point the edges, or some rounded version of the edges. The height over the remainder of the area represens some decreasing possibility of location down to zero. In this way, the location is represented with continously decreasing possibility. Of course, in such a representation, it may be difficult or impossible to explain the possibility

[1]Suggested by Stephane Lapointe in discussion [36]

at a certain point in convincing terms. A flat model, however, may have difficulty explaining the hard and fast edge to an expectation region in a similar fashion. Any model adapated must have the property of "reasonableness" in the eyes of the intelligence analyst.

8.3.3 Planning intelligence collection using "holes"

While it is true that an accepted template model fit over a situation can suggest the presence and location of unsighted enemy elements, the confidence in this predictions is based on the accuracy of many factors, including at least the accuracy of the sighting information, the speed at which the situation itself is changing, the manner in which the analyst controlled the search for a template fit, and the degree to which a template actually fits the situation.

Even if an analyst is hesitant to hypothesize that an element in fact may exist in one of the template suggested "holes", the existence and predicted are of these holes can serve as a guideline for possible intelligence gathering missions which could actively search for evidence for or against this existence. Essentially, we have previously suggested inserting "Expectations" into these holes, where these Expectations passively scan the arriving message information for confirming or denying evidence. A simple extension to this model would allow mission planning commanders access to the analyst's expectations and reasoning process, thus allowing them to narrow the focus of gathering missions profitably according to their own needs.

8.3.4 Terrain modelling and expectations

While we can obtain rough spatial regions corresponding to the constrained region implied from filled slot locations, this area is "dumb" in that it is not related with the terrain it supposedly sits on. An improvement would be to submit this region as a polygon or something to a terrain analysis module which could adjust this region according to terrain such as water, elevation, traversability, or roads[2] Future work in integrating the various type of analyst tools such as MOTORS and terrain analyzers can improve the range of these suggestions.

[2]Other work is being conducted on terrain analysis and mobility models at DREV, and eventually MOTORS and the terrain analysis module will co-exist and be accessible to the analyst and one another.

Chapter 9

Experimental Results

9.1 Introduction

The results detailed in this brief chapter are meant as only an indication of the approximate time scale required with rather simple application of the MOTORS toolset. The experiments shown are based on the attempt to locate all instances of the template specified in Chapter 7, Figure 7.3. The lisp specification as explained in Chapter 11 for this template is shown below.

```
    '( "test1-w"
 (
  (ts1 ARI  (Adv Rec) * * Co)
  (ts2 ARM  (Adv Rec) * * Co)
  (ts3 ARI  (Adv Rec) * * Co)
  (ts4 ART  *         * * Co)
  (ts5 AIHQ *         * * Co)
  )
 (
  (sep (ts1 ts2) (5 7))
  (left-of   (ts1 ts2))
  (sep (ts2 ts3) (5 7))
  (right-of  (ts3 ts2))
  (sep (ts2 ts4) (2 4))
  (behind-of  (ts4 ts2))
  (sep (ts4 ts5) (4 6))
  (behind-of (ts5 ts4))
    (echelon   (ts2 ts4) (2))
  (echelon   (ts2 ts5) (2))
  (same-attr-orient (ts1 ts3))
  )
 () )
```

The initial situation which includes a single instance of this template is shown next in the format explained in Chapter 11. The matching template instance is sit1-1, sit1-2, sit1-3, sit1-4, and sit1-5. This situation is shown in Figure 7.5 on page 107 in an uncluttered environment, and in Figure 7.4 on page 106 in an environment of 300 noise elements. Various numbers of situation elements are introduced to this initial situation as "noise" in the examples described in this chapter. For the experiments detailed here the distribution of the attribute instance values and the location of the situation elements are random. The entire situation size under consideration is 100 by 100, and all elements are confined to this region.

```
    '( "test-1"
  ( sit1-1  (45 50)  ARI   Adv n co)
  ( sit1-2  (50 50)  ARM   Adv n co)
  ( sit1-3  (57 50)  ARI   Adv n co)
  ( sit1-4  (50 48)  ART   Adv n co)
  ( sit1-5  (50 42)  AIHQ  Adv n co)

  ( sit2-1  (30 30)  ARI   Adv nw co)
  ( sit2-2  (38 38)  ARI   Adv nw co)

  ( sit3-1  (70 30)  ARI   Adv ne co)
  ( sit3-2  (74 26)  ARM   Adv ne co)
  ( sit3-3  (79 21)  ARI   Adv ne co)
  ( sit3-4  (69 21)  ART   Adv ne co)
  ( sit3-5  (67 19)  AIHQ  Adv ne co)
  )
```

9.2 Considerations

The particular search space is determined by the order of the variable instantiation. Some search strategies re-order (dynamic-rearrangement) the instantiation order of the variables in different sub trees on the basis of some heuristic, such as the length of the variable domains (shortest first). Other strategies which fail to specify this order and make a determination non-deterministically can experience quite different results depending on selection. One option we have given is the option to specify this ordering randomly, so as to disallow any "coincidental" domain-definition bias which may affect the results.

In addition, the order in which constraint are applied when checking variable consistency can also affect performance dramatically. If a constraint set between two variable instances is going to fail, it is best to chose a failing constraint first so as to avoid the solution of any succeeding constraints, which are irrelevant if we are only considering a success as the logical AND of all constraints between two variables. In our implementation, a function "consistent-all" is called from any application of consistency checking, which evaluate each constraint in turn until a failure or until all succeed. This function simply evaluates the constraints in the order they are located in the domain definition, unless the "sort-const" flag is set. In this case, "sort-const" may use some specific domain heuristic, or order the domain constraints randomly to disallow any coincidental bias.

9.3 Experiments

All experiments have been repeated with the same set of 10 random distributions of attributes and locations. The results charted are the *worst case*, *best case*, and *median* (where median is the average of the 5th and 6th values) discovered in each of the 10 cases. Note that in Section 9.3.3 full results are shown. In all graphs cases that exceeded 600 CPU seconds are charted for that result, or constraints evaluated at that point. These cases are marked in the full results. Please note that the presence of a negative time value in the summary figures indicates the time limit was exceeded for that example.

9.3.1 No advance constraint propagation

Figures 9.1, 9.2, 9.3, and 9.4, and show the results of identifying all (1) template instance in varying amounts of noise: from 100 to 1000 elements. Search strategies represented are Backtracking, Backtracking with Forward Checking and Dynamic Rearrangement, BackJumping, and BackMarking. Node consistency (NC) checking is done in advance in all test cases, and time charted is total CPU time including node consistency propagation time. Where constraints are charted, this includes all constraints evaluated *after* node constraint evaluation. In the case of these search only strategies, the constraints reflect the amount of constraint checking during search.

Figures 9.5, 9.6, 9.7, 9.8, and 9.10 detail the same problems, but refer to the CPU seconds required in all in place of constraint checks.

9.3.2 AC-3 advance constraint propagation

Figures in Section 9.3.1 show the results of identifying all (1) template instance in varying amounts of noise: from 100 to 1000 elements. Search strategies represented are Backtracking, Backtracking with Forward Checking and Dynamic Rearrangement, BackJumping, and BackMarking. Node consistency (NC) checking is done in advance in all test cases, and time charted is total CPU time including node consistency propagation and arc consistency propagation. Prior to commencement of search, the CSP was simplified using the AC-3 arc-consistency algorithm. In the case of hybrid strategies the constraints measured reflect the constraint checking done both during search and during execution of the propagation (AC3) algorithm.

We note that the result of applying NC and then AC-3 to these problems resulted in the location of all solutions via later search trivially. The propagation located the solutions

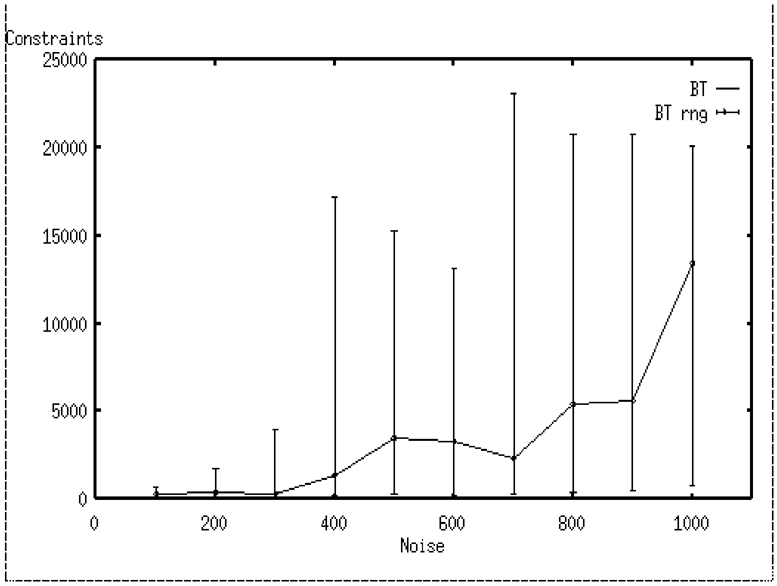


Figure 9.1: BackTrack constraint checks.

independent of search, and for this reason, Figures 9.9 and 9.10 summarize the amount of AC work done combined with only a minimal amount of search. The search work is essentially nil regardless of strategy in this case. Essentially we must devise more difficult circumstances, or less constrained templates for experimentation, and this is a part of ongoing future work in interaction, and actual characterization of both situation data and template construction.

Figure 9.10 details the same problems as Figure 9.9, but refers to the CPU seconds for problem solution in place of constraint checks.

9.3.3 Full test results

In figures 9.11, 9.12, 9.13, 9.14, and 9.15 we present the test results charts for the experiments described above. One major consequence of these results is to show the effectiveness of this approach under certain constrained conditions, even when starting from scratch, ie before node classification of variables by attribute. The total time in the charts is between initial problem generation and final solution, including node consistency work, arc consistency work, and all search.

Columns charted are as follows: “Size” representing the noise, “Mo” search mode, “NC” node constraint checking on, “AC” arc constraint checking on, “FC” forward checking on, “DR” dynamic rearrangement on, “Ident” the unique identifier of the test case, “Dsize” the average domain size after pre-search propagation, “NCC” the amount of constraint checks executed during node consistency work, “TCC” the total number of constraint checks made during AC propagation and/or search, “BT/V” the number of search nodes encountered: backtrack nodes/nodes visited, “AP” the amount of pre-search processing CPU seconds, “S”

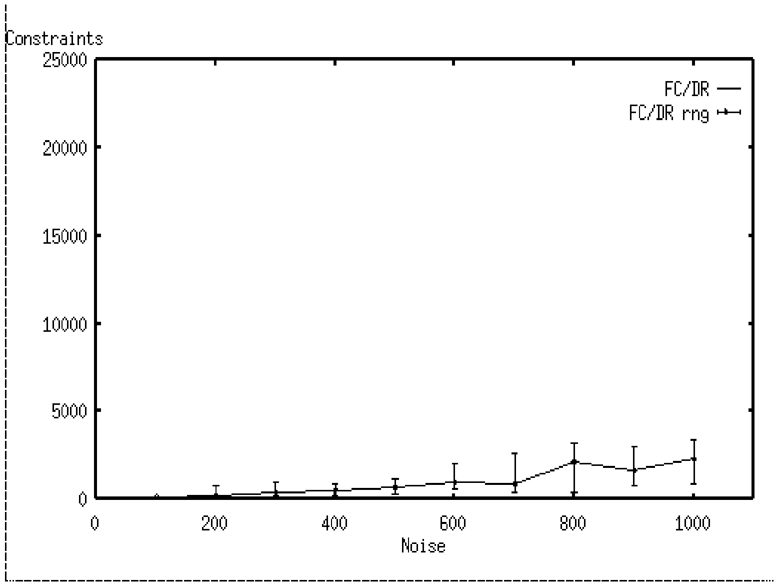


Figure 9.2: FC/DR constraint checks.

the amount of search processing CPU seconds, "Total" the sum of "AP" and "S", NumS the number of solutions found where there is only 1 "true" solution, any others are the product of noise, "BJ" the number of BackJumps during search (if applicable), "Type A/B" represents the number of constraint checks avoided during BackMarking (if applicable), "FCcost" the amount of work performed specifically in the line of Forward Checking, "DRcost" the amount of work performed in Dynamic Rearrangement, "AS/AStime" the mode and time cost for Advance Sorting of domain variables, and "ConsSort" the mode used for selecting or ordering constraints for application of consistency checking between domain values. For further details, please refer to the source code.

Figure 9.11 represents Forward Checking and Dynamic Rearrangement search, Figure 9.12 represents standard Backtracking, Figure 9.13 represents BackJumping, Figure 9.14 represents BackMarking, and Figure 9.15 represents AC3 propagation before search. Since AC3 propagation reduces the space to require almost no search, only one chart is shown, that of standard BackTracking.

9.4 Conclusions

As discussed elsewhere in this document, we can conclude the following on the basis of the few experiemnts shown:

1. Recognition of templates such as our example test case can be accomplished in reasonable time periods, even using only the simple configurations of MOTORS such as Forward Checking, or AC propagation.

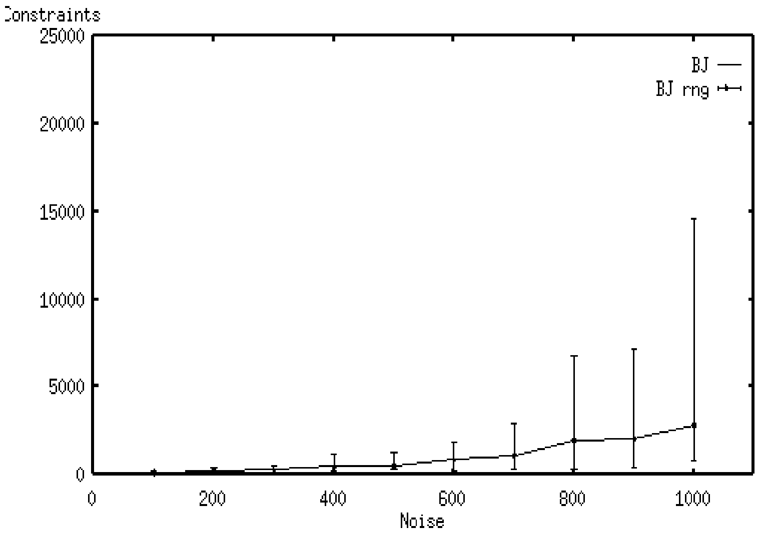


Figure 9.3: BackJump constraint checks.

2. Our test cases have the advantage of dramatically reduced variable domains after application of node-consistency preprocessing. Column "Dsize" in the summary charts shows the average domain size at this point. Much larger domain sizes resulting from a greater number of "potential" matches with variables will create more difficult problems.

3. According to the above observation, we must better characterize actual data in order to better predict real-world performance.

4. In these more complex cases, we need to evaluate the performance of other AC and partial AC algorithms in order to better design and creat hybrid solution strategies, and abstract based strategies.

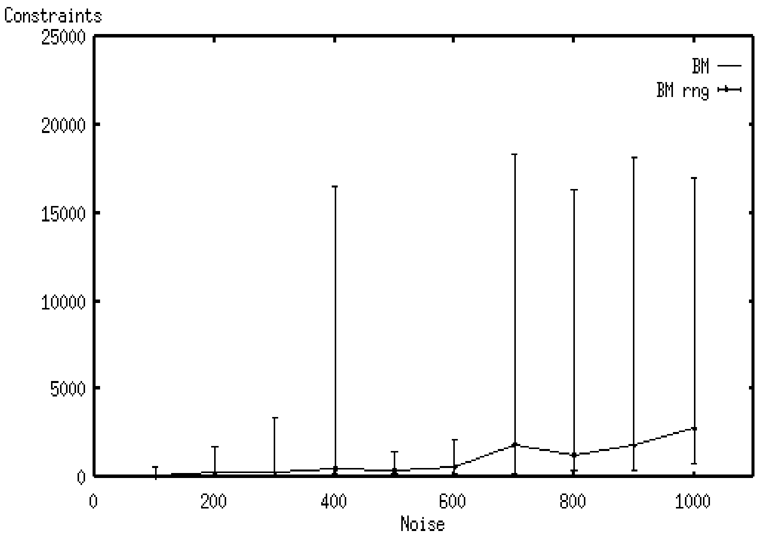


Figure 9.4: BackMark constraint checks.

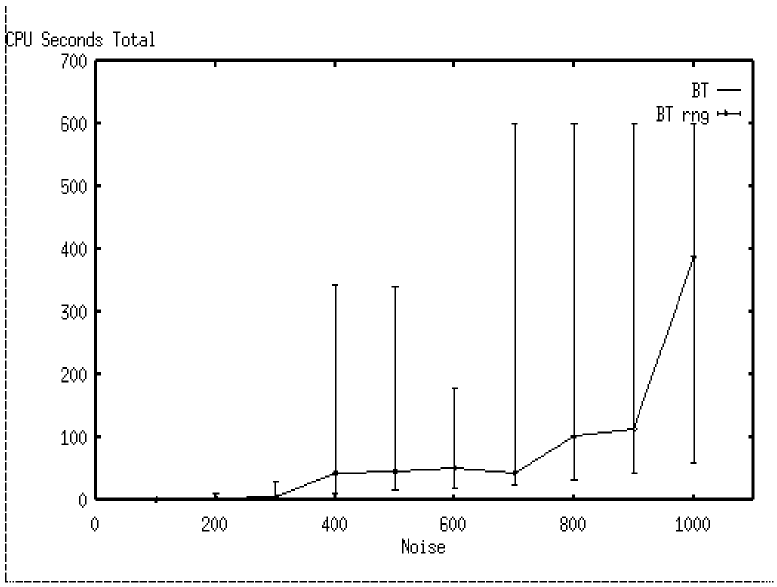


Figure 9.5: BackTrack CPU seconds (total).

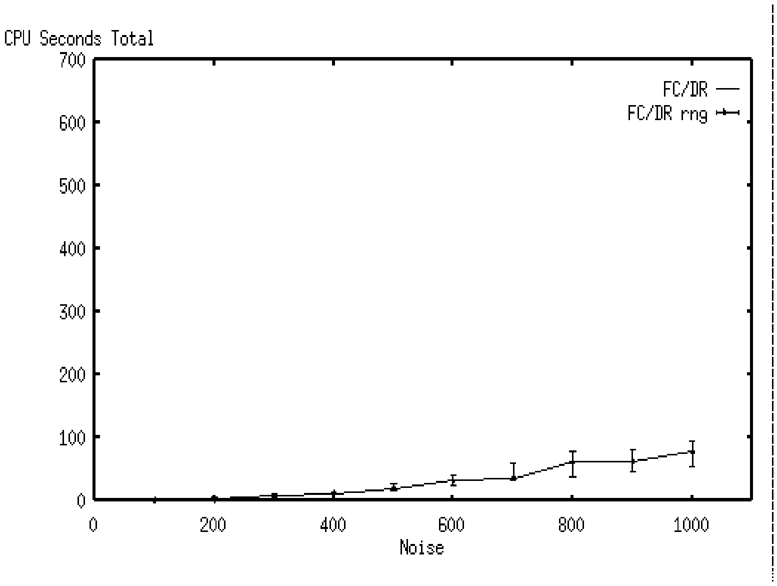


Figure 9.6: FC/DR CPU seconds (total).

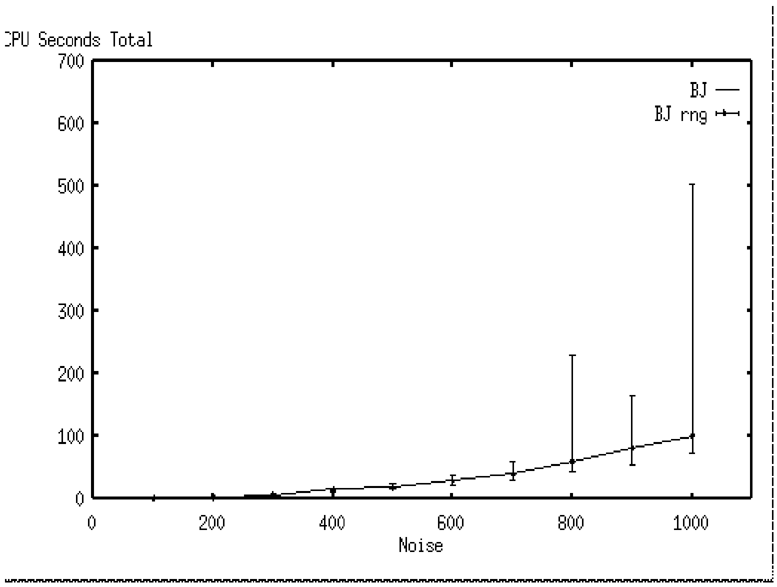


Figure 9.7: BackJump CPU seconds (total).

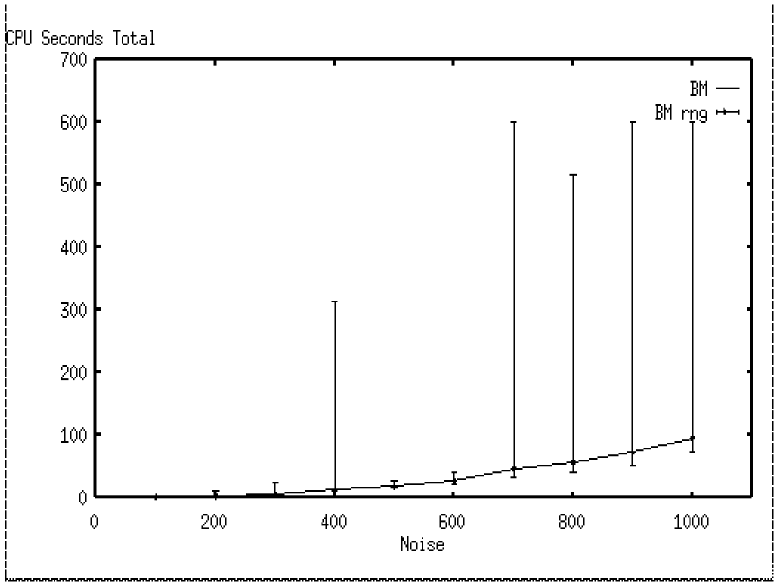


Figure 9.8: BackMark CPU seconds (total).

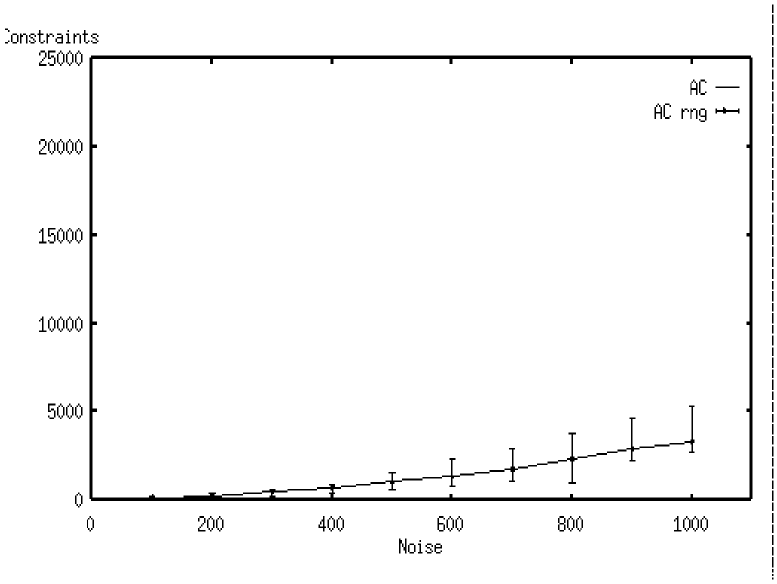


Figure 9.9: Arc Consistent (AC3) constraint checks.

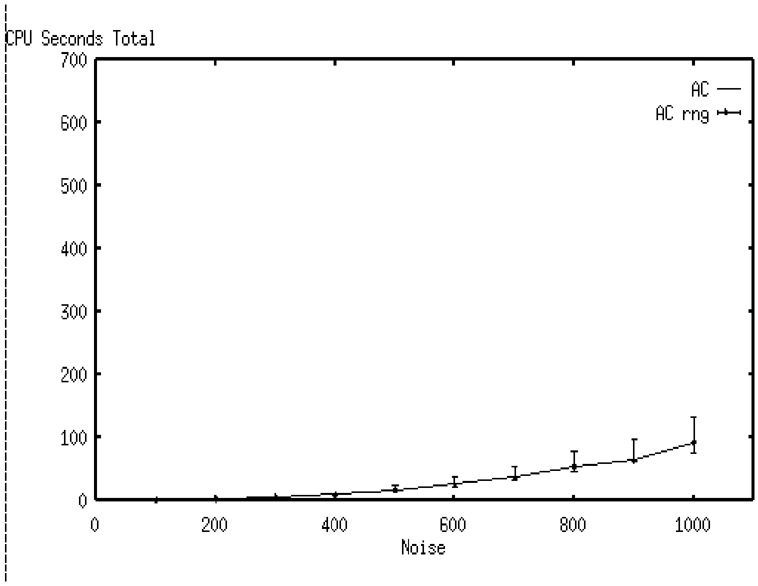


Figure 9.10: Arc Consistent (AC3) CPU seconds (total).

Size	Mo/NC/	AC/ FC/	DR	Ident	Dsize	NCC	TCC	BT/V	AP	S	Total	NumS	BJ	TypA/B	FCcost	DRcost	AS/AStime	ConsSort
100	bt/T/	NIL/T/	T	1414384866	6.2	781	116	6/7	0.65	0.15	0.8	1	0	0/0	90	12	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	1621232424	4.8	760	99	11/12	0.62	0.08	0.7	1	0	0/0	142	27	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	1972691557	5.8	755	97	9/10	0.62	0.09	0.71	1	0	0/0	154	21	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	2028154060	7.4	788	124	11/12	0.68	0.13	0.82	1	0	0/0	255	27	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	2175312090	5.6	757	134	12/13	0.62	0.13	0.75	1	0	0/0	184	29	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	2258260002	5.6	774	112	6/7	0.65	0.13	0.78	1	0	0/0	81	12	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	4997565789	7.8	805	138	10/11	0.63	0.25	0.88	1	0	0/0	266	24	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	5192471197	5.8	778	88	7/8	0.67	0.12	0.79	1	0	0/0	118	15	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	5448756273	7.4	797	130	12/13	0.7	0.16	0.86	1	0	0/0	288	30	RANDOM/0.0	RANDOM
100	bt/T/	NIL/T/	T	8634259052	6.8	798	105	8/9	0.62	0.16	0.78	1	0	0/0	167	18	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	1414384866	10.0	1446	242	7/8	2.3	0.55	2.85	1	0	0/0	190	15	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	1621232424	8.0	1411	175	14/15	2.35	0.28	2.63	1	0	0/0	340	36	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	1972691557	7.8	1407	109	9/10	2.34	0.15	2.49	1	0	0/0	224	21	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	2028154060	9.6	1423	322	13/14	2.32	0.83	3.15	1	0	0/0	400	33	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	2175312090	9.8	1418	356	18/19	2.29	0.64	2.94	1	0	0/0	527	47	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	2258260002	8.2	1417	168	13/14	2.33	0.3	2.63	1	0	0/0	345	32	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	4997565789	12.2	1479	762	16/17	2.3	2.73	5.03	1	0	0/0	630	42	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	5192471197	7.8	1418	154	11/12	2.28	0.27	2.55	1	0	0/0	267	26	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	5448756273	9.2	1422	189	14/15	2.32	0.4	2.72	1	0	0/0	413	35	RANDOM/0.0	RANDOM
200	bt/T/	NIL/T/	T	8634259052	9.2	1444	210	7/8	2.3	0.6	2.9	1	0	0/0	177	15	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	1414384866	12.8	2094	385	8/9	5.23	1.46	6.69	1	0	0/0	304	18	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	1621232424	9.6	2038	278	8/9	5.11	0.81	5.92	1	0	0/0	221	18	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	1972691557	9.8	2040	212	12/13	5.1	0.48	5.58	1	0	0/0	385	29	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	2028154060	11.8	2076	534	26/27	5.31	2.63	7.94	1	0	0/0	850	72	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	2175312090	12.6	2072	416	19/21	5.17	1.27	6.44	2	0	0/0	785	51	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	2258260002	12.4	2081	455	10/12	5.1	1.93	7.03	2	0	0/0	395	24	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	4997565789	14.6	2115	1001	17/18	5.11	6.32	11.43	1	0	0/0	832	45	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	5192471197	11.4	2077	471	11/12	5.14	2.22	7.36	1	0	0/0	400	27	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	5448756273	10.2	2044	179	13/14	5.15	0.52	5.67	1	0	0/0	441	33	RANDOM/0.0	RANDOM
300	bt/T/	NIL/T/	T	8634259052	12.0	2096	466	10/11	5.16	2.27	7.43	1	0	0/0	378	24	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	1414384866	14.6	2720	692	22/23	9.25	4.0	13.25	1	0	0/0	1058	60	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	1621232424	11.4	2672	367	17/18	9.08	1.68	10.76	1	0	0/0	652	45	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	1972691557	11.8	2671	236	12/13	9.09	0.62	9.71	1	0	0/0	486	30	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	2028154060	15.8	2747	759	31/32	9.2	5.24	14.44	1	0	0/0	1462	87	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	2175312090	15.2	2723	474	20/22	9.12	2.1	11.22	2	0	0/0	1048	54	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	2258260002	15.0	2726	888	28/30	9.11	5.33	14.44	2	0	0/0	1271	75	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	4997565789	17.0	2754	559	18/19	9.21	2.78	11.99	1	0	0/0	1097	48	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	5192471197	14.6	2732	296	14/15	9.12	1.0	10.12	1	0	0/0	730	36	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	5448756273	12.4	2682	508	11/12	9.1	2.48	11.58	1	0	0/0	438	27	RANDOM/0.0	RANDOM
400	bt/T/	NIL/T/	T	8634259052	14.0	2725	828	23/24	9.2	4.63	13.83	1	0	0/0	1022	63	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	1414384866	17.4	3367	1001	14/15	14.21	7.33	21.54	1	0	0/0	844	36	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	1621232424	13.4	3299	523	20/21	14.29	3.23	17.52	1	0	0/0	882	54	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	1972691557	13.8	3319	245	12/13	14.39	0.94	15.33	1	0	0/0	586	30	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	2028154060	18.6	3385	1112	24/25	14.24	7.83	22.07	1	0	0/0	1578	66	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	2175312090	17.2	3360	738	11/13	14.21	6.39	20.6	2	0	0/0	633	27	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	2258260002	18.6	3400	567	20/22	14.49	2.81	17.3	2	0	0/0	1348	53	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	4997565789	19.0	3399	1044	24/25	15.26	11.3	26.56	1	0	0/0	1599	65	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	5192471197	18.4	3394	602	20/21	15.79	2.57	18.36	1	0	0/0	1323	53	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	5448756273	15.2	3325	419	17/18	14.59	1.99	16.58	1	0	0/0	908	44	RANDOM/0.0	RANDOM
500	bt/T/	NIL/T/	T	8634259052	16.8	3370	1176	27/28	14.59	8.84	23.43	1	0	0/0	1466	75	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	1414384866	20.2	4014	649	20/21	21.03	3.84	24.87	1	0	0/0	1487	54	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	1621232424	14.4	3921	558	10/11	20.78	4.89	25.67	1	0	0/0	461	24	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	1972691557	16.4	3951	919	14/15	20.94	9.92	30.86	1	0	0/0	790	36	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	2028154060	20.0	4011	1446	37/38	20.81	15.43	36.24	1	0	0/0	2280	105	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	2175312090	19.6	4008	962	12/14	21.05	10.27	31.32	2	0	0/0	810	30	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	2258260002	20.8	4038	583	20/22	20.72	3.57	24.29	2	0	0/0	1546	53	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	4997565789	23.0	4058	2025	38/39	20.87	20.17	41.04	1	0	0/0	2870	108	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	5192471197	19.6	4025	1263	26/27	20.87	11.74	32.61	1	0	0/0	1761	72	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	5448756273	18.6	3977	1052	27/28	23.16	9.32	32.49	1	0	0/0	1726	75	RANDOM/0.0	RANDOM
600	bt/T/	NIL/T/	T	8634259052	19.6	4012	1704	36/37	21.37	15.67	37.04	1	0	0/0	2173	102	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	1414384866	23.4	4659	891	24/25	29.38	6.74	36.13	1	0	0/0	2139	66	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	1621232424	17.2	4582	445	19/20	29.69	3.19	32.88	1	0	0/0	1172	51	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	1972691557	18.2	4582	343	14/15	29.51	2.17	31.68	1	0	0/0	945	36	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	2028154060	24.4	4693	2117	42/43	29.35	26.43	55.78	1	0	0/0	3315	120	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	2175312090	21.2	4646	683	23/25	28.94	5.84	34.78	2	0	0/0	1788	63	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	2258260002	23.0	4683	2005	40/42	29.22	21.53	50.77	2	0	0/0	2961	114	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	4997565789	26.4	4719	2593	42/43	28.72	32.23	60.96	1	0	0/0	3671	120	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	5192471197	22.0	4656	642	22/23	29.05	4.58	33.63	1	0	0/0	1791	59	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	5448756273	22.6	4646	873	25/26	29.57	5.87	35.44	1	0	0/0	2063	68	RANDOM/0.0	RANDOM
700	bt/T/	NIL/T/	T	8634259052	21.8	4665	2147	39/40	29.55	25.12	54.68	1	0	0/0	2658	111	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	1414384866	27.6	5329	2552	21/22	38.43	36.23	74.66	1	0	0/0	2169	57	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	1621232424	20.6	5236	1084	13/14	38.89	12.77	51.66	1	0	0/0	936	33	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	1972691557	20.4	5242	408	15/16	37.8	2.78	40.58	1	0	0/0	1153	39	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	2028154060	26.4	5326	2473	38/39	38.25	30.58	68.83	1	0	0/0	3441	108	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	2175312090	23.8	5298	1880	31/33	41.06	22.38	63.45	2	0	0/0	2594	87	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	2258260002	25.6	5323	2537	44/46	41.55	37.06	78.61	2	0	0/0	3561	120	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	4997565789	30.2	5379	3232	24/25	30.71	47.72	78.44	1	0	0/0	2739	66	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	5192471197	25.0	5312	2254	48/49	32.26	27.57	59.83	1	0	0/0	3642	138	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	5448756273	25.0	5293	1991	39/40	32.03	23.31	55.34	1	0	0/0	3280	110	RANDOM/0.0	RANDOM
800	bt/T/	NIL/T/	T	8634259052	24.6	5311	769	20/21	32.16	6.33	38.49	1	0	0/0	1870	53	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	1414384866	30.0	5962	2878	50/51	40.37	39.66	80.03	1	0	0/0	4908	139	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	1621232424	22.8	5873	751	22/23	41.23	5.59	46.82	1	0	0/0	1868	59	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	1972691557	22.6	5883	1510	16/17	40.17	22.51	62.68	1	0	0/0	1309	42	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	2028154060	28.8	5965	3020	49/50	37.84	41.29	79.13	1	0	0/0	4548	140	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	2175312090	26.6	5953	1701	15/17	38.25	23.45	61.7	2	0	0/0	1462	39	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	2258260002	28.2	5973	1151	27/29	39.14	9.47	48.61	2	0	0/0	2892	73	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	4997565789	32.4	6009	1624	29/30	37.94	15.77	53.71	1	0	0/0	3602	79	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	5192471197	28.4	5969	2656	37/38	37.66	29.43	67.09	1	0	0/0	3722	105	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	5448756273	26.2	5913	2222	38/39	38.84	24.14	62.98	1	0	0/0	3408	108	RANDOM/0.0	RANDOM
900	bt/T/	NIL/T/	T	8634259052	26.8	5939	866	20/21	38.77	6.44	45.21	1	0	0/0	2065	53	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	1414384866	33.2	6624	1947	35/36	47.48	20.19	67.67	1	0	0/0	4410	95	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	1621232424	25.0	6506	2306	37/38	47.77	26.61	74.38	1	0	0/0	3113	104	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	1972691557	24.6	6516	2470	40/41	46.73	33.26	79.99	1	0	0/0	3160	110	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	2028154060	31.4	6602	1331	28/29	46.94	11.67	58.62	1	0	0/0	3383	77	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	2175312090	30.0	6622	3051	53/55	48.0	40.16	88.16	2	0	0/0	5101	150	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	2258260002	31.8	6648	3403	52/54	47.15	47.97	95.12	2	0	0/0	5394	143	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	4997565789	35.0	6656	1833	31/32	49.09	21.74	70.83	1	0	0/0	4188	85	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	5192471197	30.4	6607	3181	42/43	47.89	39.38	87.27	1	0	0/0	4505	120	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	5448756273	29.4	6575	2296	18/19	47.43	33.23	80.66	1	0	0/0	1986	48	RANDOM/0.0	RANDOM
1000	bt/T/	NIL/T/	T	8634259052	29.0	6572	871	22/23	47.79	7.53	55.32	1	0	0/0	2474	59	RANDOM/0.0	RANDOM

Figure 9.11: Forward Checking, Dynamic Rearrangement results.

Size	Mo/NC/ AC/ FC/ DR	Ident	Dsize	NCC	TCC	BT/V	AP	S	Total	NumS	BJ	TypA/B	FCcost	DRcost	AS/AStime	ConsS
100	bt/T/ NIL/NIL/NIL	1414384866	6.2	781	69	8/57	0.75	0.14	0.9	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	1621232424	4.8	760	81	34/91	0.68	0.12	0.8	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	1972691557	5.8	755	288	50/269	0.64	0.51	1.15	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	2028154060	7.4	788	315	79/324	0.7	0.52	1.22	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	2175312090	5.6	757	417	81/421	0.63	0.74	1.37	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	2258260002	5.6	774	46	6/36	0.65	0.06	0.71	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	4997565789	7.8	805	699	107/667	0.65	1.75	2.4	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	5192471197	5.8	778	54	7/41	0.66	0.08	0.74	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	5448756273	7.4	797	588	69/582	0.72	1.57	2.3	1	0	0/0	0	0	RAND/0.0	RAND
100	bt/T/ NIL/NIL/NIL	8634259052	6.8	798	477	60/446	0.69	1.11	1.8	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	1414384866	10.0	1446	120	8/94	2.39	0.21	2.61	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	1621232424	8.0	1411	1072	111/926	2.48	3.41	5.89	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	1972691557	7.8	1407	1017	91/864	2.42	3.72	6.15	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	2028154060	9.6	1423	246	13/201	2.56	0.75	3.31	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	2175312090	9.8	1418	165	18/134	2.41	0.36	2.77	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	2258260002	8.2	1417	1081	133/1061	2.36	3.66	6.02	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	4997565789	12.2	1479	251	16/205	2.35	0.71	3.06	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	5192471197	7.8	1418	611	87/569	2.33	1.52	3.85	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	5448756273	9.2	1422	1727	151/1600	2.3	8.38	10.68	1	0	0/0	0	0	RAND/0.0	RAND
200	bt/T/ NIL/NIL/NIL	8634259052	9.2	1444	155	12/130	2.28	0.45	2.73	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	1414384866	12.8	2094	239	13/190	5.34	0.73	6.07	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	1621232424	9.6	2038	107	8/92	5.28	0.36	5.64	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	1972691557	9.8	2040	1885	113/1536	5.2	10.55	15.75	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	2028154060	11.8	2076	3288	240/2777	5.12	21.07	26.19	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	2175312090	12.6	2072	4003	300/3444	5.17	23.85	29.02	2	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	2258260002	12.4	2081	217	13/177	5.16	0.59	5.75	2	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	4997565789	14.6	2115	306	18/255	5.34	1.24	6.58	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	5192471197	11.4	2077	278	16/246	5.1	1.64	6.74	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	5448756273	10.2	2044	2199	161/2020	5.13	14.7	19.83	1	0	0/0	0	0	RAND/0.0	RAND
300	bt/T/ NIL/NIL/NIL	8634259052	12.0	2096	135	10/114	5.19	0.41	5.6	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	1414384866	14.6	2720	2154	287/1915	9.33	18.24	27.57	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	1621232424	11.4	2672	5638	422/5206	9.14	49.83	58.97	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	1972691557	11.8	2671	157	12/139	9.43	0.55	9.98	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	2028154060	15.8	2747	600	32/490	9.26	3.62	12.89	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	2175312090	15.2	2723	16100	731/12815	9.26	290.79	300.05	2	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	2175312090	15.2	2723	17164	777/13666	9.26	334.3	343.56	2	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	2258260002	15.0	2726	505	40/450	9.3	3.31	12.61	2	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	4997565789	17.0	2754	14071	776/11782	9.32	253.28	262.6	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	5192471197	14.6	2732	429	34/373	10.46	1.87	12.33	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	5448756273	12.4	2682	216	13/174	9.18	0.87	10.05	1	0	0/0	0	0	RAND/0.0	RAND
400	bt/T/ NIL/NIL/NIL	8634259052	14.0	2725	5407	219/4511	9.34	51.52	60.86	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	1414384866	17.4	3367	583	23/524	15.03	5.67	20.7	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	1621232424	13.4	3299	5067	284/4245	14.69	52.13	66.83	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	1972691557	13.8	3319	3167	314/3074	14.61	20.5	35.11	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	2028154060	18.6	3385	307	24/273	14.52	2.38	16.9	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	2175312090	17.2	3360	685	29/572	14.58	5.79	20.37	2	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	2258260002	18.6	3400	15315	1216/14212	14.61	326.88	341.49	2	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	4997565789	19.0	3399	3877	209/3584	14.61	53.49	68.1	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	5192471197	18.4	3394	4990	370/4283	14.91	42.36	57.27	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	5448756273	15.2	3325	247	17/205	14.64	1.24	15.88	1	0	0/0	0	0	RAND/0.0	RAND
500	bt/T/ NIL/NIL/NIL	8634259052	16.8	3370	8894	311/7427	16.16	127.41	143.57	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	1414384866	20.2	4014	13094	773/11276	17.22	193.4	210.62	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	1621232424	14.4	3921	181	10/156	17.9	0.93	18.83	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	1972691557	16.4	3951	264	16/222	17.6	1.17	18.77	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	2028154060	20.0	4011	631	48/582	17.24	6.53	23.77	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	2175312090	19.6	4008	355	13/301	18.24	2.67	20.91	2	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	2258260002	20.8	4038	12134	1068/10530	17.17	160.34	177.51	2	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	4997565789	23.0	4058	5453	251/4595	17.14	63.95	81.09	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	5192471197	19.6	4025	4986	393/4487	16.67	62.42	79.09	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	5448756273	18.6	3977	1625	82/1330	16.56	9.28	25.85	1	0	0/0	0	0	RAND/0.0	RAND
600	bt/T/ NIL/NIL/NIL	8634259052	19.6	4012	6459	431/5617	16.83	80.02	96.86	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	1414384866	23.4	4659	523	25/446	22.76	3.03	25.79	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	1621232424	17.2	4582	3169	386/3000	22.97	24.59	47.57	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	1972691557	18.2	4582	279	15/232	22.75	1.59	24.34	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	2028154060	24.4	4693	5049	279/4766	23.09	70.18	93.27	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	2175312090	21.2	4646	311	23/277	22.81	2.09	24.9	2	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
700	bt/T/ NIL/NIL/NIL	2258260002	23.0	4683	23117	777/18114	22.84	577.26	600.11	2	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	4997565789	26.4	4719	3894	145/3263	22.75	46.73	69.48	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
700	bt/T/ NIL/NIL/NIL	5192471197	22.0	4656	20652	768/17765	22.57	578.57	601.14	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	5448756273	22.6	4646	459	26/393	24.36	2.74	27.1	1	0	0/0	0	0	RAND/0.0	RAND
700	bt/T/ NIL/NIL/NIL	8634259052	21.8	4665	1951	65/1597	22.62	17.48	40.1	1	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	1414384866	27.6	5329	992	30/818	30.31	7.39	37.7	1	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	1621232424	20.6	5236	396	16/329	29.96	2.51	32.47	1	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	1972691557	20.4	5242	506	31/450	29.88	3.06	32.94	1	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	2028154060	26.4	5326	3276	124/2702	29.7	34.92	64.62	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
800	bt/T/ NIL/NIL/NIL	2175312090	23.8	5298	20809	512/17123	30.03	570.0	600.03	2	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	2258260002	25.6	5323	9784	778/9830	29.96	269.24	299.21	2	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	4997565789	30.2	5379	1249	24/1031	29.91	16.33	46.24	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
800	bt/T/ NIL/NIL/NIL	5192471197	25.0	5312	17004	1272/17079	30.78	569.26	600.04	1	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	5448756273	25.0	5293	7589	395/7118	31.08	108.34	139.42	1	0	0/0	0	0	RAND/0.0	RAND
800	bt/T/ NIL/NIL/NIL	8634259052	24.6	5311	11170	550/9741	29.97	167.62	197.59	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
900	bt/T/ NIL/NIL/NIL	1414384866	30.0	5962	16695	744/15956	39.44	560.59	600.03	1	0	0/0	0	0	RAND/0.0	RAND
900	bt/T/ NIL/NIL/NIL	1621232424	22.8	5873	8125	544/6895	39.3	100.64	139.94	1	0	0/0	0	0	RAND/0.0	RAND
900	bt/T/ NIL/NIL/NIL	1972691557	22.6	5883	519	23/439	38.57	4.2	42.77	1	0	0/0	0	0	RAND/0.0	RAND
900	bt/T/ NIL/NIL/NIL	2028154060	28.8	5965	1906	86/1744	38.09	26.27	64.36	1	0	0/0	0	0	RAND/0.0	RAND
900	bt/T/ NIL/NIL/NIL	2175312090	26.6	5953	704	18/594	38.95	7.79	46.74	2	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
900	bt/T/ NIL/NIL/NIL	2258260002	28.2	5973	20312	551/16477	38.29	561.76	600.05	2	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
900	bt/T/ NIL/NIL/NIL	4997565789	32.4	6009	20748	689/16898	39.18	560.87	600.05	1	0	0/0	0	0	RAND/0.0	RAND
900	bt/T/ NIL/NIL/NIL	5192471197	28.4	5969	2163	40/1798	39.32	22.9	62.22	1	0	0/0	0	0	RAND/0.0	RAND
900	bt/T/ NIL/NIL/NIL	5448756273	26.2	5913	3218	128/2760	38.04	39.1	77.14	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
900	bt/T/ NIL/NIL/NIL	8634259052	26.8	5939	19428	567/16511	38.23	561.88	600.11	1	0	0/0	0	0	RAND/0.0	RAND
1000	bt/T/ NIL/NIL/NIL	1414384866	33.2	6624	1083	35/893	49.13	10.73	59.86	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
1000	bt/T/ NIL/NIL/NIL	1621232424	25.0	6506	20064	408/16329	47.76	552.27	600.03	1	0	0/0	0	0	RAND/0.0	RAND
1000	bt/T/ NIL/NIL/NIL	1972691557	24.6	6516	6786	353/6318	48.57	129.11	177.68	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
1000	bt/T/ NIL/NIL/NIL	2028154060	31.4	6602	18682	342/15443	48.68	551.35	600.03	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
1000	bt/T/ NIL/NIL/NIL	2175312090	30.0	6622	19503	664/16756	47.51	552.52	600.03	2	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
1000	bt/T/ NIL/NIL/NIL	2258260002	31.8	6648	19708	591/16668	47.73	553.86	601.6	2	0	0/0	0	0	RAND/0.0	RAND
1000	bt/T/ NIL/NIL/NIL	4997565789	35.0	6656	1471	37/1208	47.73	19.88	67.61	1	0	0/0	0	0	RAND/0.0	RAND
1000	bt/T/ NIL/NIL/NIL	5192471197	30.4	6607	780	42/684	49.37	12.37	61.74	1	0	0/0	0	0	RAND/0.0	RAND
1000	bt/T/ NIL/NIL/NIL	5448756273	29.4	6575	890	19/775	47.45	12.65	60.11	1	0	0/0	0	0	RAND/0.0	RAND
> Time limit bound exceeded < 600 >																
1000	bt/T/ NIL/NIL/NIL	8634259052	29.0	6572	18990	379/15962	47.94	552.09	600.03	1	0	0/0	0	0	RAND/0.0	RAND

Figure 9.12: BackTracking results.

Size	Mo/NC/ AC/ FC/ DR	Ident	Dsize	NCC	TCC	BT/V	AP	S	Total	NumS	BJ	TypA/B	FCcost	DRcost	AS/AStime	ConsS
100	bj/T/ NIL/NIL/NIL	1414384866	6.2	781	63	8/51	0.76	0.08	0.84	1	1	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	1621232424	4.8	760	44	20/49	0.68	0.06	0.74	1	7	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	1972691557	5.8	755	101	20/89	0.64	0.12	0.76	1	5	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	2028154060	7.4	788	121	28/107	0.65	0.15	0.8	1	6	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	2175312090	5.6	757	136	37/129	0.65	0.17	0.82	1	13	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	2258260002	5.6	774	46	6/36	0.63	0.06	0.69	1	0	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	4997565789	7.8	805	205	43/187	0.64	0.41	1.05	1	15	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	5192471197	5.8	778	54	7/41	0.66	0.08	0.74	1	0	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	5448756273	7.4	797	166	34/167	0.72	0.31	1.03	1	5	0/0	0	0	RAND/0.0	RAND
100	bj/T/ NIL/NIL/NIL	8634259052	6.8	798	118	28/102	0.69	0.21	0.9	1	12	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	1414384866	10.0	1446	112	8/88	2.29	0.24	2.53	1	1	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	1621232424	8.0	1411	211	49/193	2.37	0.4	2.77	1	10	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	1972691557	7.8	1407	221	31/181	2.38	0.6	2.98	1	6	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	2028154060	9.6	1423	246	13/201	2.48	0.77	3.26	1	0	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	2175312090	9.8	1418	165	18/134	2.4	0.36	2.77	1	0	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	2258260002	8.2	1417	313	57/304	2.46	0.88	3.34	1	10	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	4997565789	12.2	1479	251	16/205	2.48	0.76	3.25	1	0	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	5192471197	7.8	1418	251	37/215	2.43	0.49	2.92	1	6	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	5448756273	9.2	1422	375	55/358	2.43	1.33	3.76	1	9	0/0	0	0	RAND/0.0	RAND
200	bj/T/ NIL/NIL/NIL	8634259052	9.2	1444	120	12/101	2.34	0.35	2.69	1	2	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	1414384866	12.8	2094	202	13/158	5.37	0.63	6.0	1	3	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	1621232424	9.6	2038	107	8/92	5.2	0.38	5.58	1	0	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	1972691557	9.8	2040	530	47/433	5.24	2.32	7.56	1	8	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	2028154060	11.8	2076	461	64/401	5.42	1.98	7.4	1	30	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	2175312090	12.6	2072	487	76/436	5.23	1.61	6.84	2	16	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	2258260002	12.4	2081	207	13/170	5.11	0.61	5.72	2	1	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	4997565789	14.6	2115	298	18/247	5.16	1.21	6.37	1	1	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	5192471197	11.4	2077	222	15/196	5.22	1.28	6.5	1	3	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	5448756273	10.2	2044	285	44/274	5.27	1.28	6.55	1	9	0/0	0	0	RAND/0.0	RAND
300	bj/T/ NIL/NIL/NIL	8634259052	12.0	2096	135	10/114	5.19	0.42	5.61	1	0	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	1414384866	14.6	2720	315	53/277	9.41	1.55	10.96	1	18	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	1621232424	11.4	2672	459	74/405	9.27	1.88	11.15	1	23	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	1972691557	11.8	2671	157	12/139	9.17	0.52	9.7	1	0	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	2028154060	15.8	2747	576	32/479	9.24	3.33	12.58	1	1	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	2175312090	15.2	2723	1145	100/941	9.23	5.65	14.89	2	17	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	2258260002	15.0	2726	422	38/389	9.31	2.73	12.04	2	1	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	4997565789	17.0	2754	848	67/669	9.25	3.92	13.18	1	16	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	5192471197	14.6	2732	262	24/222	9.39	0.98	10.37	1	8	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	5448756273	12.4	2682	197	13/153	9.17	0.65	9.82	1	1	0/0	0	0	RAND/0.0	RAND
400	bj/T/ NIL/NIL/NIL	8634259052	14.0	2725	728	59/593	9.17	3.81	12.98	1	27	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	1414384866	17.4	3367	443	21/390	14.51	3.65	18.16	1	5	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	1621232424	13.4	3299	512	63/424	14.3	2.71	17.01	1	29	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	1972691557	13.8	3319	409	57/351	14.55	1.57	16.12	1	23	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	2028154060	18.6	3385	307	24/273	14.64	2.16	16.8	1	0	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	2175312090	17.2	3360	375	20/311	14.49	2.69	17.18	2	5	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	2258260002	18.6	3400	1253	155/1133	15.74	5.98	21.72	2	63	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	4997565789	19.0	3399	1162	61/989	15.54	10.12	25.67	1	16	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	5192471197	18.4	3394	1279	97/1007	14.61	6.01	20.63	1	13	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	5448756273	15.2	3325	247	17/205	14.75	1.13	15.88	1	0	0/0	0	0	RAND/0.0	RAND
500	bj/T/ NIL/NIL/NIL	8634259052	16.8	3370	995	71/822	14.57	6.58	21.15	1	33	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	1414384866	20.2	4014	1074	76/864	20.95	5.18	26.13	1	14	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	1621232424	14.4	3921	181	10/156	21.59	1.05	22.64	1	0	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	1972691557	16.4	3951	257	16/213	21.43	1.39	22.82	1	1	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	2028154060	20.0	4011	587	48/545	21.78	7.57	29.35	1	1	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	2175312090	19.6	4008	355	13/301	21.47	3.31	24.78	2	0	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	2258260002	20.8	4038	1355	120/1055	21.74	7.35	29.09	2	16	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	4997565789	23.0	4058	1853	78/1514	21.16	15.99	37.16	1	17	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	5192471197	19.6	4025	606	63/527	22.41	5.78	28.19	1	22	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	5448756273	18.6	3977	1098	59/941	20.86	8.54	29.41	1	13	0/0	0	0	RAND/0.0	RAND
600	bj/T/ NIL/NIL/NIL	8634259052	19.6	4012	1471	112/1151	20.87	13.5	34.37	1	29	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	1414384866	23.4	4659	523	25/446	28.7	3.77	32.47	1	0	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	1621232424	17.2	4582	405	56/360	28.65	2.32	30.97	1	15	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	1972691557	18.2	4582	279	15/232	28.7	1.9	30.6	1	0	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	2028154060	24.4	4693	2048	88/1737	28.96	20.85	49.82	1	20	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	2175312090	21.2	4646	311	23/277	29.18	2.61	31.79	2	0	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	2258260002	23.0	4683	2065	176/1740	29.08	19.4	48.48	2	56	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	4997565789	26.4	4719	2252	69/1831	28.57	22.97	51.54	1	4	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	5192471197	22.0	4656	2930	220/2632	28.96	31.51	60.47	1	93	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	5448756273	22.6	4646	434	26/366	29.94	2.78	32.72	1	1	0/0	0	0	RAND/0.0	RAND
700	bj/T/ NIL/NIL/NIL	8634259052	21.8	4665	1688	53/1352	29.43	16.59	46.04	1	1	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	1414384866	27.6	5329	799	27/638	37.91	6.41	44.32	1	3	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	1621232424	20.6	5236	326	15/272	40.28	2.7	43.0	1	1	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	1972691557	20.4	5242	370	26/322	39.69	2.83	42.54	1	9	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	2028154060	26.4	5326	2294	64/1843	39.43	27.47	66.92	1	3	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	2175312090	23.8	5298	1859	100/1481	39.12	18.89	58.01	2	45	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	2258260002	25.6	5323	6784	537/6632	37.8	192.38	230.19	2	12	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	4997565789	30.2	5379	1249	24/1031	38.79	20.21	59.01	1	0	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	5192471197	25.0	5312	5086	615/5203	39.02	112.41	151.44	1	267	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	5448756273	25.0	5293	2051	90/1752	38.73	22.58	61.33	1	21	0/0	0	0	RAND/0.0	RAND
800	bj/T/ NIL/NIL/NIL	8634259052	24.6	5311	2008	102/1642	39.73	21.27	61.01	1	17	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	1414384866	30.0	5962	2955	116/2508	50.33	44.51	94.86	1	32	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	1621232424	22.8	5873	1326	96/1071	49.52	8.83	58.37	1	16	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	1972691557	22.6	5883	436	22/363	49.21	4.54	53.76	1	3	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	2028154060	28.8	5965	1765	86/1622	50.85	30.97	81.82	1	5	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	2175312090	26.6	5953	704	18/594	48.97	9.56	58.54	2	0	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	2258260002	28.2	5973	7140	253/5779	51.97	114.35	166.34	2	45	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	4997565789	32.4	6009	5407	194/4289	51.05	66.2	117.25	1	21	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	5192471197	28.4	5969	2046	40/1716	52.47	27.61	80.09	1	3	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	5448756273	26.2	5913	2031	65/1670	53.03	25.66	78.72	1	3	0/0	0	0	RAND/0.0	RAND
900	bj/T/ NIL/NIL/NIL	8634259052	26.8	5939	4879	252/4224	50.0	73.26	123.27	1	110	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	1414384866	33.2	6624	1083	35/893	63.07	14.0	77.09	1	0	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	1621232424	25.0	6506	3308	138/2729	59.63	49.11	108.74	1	66	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	1972691557	24.6	6516	2314	84/1921	58.76	34.03	92.79	1	19	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	2028154060	31.4	6602	5680	161/4693	61.07	100.9	161.97	1	29	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	2175312090	30.0	6622	5476	401/4739	62.68	95.74	158.42	2	142	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	2258260002	31.8	6648	14558	700/12331	60.27	443.92	504.19	2	240	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	4997565789	35.0	6656	1173	34/965	59.43	15.89	75.32	1	2	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	5192471197	30.4	6607	780	42/684	59.83	13.78	73.62	1	0	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	5448756273	29.4	6575	890	19/775	59.01	15.32	74.34	1	0	0/0	0	0	RAND/0.0	RAND
1000	bj/T/ NIL/NIL/NIL	8634259052	29.0	6572	5809	209/4985	59.23	105.72	164.95	1	88	0/0	0	0	RAND/0.0	RAND

Figure 9.13: BackJumping results.

Size	Mo/NC/ AC/ FC/ DR	Ident	Dsize	NCC	TCC	BT/V	AP	S	Total	NumS	BJ	TypA/B	FCcost	DRcost	AS/AStime	ConsSort
100	bm/T/ NIL/NIL/NIL	1414384866	6.2	781	69	8/57	0.65	0.13	0.78	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	1621232424	4.8	760	36	34/91	0.63	0.06	0.69	1	0	30/2	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	1972691557	5.8	755	62	50/269	0.61	0.12	0.74	1	0	174/6	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	2028154060	7.4	788	112	79/324	0.68	0.2	0.88	1	0	154/23	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	2175312090	5.6	757	108	81/421	0.64	0.2	0.85	1	0	254/37	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	2258260002	5.6	774	46	6/36	0.67	0.07	0.75	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	4997565789	7.8	805	161	107/667	0.76	0.48	1.24	1	0	430/108	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	5192471197	5.8	778	54	7/41	0.68	0.08	0.77	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	5448756273	7.4	797	588	69/582	0.66	1.33	1.99	1	0	0/378	0	0	RANDOM/0.0	RANDOM
100	bm/T/ NIL/NIL/NIL	8634259052	6.8	798	75	60/446	0.64	0.21	0.86	1	0	328/16	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	1414384866	10.0	1446	120	8/94	2.38	0.21	2.61	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	1621232424	8.0	1411	1067	111/926	2.38	3.07	5.45	1	0	5/595	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	1972691557	7.8	1407	975	91/864	2.33	3.34	5.67	1	0	29/661	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	2028154060	9.6	1423	246	13/201	2.31	0.72	3.03	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	2175312090	9.8	1418	165	18/134	2.49	0.36	2.86	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	2258260002	8.2	1417	960	133/1061	2.31	3.0	5.31	1	0	111/658	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	4997565789	12.2	1479	251	16/205	2.39	0.7	3.09	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	5192471197	7.8	1418	107	87/569	2.4	0.32	2.73	1	0	400/20	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	5448756273	9.2	1422	1727	151/1600	2.36	8.0	10.36	1	0	0/1320	0	0	RANDOM/0.0	RANDOM
200	bm/T/ NIL/NIL/NIL	8634259052	9.2	1444	113	12/130	2.39	0.34	2.73	1	0	27/3	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	1414384866	12.8	2094	197	13/190	5.39	0.6	5.99	1	0	29/2	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	1621232424	9.6	2038	107	8/92	5.22	0.33	5.55	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	1972691557	9.8	2040	1796	113/1536	5.26	9.75	15.01	1	0	54/1202	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	2028154060	11.8	2076	384	240/2777	5.1	1.99	7.1	1	0	2240/16	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	2175312090	12.6	2072	3353	300/3444	5.21	18.12	23.35	2	0	558/2146	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	2258260002	12.4	2081	181	13/177	5.09	0.47	5.57	2	0	28/3	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	4997565789	14.6	2115	306	18/255	5.31	1.18	6.5	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	5192471197	11.4	2077	236	16/246	5.16	1.42	6.58	1	0	27/1	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	5448756273	10.2	2044	2199	161/2020	5.12	14.29	19.41	1	0	0/1690	0	0	RANDOM/0.0	RANDOM
300	bm/T/ NIL/NIL/NIL	8634259052	12.0	2096	135	10/114	5.23	0.44	5.68	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	1414384866	14.6	2720	211	287/1915	9.23	1.68	11.14	1	0	1469/13	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	1621232424	11.4	2672	2098	422/5206	9.25	12.15	21.41	1	0	2795/1873	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	1972691557	11.8	2671	157	12/139	9.21	0.53	9.74	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	2028154060	15.8	2747	600	32/490	9.39	3.5	12.89	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	2175312090	15.2	2723	16570	777/13666	9.22	304.21	313.44	2	0	463/12351	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	2258260002	15.0	2726	421	40/450	9.21	2.97	12.19	2	0	63/3	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	4997565789	17.0	2754	525	776/11782	9.39	4.52	13.91	1	0	10570/488	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	5192471197	14.6	2732	273	34/373	9.58	1.02	10.61	1	0	116/2	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	5448756273	12.4	2682	216	13/174	9.46	0.77	10.27	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bm/T/ NIL/NIL/NIL	8634259052	14.0	2725	640	219/4511	9.46	4.26	13.72	1	0	3800/24	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	1414384866	17.4	3367	502	23/524	15.63	4.77	20.41	1	0	68/1	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	1621232424	13.4	3299	398	284/4245	14.42	2.99	17.42	1	0	3653/39	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	1972691557	13.8	3319	209	314/3074	14.79	1.32	16.11	1	0	2579/166	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	2028154060	18.6	3385	307	24/273	14.51	2.18	16.7	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	2175312090	17.2	3360	431	29/572	14.54	3.92	18.47	2	0	195/4	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	2258260002	18.6	3400	533	1216/14212	15.06	5.83	20.92	2	0	12537/835	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	4997565789	19.0	3399	1468	209/3584	14.47	11.26	25.76	1	0	2143/737	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	5192471197	18.4	3394	359	370/4283	14.79	2.51	17.31	1	0	3633/63	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	5448756273	15.2	3325	247	17/205	14.6	1.16	15.76	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bm/T/ NIL/NIL/NIL	8634259052	16.8	3370	829	311/7427	14.7	6.81	21.51	1	0	6450/30	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	1414384866	20.2	4014	633	773/11276	20.96	6.13	27.09	1	0	9955/360	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	1621232424	14.4	3921	181	10/156	21.13	1.07	22.21	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	1972691557	16.4	3951	210	16/222	21.18	1.18	22.36	1	0	38/2	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	2028154060	20.0	4011	631	48/582	22.47	8.13	30.61	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	2175312090	19.6	4008	326	13/301	21.44	3.07	24.52	2	0	26/1	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	2258260002	20.8	4038	552	1068/105 30	21.345.76		27.11	2	0	8984/643	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	4997565789	23.0	4058	2173	251/4595	21.08	20.29	41.37	1	0	2526/834	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	5192471197	19.6	4025	380	393/4487	21.24	4.71	25.95	1	0	3780/15	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	5448756273	18.6	3977	1027	82/1330	22.54	8.56	31.2	1	0	374/288	0	0	RANDOM/0.0	RANDOM
600	bm/T/ NIL/NIL/NIL	8634259052	19.6	4012	644	431/5617	21.14	7.18	28.34	1	0	4675/44	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	1621232424	17.2	4582	213	386/3000	29.03	2.13	31.19	1	0	2442/22	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	1972691557	18.2	4582	229	15/232	29.52	1.67	31.26	1	0	30/1	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	2028154060	24.4	4693	3053	279/4766	31.71	33.48	65.24	1	0	1853/1387	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	2175312090	21.2	4646	311	23/277	29.8	2.91	32.73	2	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	2258260002	23.0	4683	12903	2773/69501	29.91	267.37	297.28	2	0	55216/11703	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	4997565789	26.4	4719	1872	145/3263	29.39	19.57	48.99	1	0	1596/19	0	0	RANDOM/0.0	RANDOM
> Time limit bound exceeded		< 600 >														
700	bm/T/ NIL/NIL/NIL	5192471197	22.0	4656	18356	2198/49955	29.7	-33.31	-3.49	1	0	32813/15568	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	5448756273	22.6	4646	459	26/393	33.37	3.29	36.73	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bm/T/ NIL/NIL/NIL	8634259052	21.8	4665	1402	65/1597	30.29	16.05	46.35	1	0	396/12	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	1414384866	27.6	5329	823	30/818	38.72	7.66	46.38	1	0	133/2	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	1621232424	20.6	5236	375	16/329	39.14	3.13	42.29	1	0	18/0	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	1972691557	20.4	5242	369	31/450	38.91	2.63	41.55	1	0	106/2	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	2028154060	26.4	5326	1951	124/2702	40.45	24.41	64.87	1	0	1020/20	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	2175312090	23.8	5298	1529	690/2412 8	37.47	18.58	56.06	2	0	22222/134	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	2258260002	25.6	5323	1302	778/9830	37.39	19.94	57.34	2	0	7977/543	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	4997565789	30.2	5379	1249	24/1031	37.19	20.15	57.35	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	5192471197	25.0	5312	16352	2972/39977	38.49	477.39	515.89	1	0	21983/15682	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	5448756273	25.0	5293	3161	395/7118	39.83	35.61	75.45	1	0	3997/1703	0	0	RANDOM/0.0	RANDOM
800	bm/T/ NIL/NIL/NIL	8634259052	24.6	5311	671	550/9741	37.81	8.51	46.32	1	0	8640/187	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	1414384866	30.0	5962	7348	882/19068	52.7	124.53	177.24	1	0	11679/5225	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	1621232424	22.8	5873	429	544/6895	47.74	4.87	52.63	1	0	6008/58	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	1972691557	22.6	5883	437	23/439	48.64	4.71	53.35	1	0	68/2	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	2028154060	28.8	5965	1906	86/1744	47.33	31.8	79.14	1	0	0/154	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	2175312090	26.6	5953	595	18/594	47.51	8.61	56.13	2	0	84/3	0	0	RANDOM/0.0	RANDOM
> Time limit bound exceeded		< 600 >														
900	bm/T/ NIL/NIL/NIL	2258260002	28.2	5973	18138	653/19292	48.43	-2175.19	-2126.75	2	0	4437/13895	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	4997565789	32.4	6009	997	1202/29489	47.23	15.75	62.98	1	0	27504/144	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	5192471197	28.4	5969	2163	40/1798	48.09	27.78	75.87	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bm/T/ NIL/NIL/NIL	5448756273	26.2	5913	1681	128/2760	47.03	21.58	68.62	1	0	1239/21	0	0	RANDOM/0.0	RANDOM
> Time limit bound exceeded		< 600 >														
900	bm/T/ NIL/NIL/NIL	8634259052	26.8	5939	17573	1426/40848	47.1	-2986.88	-2939.78	1	0	25125/14618	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	1414384866	33.2	6624	1083	35/893	59.27	12.84	72.11	1	0	0/40	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	1621232424	25.0	6506	1901	682/2688 9	58.17	26.56	84.73	1	0	24701/85	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	1972691557	24.6	6516	3669	353/6318	59.2	48.06	107.26	1	0	2761/1724	0	0	RANDOM/0.0	RANDOM
> Time limit bound exceeded		< 600 >														
1000	bm/T/ NIL/NIL/NIL	2028154060	31.4	6602	17027	322/14529	58.16	-3867.8	-3809.62	1	0	396/13347	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	2175312090	30.0	6622	8034	2131/62248	59.13	157.31	216.45	2	0	53060/5782	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	2258260002	31.8	6648	10944	2694/82722	59.45	229.98	289.45	2	0	70688/8338	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	4997565789	35.0	6656	1184	37/1208	59.5	17.57	77.08	1	0	211/1	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	5192471197	30.4	6607	780	42/684	59.54	13.52	73.07	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bm/T/ NIL/NIL/NIL	5448756273	29.4	6575	890	19/775	58.39	15.3	73.7	1	0	0/0	0	0	RANDOM/0.0	RANDOM
> Time limit bound exceeded		< 600 >														
1000	bm/T/ NIL/NIL/NIL	8634259052	29.0	6572	16716	1049/44036	59.48	-5207.83	-5148.34	1	0	29432/13654	0	0	RANDOM/0.0	RANDOM

Figure 9.14: BackMarking results.

Size	Mo/NC/ AC/ FC/ DR	Ident	Dsize	NCC	TCC	BT/V	AP	S	Total	NumS	BJ	TypA/B	FCcost	DRcost	AS/AStime	ConsSort
100	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	781	111	4/5	0.69	0.01	0.7	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	760	74	4/5	0.57	0.0	0.57	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	755	124	4/5	0.62	0.0	0.62	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	788	164	4/5	0.68	0.01	0.69	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.0	757	101	4/5	0.59	0.01	0.6	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.0	774	98	4/5	0.59	0.0	0.6	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	805	186	4/5	0.72	0.01	0.73	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	778	113	4/5	0.68	0.0	0.68	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	797	151	4/5	0.65	0.01	0.66	1	0	0/0	0	0	RANDOM/0.0	RANDOM
100	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	798	121	4/5	0.66	0.02	0.68	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	1446	265	4/5	2.56	0.01	2.57	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	1411	158	4/5	2.22	0.02	2.24	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	1407	264	4/5	2.44	0.01	2.45	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	1423	309	4/5	2.65	0.0	2.65	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.0	1418	225	4/5	2.48	0.01	2.49	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.0	1417	230	4/5	2.44	0.01	2.45	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	1479	421	4/5	2.8	0.01	2.81	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	1418	200	4/5	2.41	0.01	2.42	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	1422	237	4/5	2.63	0.01	2.64	1	0	0/0	0	0	RANDOM/0.0	RANDOM
200	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	1444	247	4/5	2.49	0.01	2.5	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	2094	474	4/5	5.95	0.01	5.96	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	2038	243	4/5	5.02	0.01	5.03	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	2040	361	4/5	5.54	0.01	5.55	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	2076	460	4/5	6.09	0.01	6.1	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	2072	398	7/9	5.59	0.02	5.61	2	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	2081	542	8/10	5.77	0.01	5.78	2	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	2115	616	4/5	6.56	0.01	6.57	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	2077	432	4/5	5.7	0.0	5.71	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	2044	303	4/5	5.6	0.01	5.61	1	0	0/0	0	0	RANDOM/0.0	RANDOM
300	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	2096	505	4/5	6.03	0.0	6.03	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	2720	741	4/5	11.61	0.01	11.62	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	2672	364	4/5	9.24	0.01	9.25	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	2671	491	4/5	9.98	0.01	9.99	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	2747	769	4/5	11.67	0.01	11.68	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	2723	622	4/6	10.23	0.01	10.24	2	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	2726	917	7/9	11.93	0.02	11.95	2	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	2754	895	4/5	12.21	0.01	12.22	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	2732	699	4/5	10.71	0.0	10.71	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	2682	428	4/5	9.85	0.01	9.86	1	0	0/0	0	0	RANDOM/0.0	RANDOM
400	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	2725	734	4/5	11.18	0.01	11.19	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	3367	1210	4/5	20.21	0.01	20.22	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	3299	593	4/5	15.44	0.0	15.44	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	3319	688	4/5	15.78	0.0	15.78	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	3385	1246	4/5	20.05	0.01	20.06	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	3360	821	4/6	16.19	0.02	16.21	2	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	3400	1333	5/7	20.56	0.02	20.58	2	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	3399	1575	4/5	24.04	0.01	24.05	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	3394	1008	4/5	17.68	0.01	17.69	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	3325	671	4/5	16.81	0.0	16.81	1	0	0/0	0	0	RANDOM/0.0	RANDOM
500	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	3370	1045	4/5	18.52	0.0	18.52	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	4014	1652	4/5	30.77	0.01	30.78	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	3921	736	4/5	22.96	0.02	22.98	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	3951	998	4/5	23.96	0.0	23.96	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	4011	1518	4/5	29.97	0.0	29.97	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	4008	1066	7/9	24.26	0.03	24.29	2	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	4038	1700	6/8	30.37	0.03	30.4	2	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	4058	2340	4/5	39.02	0.01	39.03	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	4025	1185	4/5	25.68	0.01	25.69	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	3977	964	4/5	24.62	0.01	24.63	1	0	0/0	0	0	RANDOM/0.0	RANDOM
600	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	4012	1430	4/5	27.7	0.01	27.71	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	4659	2108	4/5	42.12	0.01	42.13	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	4582	1036	4/5	31.5	0.01	31.51	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	4582	1561	4/5	40.0	0.01	40.01	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	4693	2361	4/5	43.16	0.01	43.17	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	4646	1331	8/10	33.3	0.02	33.33	2	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	4683	2130	8/10	41.24	0.03	41.27	2	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	4719	2920	4/5	52.83	0.01	52.84	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	4656	1472	4/5	35.92	0.0	35.92	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	4646	1777	4/5	37.67	0.01	37.68	1	0	0/0	0	0	RANDOM/0.0	RANDOM
700	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	4665	1732	4/5	37.2	0.01	37.21	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	5329	2896	4/5	59.5	0.01	59.51	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	5236	1608	4/5	44.91	0.0	44.91	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	5242	2013	4/5	52.15	0.0	52.15	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	5326	2696	4/5	58.62	0.01	58.63	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	5298	2005	6/8	49.1	0.03	49.13	2	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	5323	2525	5/7	56.51	0.03	56.54	2	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	5379	3780	4/5	75.82	0.0	75.83	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	5312	2339	4/5	55.17	0.01	55.18	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	5293	2353	4/5	52.52	0.01	52.53	1	0	0/0	0	0	RANDOM/0.0	RANDOM
800	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	5311	2145	4/5	50.02	0.01	50.03	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	5962	3402	4/5	80.51	0.01	80.52	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	5873	2186	4/5	63.74	0.01	63.75	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	5883	2440	4/5	66.65	0.01	66.66	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	5965	3361	4/5	79.27	0.01	79.28	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	5953	2444	6/8	65.64	0.03	65.67	2	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	5973	3186	7/9	76.64	0.03	76.67	2	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	6009	4639	4/5	105.32	0.01	105.33	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	5969	2887	4/5	73.25	0.0	73.25	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	5913	2570	4/5	70.33	0.01	70.34	1	0	0/0	0	0	RANDOM/0.0	RANDOM
900	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	5939	2599	4/5	68.68	0.01	68.69	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	1414384866	1.0	6624	4209	4/5	107.05	0.01	107.06	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	1621232424	1.0	6506	2741	4/5	79.36	0.01	79.37	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	1972691557	1.0	6516	2792	4/5	87.31	0.02	87.33	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	2028154060	1.0	6602	4012	4/5	105.26	0.01	105.27	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	2175312090	1.2	6622	3224	4/6	90.56	0.03	90.59	2	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	2258260002	1.2	6648	3885	4/6	100.23	0.03	100.27	2	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	4997565789	1.0	6656	5283	4/5	134.24	0.02	134.26	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	5192471197	1.0	6607	3441	4/5	94.7	0.01	94.71	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	5448756273	1.0	6575	3146	4/5	88.01	0.01	88.02	1	0	0/0	0	0	RANDOM/0.0	RANDOM
1000	bt/T/ BEFORE/ NIL/ NIL	8634259052	1.0	6572	2984	4/5	85.68	0.01	85.69	1	0	0/0	0	0	RANDOM/0.0	RANDOM

Figure 9.15: AC3 results.

Chapter 10

Related Future Work and Investigation

Throughout the paper, different areas have been identified as rich research and application opportunities for extension of the model of template recognition. Many of these possible research areas have broad application across a much wider filed than just recognition, including heuristic search, constraint propagation, abstract reasoning, default reasoning, and vision. In this chapter we will briefly outline some areas that have arisen that have particularly broad application, or which appear to have application to our problem, but have not been examined in any depth previously in this paper.

Note that the work presented in Section 10.8 is presented by Denis McGonigal, summer student under the supervision of Steven Woods.

10.1 Temporal aspects of template recognition

Initially we set out to study the temporal aspects of arriving messages in determination of the existence of templates. Towards this, we first simplified the problem to examine static models, as described earlier. We will extend our model of recognition in the future to accomodate messages over time, and attempt to extend the template models appropriately to reflect both the dynamic nature of the individual elements we are trying to fit, but also the evolution and movement which is an integral part of what templates are in the context of military operations.

10.2 Hybrid constraint propagation and search

While we have identified a unified model for recognition which incorporate search and constraint propagation which clearly defines where heuristics will be of benefit in terms of our model of abstraction, the actual selection and identification of the most effective heuristics is far from clear. More experimentation and examination of the process of analyst identification must be undertaken to better understand these issues. In particular, methods that take advantage of spatial heuristics seem to offer a rich area of investigation.

10.3 Other approaches to model pattern recognition

Besides constraint satisfaction, there are several other approaches that could be taken to address the problem of model pattern recognition as we have defined it. In this section we merely comment briefly on these addresses methods and technologies, and where possible give reference to this work. In the future, we are interested in examining each of these, and determining what role they may play in future systems. Quite possibly, some of these technologies can be merged or adapted for use in MOTORS.

Many must be considered in adapting any model of recognition, or in attempting to extend the recognition procedure. The main point that must be recognized is that the system needs to be interactive and explainable to the analyst. We have adopted abstraction as one way of attaining this communication, and other strategies must either accomodate this, or offer alternatives.

10.3.1 RETE algorithm

The entire conception of many pattern/many object matching is what model pattern recognition is all about. The RETE algorithm developed by Forgy [15] is well known in the construction of rule based systems.

10.3.2 Model recognition in vision

Some vision problems are highly related to template matching. The process of identifying objects in a confused world relates to both paradigms. Vision often involves lower level recognition of dot patterns that constitute base parts (such as vertices, edges, etc.) of larger scale objects. In this context, template recognition would appear to be closer to forming objects given the base parts. In vision, this formulation is often based on decision-tree construction based on identified differences among the objects one is interested in distinguishing amonst a picture resolved into many base parts. If we were trying to identify which of a set of templates was present in a set of base data, this type of approach would seem logical. However, we have been concerned with identifying all instances of a single object in a set of data, and as a result the question is slightly different. In the future we will be interested in considering multiple template identification, or possibly most-likely template identification, and in that instance, the problems would be more similar.

If we consider that template matching does involve correspondences of a spatial nature, it is possible that discrimination algorithms that identify base objects from point data might be of assistance in our process, perhaps in offering candidates for consideration based on identification of certain unit deployment patterns. Several difficulties exist, such as the error inherent in locating sightings, so that rather than points, we likely are talking about smudges of some size and shape, and additionally, we are concerned with many other attribtutes in addition to position, such as types, size, activity and so on. At any rate, some extension of a pattern identification algorithm could be useful.

In [58], a method is proposed using a data-parallel algorithm for the identification of dot patterns in large data sets. We are interested in futher investigating this approach to see how data-parallel algorithms might be used to restrict our search space, possibly in advance, or possibly during search.

10.4 Connectionism

Certainly there have been many uses of connectionist algorithms in the field of vision. We realize that these approaches have sometimes proved useful where other methods have failed, and for this reason are interested in investigating the applicability connectionist approaches to heuristic creation, and pattern recognition.

10.4.1 Expert systems

Rule based systems, both forward chaining and backward chaining have been used for similar systems required to provide and explain hypothetical explanations in other domains. The implementation of an inferencing system (such as that provided by CLIPS, for instance) together with an ability to reason about spatial concepts could provide the analyst with the ability to form specific queries particular to a case in question. Conceivably such a system could benefit from the use of MOTORS tools as input to hypothesis and explanation modules.

In many ways, MOTORS is already an expert system without the formal use of rules. Expert knowledge is embedded in the definition of templates, and even further in the creation of abstraction hierarchies which capture a portion of the way in which analysts identify templates. Construction of a user interface based on abstraction as has been discussed in this paper further advances towards the domain of "expert systems". Providing the analyst with the ability to further constraint or control search and the recognition process via adjustable rules could be one way in which the MOTORS toolset could be given the flexibility it needs to function in real situations where templates must change dynamically, and the way in which they must be located evolves.

10.4.2 Statistical methods

Other work [36] being conducted at DREV in the Combat Intelligence Automation Group has focused on statistical identification of templates in message data[36]. We will not attempt to outline this work even initially here, but merely point out another alternative point of view for the reader to investigate further.

A method which attempted to "order" possible solutions according to a predefined measure of "goodness" or "fit" would be beneficial, particularly if such a measure could be configured and supplemented by the analyst. The CSP approach to recognition evaluate constraints for each candidate solutions, and considering these constraints as the building blocks of an understandable measure for a solution suggests how to compare solutions in that paradigm. Statistical fitting is capable of functioning in a similar fashion.

10.4.3 Simulated annealing

Consider that a template is like a game in which you try to make several steel balls fall into the holes on a plate which you manipulate. The situation elements are the balls, and the template slots the holes. The basic conception behind simulated annealing is that when one adds energy to a system, you disturb the elements of the fit, essentially allowing some of them to escape their holes or local minimums. While in attempting to solve such a puzzle, a little shaking might be good and force the balls to find a different static state or local minimum, while a lot might move "good" or satisfactory fits. This model is similar to hill climbing approaches that select quick fit solutions and then repair them according to repair heuristics.

The difficulty with modelling our problem in this way is that we are interested in finding all candidate solutions so the analyst may experientially discriminate on the basis of factors not accounted for in a particular model. Single solution or several solution models that don't guarantee all solutions will be found or suggested could suffice, if we could guarantee in some way the degree of match that woudl be found. For instance, that a template instance of a certain degree of certainty as defined in a model would not be missed.

Annealing must deal with many hard issues of selecting attractiveness/magnetism, between slots and situation elements, of determining energy addition, and other issues such as the effect of gravity in the model. It is not clear how these issues relate to the manual identification process currently used, and how solutions arrived at in this fashion could be acceptably justfied to the analyst.

10.5 Parallelization of constraint propagation

We are interested in possible use of data-parallel algorithms in constraint satisfaction. To date we have not investigated this option further. In [49], Nadel points out:

> ...parallel approaches to solving constraint satisfaction problems are currently receiving considerable attetion - see for example Freuder and Quinn (1985) [18], McCall et al. (1985) [43], and Kasif (1986) [31]. The latter paper sounds a note

of restraint by showing that Constraint Satisfaction is in a sense an inherenetly nonparallizeable problem.

A further followup by Kasif of this work may be found in [32].

10.6 Distribution of search

The search for all solutions to a problem results in a search space with many branches or alternative points. Some of our work in this paper has focused on attempting to reduce the number and since of these subtrees, both by thoughtful constraint propagation, and specific search and domain heuristics.

In our abstract approach to interaction, solutions "trickle-in" to the analyst for consideration, and while we attempt to order these in some kind of "best-first" fashion as much as possible, it may be the case that getting as many solutions as possible as quickly as possible will allow the analyst a better basis for comparison of solutions. A distributed system which at some point in the search committed separate processes to locating solutions in separate search subtrees might provide us with more solutions, faster, and as such should be considered for incorporation into our model. In addition to abstraction, problem partitioning suggests alternatives on which to base a strategy of distributing portions of the search, and then re-integrating these portions later.

10.7 Interaction issues

In designing an interactive system, the way in which the analyst perceives and utilizes the system is of prime importance. For such a system to be useful it must provide understandable answers, be capable of explaining inferences or suggestions, and be able to communicate with the analyst in familiar terms.

10.7.1 Interaction and search

While we have accounted for interacting search and analyst desires through a model using abstraction, the depth-first search strategy selected is by no means the only solution to this problem. While there are representation and comparison issues to consider when adopting some other strategy that may suggest solutions from various hierarchical representations in parallel, it is not clear that the solution or solutions the analyst is searching for will definitely be found most efficiently in a depth-first interaction.

10.7.2 Representation change

We have presented a model of interaction based on interaction in Chapter 7. One major component of this work was the conception of representing part of the process of template recognition by explicitly identifying the factors that indicate increasing confidence in a particular partially matched template instance. In particular, the abstraction hierarchy of a particular template is created in such a way that increasining specificity of a representation maps to the amount of confidence attributed to a solution for that representation. The difference sets between levels (the explicity representation of the change between levels) is used both as part of the reduction of candidate process, and as a way in which to order possible partial solutions.

We are only in the early stages of examining the issue of representation change in abstraction hierarchies, and other work in this area needs to be examined more closely. In particular, the work of Holte [27, 25, 26] appears to be of interest.

10.7.3 Modelling the analyst process

The model of template recognition presented in this paper models a specific part of the analyst's function in data fusion. Our previous work in [68] addressed a broader question about representing the inferences an analyst makes so as to capture information relating message sightings with conclusions based on military doctrine or other conclusions, and explaining analyst expectations in terms of other messages and previously determined conclusions. In this way a model of the reasoning process was explicity built and could be used to explain the current beliefs about enemy positions.

Each work addresses a part of the analyst fusion task in the context of different models. Templates are discussed independently of other user hypotheses, and Expectations are discussed only briefly alluding to the important role Templates might have when considered as a set of Expectations about enemy position and activity.

A more complete model would further add a temporal aspect giving Messages, Conclusions, Expectations, Hypotheses, and Templates a third dimension. All of these concepts decay over time as their information content becomes less indicative of the real situation, and additional domain knowledge can help build a current model based on factors such as terrain, knowledge about enemy element mobility, and additional information describing possible or proabable enemy goals or plans of movement.

Ideally, a model should exist in which all of these concepts can be discussed with common reference. This work and our earlier work lead towards this model, however, a complete model is still a long term goal.

The Intelligence Fusion systems under development in Canada (TIF - Tactical Information Fusion prototype), and in Australia (TMIPS - Tactical Military Information Processing System) each model only some aspects of the entire domain, and each fail to model temporality adequately.

10.7.4 Distribution of sightings and consequences

We are interested in identifying a "typical" distribution of sightings in terms of locations and types in order to better evaluate particular location schemes in terms of these distributions. It has been pointed out that clustering of candidate elements could dramaticaly affect any scheme for template recognition, even in the presence of broad spatial heuristics. Our current implementation uses only random locational distributions, and only rudimentary distributions for attributes.

Further, the identification of actual ranges of locations for sightings is important also. Specifically, a sighting is not at a point in space generally, but rather in an area of some type. We need to further restrict these spaces realistically in order to attempt to match templates with "real" data. The following section deals with our attempt to discuss this problem with an actual military intelligence analyst.

10.8 Identifying locational range of reported sightings

Introduction

A message of an enemy unit or equipment sighting usually contains information such as the unit size and the location of the sighting. Given other information such as the role, activity, terrain, etc., the intelligence officer (IO) can mentally approximate the region (containing this location) of which that unit currently spans. This region forms some connected set from which the IO can approximate its boundary or contour. When other messages of a unit's assets are received the IO is allowed to construct a better approximation of the region and its boundary.

The goal of the work outlined in this section is to develop a procedure that will approximate the regions introduced above, allowing a numerical data (and graphical) representation of the region itself and its boundary. This would allow us to give a better detail of the IO's belief about the distribution of sightings of a certain type and size.

The work in this section has been drawn from discussions with an intelligence officer at Valcartier, Quebec, and from the following reference materials: [51, 50, 52, 2, 56, 55].

PROCEDURE

After consulting with an IO, it was decided to use an averaging method as the procedure to approximate the regions and their boundaries. No other concrete options were formulated.

It was also decided, that the procedure will apply to military units from the company level and greater for defensive operations, as well as those from the platoon level and higher for offensive operations.

In order to obtain the sets defining these regions, information on the elements (subunits and assets) of a unit and general spatial constraints between them are used (i.e. templates). The elements are determined by the role and level of the unit whereas general spatial constraints are defined by the unit's activity, the terrain, the weather, etc. .

For a given unit and situation, a (finite) set of templates that apply to the situation are considered. For each template, a solution mean is determined by some weighted verage of the solutions of the constraints.

For example, if the template for a given unit and situation has two subunits A and B, where the constraint between these two is that they must be separated by a distance of x to y units in distance, then a mean, MA, B(x,y) of the solutions is calculated.

If a template has more units and constraints in the template, then means of the solutions of the constraints are to be calculated. A problem does arise when calculating means of solutions, being that this mean may not be a solution itself.

After a solution mean of a template has been determined, one can define the boundary of the region around the units' subunits. So now a region for a template has been defined.

Taking all template applicable to the unit and situation a mean of the regions is calculated. It is assumed that the regions of the templates are 'centred' at one location. There are many options available to calculate this mean. Examples are listed below.

- Calculate the mean of area that they span preserving their geometric form.
- Calculate the mean of the orientation of many of the same geometric form, all equivalent in area.
- Calculate the mean of lengths of a regions along some axis, say of symmetry.

EXAMPLES

MARCH

When a regiment is in march, the battalions, companies and platoons will march along one itinerary (march along one road in columnar fashion). An example with data is given in the following section.

MOTOR RIFLE REGIMENT IN MARCH This example drawn from [52].

Units are listed in order of march. Here the covering is simply a path segment along a road or itinerary. Lengths of the unit coverings and distances between units are also given.

There are also 2 MRP's that act as guards. There is one on each wing of of the regiment, each MRP is 1-3 km away from the main body.

There is no information on the covering of the guards (MRP) on the wings other than being 1-3 km away from the main body on either side.

UNIT	MINIMUM LENGTH (km)	MAXIMUM LENGTH (km)
Advance Guard MRB (+)	15	25
	20	30
RHQ and Combat Support Elements	1.8	5.2
	3	5
Tank Battalion	3	5
	3	5
MRB	1.6	3.2
	3	5
MRB (-)	1.5	2.8
	3	5
Rear Services	3.5	6.4
	1	3
MRP	1	3
	(There are also 2MRP's on either side of the MRB)	

Figure 10.1: MRR in March template description.

SOLUTION For the situation of MRR in March, there is only one applicable template. Assuming that the lengths of the regiment's subunits and lengths between the subunits are uniformly distributed between the minimum length and maximum length, we may take the mean length as an average of these two lengths.

The region defined by this solution is a path following one itinerary about 70 km in length. In the area between the regimental HQ and the tank battalion a short line segment may cross the path to represent the 2 MRP guards.

OFFENSIVE OPERATIONS

ARTILLERY BATTALION An artillery battalion consists of 3 artillery companies situated at about 0.5 to 1.5 km from each other. The 3 companies form a triangle usually oriented towards the enemy.

The region formed by the taking the mean of the constraints is an equilateral triangle with each side 1km in length. The triangle would point towards the enemy along an axis of symmetry.

UNIT	AVERAGE LENGTH (km)
Advance Guard MRB (+)	20
	15
RHQ and Combat Support Elements	3.5
	4
Tank Battalion	4
	4
MRB	2.4
	4
MRB (-)	2.2
	4
Rear Services	5
	2
MRP	2km on either side of the MRB

Figure 10.2: Solution MRR in March.

Chapter 11

MOTORS Implementation

The MOTORS[1] toolset consists of search algorithms, constraint propagation algorithms, and support for the integration of search and propagation. The search algorithms provide for the construction of one, all or possibly some solutions to constraint satisfaction problems. The implemented search and propagation algorithms are described in general terms in Chapter 3. The search algorithms each apply varying degrees of "look-ahead" or remaining problem reduction between selection of variable domain values for instantiation, and are configured so as to allow extension of their propagation before or during search. The constraint propagation algorithms are designed as stand-alone components which take as input a constraint specification or graph, and reduce it accordingly. Essentially, a domain is constructed on top of the search and propagation facilities.

Specific domain implemented are for the N-Queens problem[2], and for the Model Pattern Recognition (MPR) Problem[3]. The Model Pattern Recognition domain implementation will be discussed in the later part of this appendix.

The purpose of this appendix is to conduct a general overview of the tools provided to date in the toolset, and to outline the code for future application, modification, and enhancement.

All source code is commented, and provision has been made for debugging flags for tracing search and propagation output. For further details, please refer to the code and make note of the control flags.

11.1 Tour of the code

The code is separated into the following modules:

[1]Unless otherwise noted, all implementation was done by Steven Woods.

[2]Original code provided by Qiang Yang and Phillip Fong, University of Waterloo.

[3]Code designed and written by Steven Woods, DREV.

- *bt.lisp* The generic backtracking algorithm. This algorithm accepts as input a search-state, the consistency function, and control flags indicating which combinations of search and propagation should be used. Options supported are currently Forward-Checking, Dynamic-Rearrangemnt, BackJumping, application of a pre-defined arc-consistency (or partial arc-consistency) algorithm during search, or application of a pre-defined spatial limiting heuristic during search.

  Currently the arc consistency algorithms available for use must be explicity hardcoded into the algorithm, however, in the future we will wish to make this either an option based on the passed arc-consis parameter, or pass the function to apply itself. The current version has the AC-3 algorithm used as an example.

- *bm.lisp* The BackMarking algorithm is implemented recursively as opposed to the iterative approach of *bm.lisp*. This algorithm accepts as input the variables list, and the consistency function. Currently application of in-search constraint propagation or spatial heuristics are not supported.

  In the future it would be desirable to investigate further how to incorporate in-search propagation without disrupting the maintenance of the various constraint results tables maintained in BackMarking.

- *ct.lisp* The constraint propagation algorithm AC-3[4], supported by the revise algorithm as described by Mackworth. This algorithm is stand-alone, and accepts as input a constraint graph defined in terms of a variable and domain list, a function indicating the presence of arcs in the graph, and a function for determining the consistency of two assignments.

  Some experimentation was done with non-binary constraint graphs for consideration with MPR description, however, this proved problematic with respect to integrating with intelligent backtracking algorithms such as BackJumping. Future research on this integration can be undertaken, and reference made to other work in n-ary graph propagation, such as the algorithm GAC-4 [62, 61, 63]. It should be noted that in [59] it is shown that any CSP with n-ary constraints can be converted into another equivalent binary CSP.

- *Devel/ac4.lisp* The constraint propagation algorithm described in [46][5]. This algorithm may replace AC-3. It also accepts as input the constraint graph and outputs the simplified graph. Please note that the published algorithm is in error, and correction *has* been noted in this file according to personal communication with Hany.Tolba@loria.fr. Please note that this function is not currently operational and needs to be repaired. Notes are provided with the source code as to specific difficulties and testing.

[4]Please note that this source code for AC-3 was provided by Qiang Yang, and Phillip Fong of the University of Waterloo

[5]Implemented by Denis McGonigal under the supervision of Steven Woods.

- *Devel/acq.lisp* The partial arc-consistency algorithms[6]. These algorithms may also replace AC-3 and function in the same way, simplifying an input constraint graph. In conjunction with standard backtracking these may be utilized to form hybrid search algorithms which perform only some arc-consistency simplification of constraint graphs between instantiations. In conjunction with some heuristics determining how much constraint propagation to apply during search it is hoped that improved performance can be obtained.

  In the future, additional research will be done on determining how to best apply these propagation algorithms, and how to select which algorithm to apply. Please note that these functions have been defined separately from MOTORS and are not integrated at this time, however the functions have been tested independently.

- *queens.lisp* The N-queens implementation. Useful for testing and construction of simple, explainable examples, the queens domain has been widely used as an example of constraint propagation. In this source is also provided the alternate Confused-N-queens problem. This file defined the calls available to solve the queens or confused problems, including the calls to initial propagation, and to search. Control flags cover domain selection, search mode selection, application of node consistency, arc consistency, and optiononal flags to control search such as Forward Checking, etc. Further options include in-advance sorting of variable domains (not useful in symettric queens problems, except after constraint propagation may have affected domain sizes), and as well output options such as output to file, debugging, or printing search information at checkpoints.

  This and any other domain files must define the consistency function, and a function which indicates the presence or absence of domain constraint arcs.

  Future extension should unify the calls from the domains under a single initiation procedure, and relegate this file to domain definition only.

- *mpr-simple.lisp* The Model Pattern Recognition domain is implemented in two files. This is the first, which contains the domain function as does the queens.lisp file. Consistency functions are defined in this file as described later in this appendix. Initial calls to search and propagation modules are provided here as described for queens, as well as all constrol flags over selection of instance options such as distribution control over domain element type in random problem instances, and selection of random-key for reproduction of earlier random experiments runs.

- *mpr-setup.lisp* The second of the MPR domain definition files, this file contains all possible situation instances, and template definitions. These instances and templates are identified by name for use in the mpr-simple.lisp function calls as desired. Essentially

[6]Implemented by Denis McGonigal under the supervision of Steven Woods.

we maintain a complete list of all created test cases to facillitate later reproduction of results. In addition, this file maintains the distribution control files for the random creation of attributes in random situations. The random situation creation code resides in this file also.

- *utility.lisp* The general utility file contains functions not specific to either domain. Generally support functions such as global variable instantiation, and output. I/O functions reside here such as maintenance of random state files, creation of random situation output for later display via gnuplot, and output of test results. Note that the gnuplot interface functions as follows: when a new random situation sample is created, that sample configuration is stored in a set of files with unique file names including the random key used, and can be displayed via gnuplot creation of a postscript file and displayed via ghostview or laser printing.

- *load.lisp* The initial load file simply loads all required source code. If you load this file after starting lisp, the system loads.

- *compile.lisp* Loading this file will recompile the entire system.

- *prt-it* Shell script to dump all lisp source code to a printer.

- *COPYRIGHT-NOTICE* File containing version, ownership and authorship information.

- *EXAMPLES* File containing examples of running MOTORS from start to finish of a lisp session. This example file does not demonstrate all MOTORS functions, only a sample.

- *Batch* This directory contains scripts and output from Batch runs of MOTORS. The shell scripts to initialize the lisp process and the lisp input control files are included, and may be used as example of batch runnning the system.

- *Interest* This directory contains extend.notes, a working document detailing the work of Yang in implementing abstraction on top of constraint propagation. This work is not currently part of MOTORS.

- *Random* This directory contains random state descriptors from Lucid Common Lisp which are written out and read in by MOTORS when a unique state is generated. These allow reproduction of particular runs for further investigation. Each filename includes the key identifier needed by MOTORS to run that case.

- *Situation* This directory containts situation descriptions as they are created, including the size and random key used. From these descriptions, the situation can be drawn or shown using the shell scripts draw and show. Future integration of saved situation objects could speed up testing by eliminating the need to repeat construction of identical situations. This is currently not supported.

11.2 Model Pattern Recognition Implementation

Refer to mpr-simple.lisp and mpr-setup.lisp.

The model pattern recognition system is implemented by considering template models as constraint graphs. The following is an example of a template.

```
'( "test1-m"
  (
   (ts1 ARI  * * * Co)
   (ts2 ARM  * * * Co)
   (ts3 ARI  * * * Co)
   (ts5 (ARI AIHQ) * * * Co)
   )
  (
   (sep (ts1 ts2) (4 8))
   (left-of   (ts1 ts2))
   (sep (ts2 ts3) (4 8))
   (right-of  (ts3 ts2))
   (sep (ts2 ts5) (6 10))
   (behind-of  (ts5 ts2))
   (echelon   (ts2 ts5) (4))
   (same-attr-orient (ts1 ts3))
   )
  ( (ts1 20) (ts2 20) (ts3 20) (ts5 20) )
  )
```

This description corresponds to the csp described earlier in Chapter 7, on page 104. The first element "test1-m" is the template name. The second element is a list of node definitions prefixed by node name, including constraints on the *Type*, *Size*, *Activity*, *Orientation*, and *Absolute Location* of the template elements.

- *Type* Indicates which type of unit may be identifed as this template slot. This may be either a list of types or a single type. Possible types in the current implementation are: ARM (Armoured), AIHQ (Armoured Infantry HeadQuarters), ARI (Armoured Infantry), ART (Armoured Tank), or * (unknown).

- *Size* Indicates the possible size of a unit. This may be either a list of sizes, or a single size. Possible sizes in the current implementation are: Grp (Group), Co (Company), Bn (Battallion), PL (Platoon), or * (unknown).

- *Activity* Indicates the possible activity of a unit. This may be either a list of activities, or a single activity. Possible activities in the current implementation are: Adv (Advance), Rec (Reconnaissance), Dep (Deployed), Ret (Retreat), oir * (unknown).

- *Orientation* Indicates the possible orientation of a unit. This may be either a list of orientations, or a single orientation. Possible orientations in the current implementation are: N, NE, E, SE, S, SW, W, NW, or * (unknown).

- *Absolute Position* Indicates the known range of a template slot in absolute terms (if known). Typically this value would be used to either limit a search-in-progress for a template, or to radically restrict a search based upon incontrovertible evidence received. This takes the form of a list consisting of minimum x coordinate, minimum y coordinate, maximum x coordinate, and maximum y coordinate. Generally this field will use a value of *, indicating the known range of this template slot is unknown.

The third element of our template definition is a list of inter-slot constraints. Currently supported constraints are Sep (Separation), Left-of, Right-of, Behind-of, Ahead-of, Echelon, same-attr-orient, same-attr-type, same-attr-size, and same-attr-activity. All these constraints are binary between two and only two slots. In addition, a 3-ary constraint has been used for experimentation purposes: Medial. Please note that medial is not robust, and current search and propagation does not support constraints with airity greater than 2. Each of these arc constraints are described below.

- *Separation* Two slots are constrained by the minimum and maxium distance between them. The form of this slot is a list consisting of the term "sep", a list of the two slots concerned, and a list of the minimum distance they may be apart and the maximum. The distance used is the absolute distance between the given coordinates of two situation elements. Later in this appendix the form of situation elements will be described, however, in the current implementation, each situation element has an absolute coordinate position, not a range or area. In future extensions it will be necessary to give a situation element a ranging area as opposed to a coordinate position.

- *Left-of, Right-of* These constraints are directional. A "destination" slot is constrained by a "source" slot to be either left of, or right of, the source. Left-of is defined by first defining a direction vector by taking the orientation of the source as a vector commencing at infinity and running through the situation object in the direction of the orientation. Any object to the right of the direction vector is "Right-of" the source object, and any object to the left of the direction vector is "Left-of" the source object. This method is only an approximation, and while effective, might not fully capture the intended meaning of left or right. If the source slot instance has no known orientation, these constraints should not apply, and may be thought of as being universally satisfied. Note that Left-of and Right-of are *not* necessarily complimentary since each are defined in terms of the orientation vector of the source, if one exists.

- *Ahead-of, Behind-of* These constraints are directional also. We define a balance line through the "source" slot by rotating the direction vector 90 degrees. Any object to the side of the balance line in the direction of the orientation is "Ahead-of" the source, and any object to the side of the balance line in the opposite direction of the source orientation is "Behind-of" the source object. If the source slot instance has no known orientation, these constraints should not apply, and may be thought of as being universally satisfied. Note that Ahead-of and Behind-of are *not* necessarily complimentary since each are defined in terms of the orientation vector of the source, if one exists.

- *Echelon* Once again, the echelon constraint is a directed one. The source slot and its orientation are used to limit the location of the destination slot. We once again create the direction vector from infinity through the source slot in the direction of the source slot's orientation. We now imagine a corridor created by making this direction vector a plane of some width. For the echelon constraint to hold, the destination object must be within the plane created from the source object location and orientation. If a source slot instance has no orientation, the echelon constraint should not apply, and may be thought of as being satisfied universally. Note that echelon is not necessarily in both directions from some instance A to B and then from B to A, since each direction is determined by the orientation of the source, if it has one.

- *same-attr-orient* Indicates that each of the two affected slot instances must possess an orientation value determined to be "equal". This equality is currently defined as identity. Note that this constraint needs to be extended to accomodate lists of possible values as accomodates by the similar node constraint.

- *same-attr-type* Indicates that each of the two affected slot instances must possess a type value determined to be "equal". This equality is currently defined as identity. Note that this constraint needs to be extended to accomodate lists of possible values as accomodates by the similar node constraint.

- *same-attr-size* Indicates that each of the two affected slot instances must possess a size value determined to be "equal". This equality is currently defined as identity. Note that this constraint needs to be extended to accomodate lists of possible values as accomodates by the similar node constraint.

- *same-attr-activity* Indicates that each of the two affected slot instances must possess an activity value determined to be "equal". This equality is currently defined as identity. Note that this constraint needs to be extended to accomodate lists of possible values as accomodates by the similar node constraint.

The final element of the template definition is an experimental heuristic currently usually nil, known as the SCH (spatial cohesion heuristic) list. This element defines our attempt to capture the conception of "spatial locality" or cohesion in the construction of possible template instances matching this template. Currently, if the SCH flag is set to true during backtracking search, then when any slot with a value in the template SCH list is instantiated, the remaining variable domains are each reduced according to the range limit accompanying the slot identifier. Essentially we are forcing propagation of a spatial constraint similar to separation after each listed instantiation. This heuristic is under experimentation and should generally not be used. It may be very useful in cases where there are many situation elements, and a template is under constrained. It would appear that only certain slots, perhaps locationally critical ones, should be used as SCH limiters.

We have now explained how to define a particular template. The constraints provided must be viewed as examples of the types of constraints typical to this problem, as well as other spatial problems. Other constraints may be added by examining the way these samples are implemented in mpr-simple.lisp, function test-constraint-2.

Situation elements are our candidates from which we must identify a template instance. For the purposes of our experiments we define a particular instance (or instances) in a predefined set of situation elements, and then clutter the situation with additional randomly created situation elements. The following is an example of an instance of the template described above.

```
'( "test-1b"
     ( sit3-1  (70 30)  ARI   Adv ne co)
     ( sit3-2  (74 26)  ARM   Adv ne co)
     ( sit3-3  (79 21)  ARI   Adv ne co)
     ( sit3-4  (69 21)  ART   Adv ne co)
     ( sit3-5  (67 19)  AIHQ  Adv ne co) )
```

This is a partial situation composed of only the elements of the template described. In Chapter 7, Figure 7.5 on page 107 this list of situation elements is graphed as "Middle Abstract Solution", and in Figure 7.4, these elements are embedded in an additional 300 situation elements. The format of each situation element is a list composes of the sighting element identifier, the absolute sighting position consisting of an x and y coordinate, the type of the sighting if known, the activity of the sighted unit if known, the orientation of the sighted unit if known, and the size of the sighted unit if known.

Randomly created situation elements providing noise for our test cases are distributed according to space, and as well according to attribute frequency assignment. The following is a characterization of the current ability to adjust the frequency of each attribute assignment.

```
( "ddist"
  ( (ARM 1) (AIHQ 1) (ARI 1) (ART 1) (* 1) )
  ( (Grp 1) (Co 1) (Bn 1) (Pl 1) (* 1) )
  ( (Adv 1) (Rec 1) (Dep 1) (Ret 1) (* 1) )
  ( (n 1) (ne 1) (e 1) (se 1) (s 1) (sw 1) (w 1) (nw 1) (* 0) ))
```

Here, each of the element attributes Type, Size, Activity, Orientation, or * (unknown) can have configurable frequency when generated as noise. For instance, in this above example distribution “ddist”, each element has equal probability of selection in Type, Size and Activity, while all possible Orientations are equally probable except * (unknown) which is never selected. Our method essentially creates a list for each attribute composes of N elements of each attribute value where N is the value corresponding to the value. An element is selected at random from this created list and assigned to the situation element. For instance, consider a distribution for Type that looked like the following.

```
  ( (ARM 2) (AIHQ 1) (ARI 3) (ART 3) (* 0) )
```

We can then imagine a list of 8 elements composed from this distribution as

```
 ( ARM ARM AIHQ ARI ARI ART ART ART )
```

where there is a 2/8 chance of selecting ARM, a 1/8 chance of selecting AIHQ, a 3/8 chance of selecting ARI, a 3/8 chance of selecting ART, and no chance at all of selection * (unknown).

In the current implementation, selection of “noise” situation element positions is purely random, however, further investigation of different distributions should be investigated and the corresponding affect on search identified. The domain needs to be characterized for arriving message distribution both spatially and by type in order to better simulate reality in the identification process.

11.3 Running Motors

11.3.1 Input

We have discussed briefly the source code portions responsible for running the MOTORS domain problems, the process of template creation, and the process of establishing instances of the templates amidst noise. We show below the source code used in the calling procedure for MPR.

```
(defun mpr (&key
    (situation-id       "test-1")
    (sit-noise                 0)
    (template-id       "test1-w")
    (dom (list 'mpr situation-id template-id))
    (search-mode            "bt")
    (node-consis               t)
    (node-force-all          nil)
    (arc-consis              nil)
    (sch-call                nil)
```

```
(rand-dist          "ddist")
(forward-checking       nil)
(dynamic-rearrangement  nil)
(advance-sort           nil)
(one-solution-only      nil)
(cpu-sec-limit          600)
(ck-pt-interval         150)
(debug                  nil)
(debug-csp              nil)
(random-ident      'default)
(output-file            nil)
)
```

Situation-id identifies the situation instance we have created and wish to use, sit-noise the number of additional elements to generate as noise, template-id identifies the template to search for, dom is an optional domain name, search-mode one of bt (backtracking), bj (backjumping), or bm (backmarking), node-consis indicates whether or not to perform node-consistency checking prior to search (generally true), node-force-all is a indicator that allows for all node constraints to be checked if you desire (never true), arc-consis identifes when to perform the arc-consis algorithm implemented - either before search, during search, or both (this will be extended greatly in the future), sch-call indicates whether or not to apply the spatial cohesion heuristic, rand-dist determines which random situation element distribution to use, forward-checking and dynamic rearrangement are backtrack options as described earlier in the paper, advance-sort indicates whether or not to sort the domain variables according to some heuristic such as domain size, one-solution-only indicates whether or not to find all solutions, cpu-sec-limit stops the search process when it is met, ck-pt-interval prints statistics on at every expiration of the interval, debug indicates that we wish to debug the search process and see the search space, debug-csp indicates that we wish to debug the constraint application process, random-ident indicates which random seed we wish to use (default is the standard case, nil creates a new one, or some identifier indicates a predefined seed in the Random directory), and output-file indicates a file name to write output to if desired.

11.3.2 Output

Sample output from MOTORS looks something like the following example of a default test run.

```
***** OPTIONS for CSP ( (MPR test-1 test1-w) ) *****

 Situation id         :   test-1
 Situation noise add  :   0
```

```
 Situation Size       :   12
 Template id          :   test1-w
 Template Size        :   5
 Distribution id      :   ddist
 Distributions    :
((ARM 1) (AIHQ 1) (ARI 1) (ART 1) (* 1))
((GRP 1) (CO 1) (BN 1) (PL 1) (* 1))
((ADV 1) (REC 1) (DEP 1) (RET 1) (* 1))
((N 1) (NE 1) (E 1) (SE 1) (S 1) (SW 1) (W 1) (NW 1) (* 0))
 Num of constraints   :   11

 Search Mode          : bt
 Node Consis Mode     : T
 Node Force all       : NIL
 Arc  Consis Mode     : NIL
 Forward Checking     : NIL
 Dynamic Rearrangement: NIL
 Advance sort option  : NIL
 Backjump option      : NIL
 One Solution Only    : NIL
 CPU Seconds limit    : 600
 CK Point interval    : 150
 Debug                : NIL
 Debug CSP            : NIL

 Random Ident         : default
 Spatial Cohes Prune  : NIL

Domain size ( INITIAL ) statistics ...
Max[(TS1 12)]  Min[(TS1 12)]  Avg[12.0]

Domain size ( NODE ) statistics ...
Max[(TS1 6)]  Min[(TS2 2)]  Avg[3.6]

***** BACKTRACK Information *****

 Exit Location    : NULL-PARENT-STATE
 No. Solutions    : 1
 Node Consis. cks  : 132
 Ttl constr chks  : 36
 BT nodes created : 10
```

```
 Nodes visited    : 34

Advance Processing CPU Seconds: 0.02
Search             CPU Seconds: 0.06
Total              CPU Seconds: 0.08

COMPLETE
```

After a description of the options selected, the output is composed of several parts:

- *Problem Characterization* Domain sizes are evaluated in the initial problem formulation and then again after node and pre-search arc constraint propagation. The improvement can be seen in the average (Avg) measure which indicates the average size of the variable domains before search commenced. Maximum and minimum reflect similar bounds for the same information amongts all variables.
- *Exit Location* This indicates where search exited. It can be in error, from a checkpoint, or from a null-parent-state indicating the search space was completely searched.
- *No. Solutions* As one would expect, the number of template instances found in the situation.
- *Node Consis cks* The number of constraints applied to reduced the variable domains based on node constraints defined. In an implemented system these constraints would be applied in an ongoing fashion, and would not need to be applied at the beginning of a search.
- *Ttl constr chks* The number of inter-slot constraints applied during recognition, including propagation in advance and during search.
- *BT nodes created* The number of backtracking positions during search.
- *Nodes visisted* The size of the search space explored.
- *Advance Processing* The time in CPU seconds spent processing Node constraints or Arc constraints before search commenced.
- *Search* The time in CPU seconds spent in search, including in-search constraint propagation.
- *Total* The sum of Advance Processing time and Search time.

Additionally, further details are available if desired. These are obtained through the function "(show-solution)" which basically repeats the display procedure, but with more detail. The additional detail includes:

- *Matches-attribute* A breakdown for each attribute of the number of node propagation constraints.
- *Consistent-nulls* A counter of the number of times an assignment was made which required no constraint checks at all as a result of the variable lacking applicable constraints.
- *Unique restricts* A counter of the number of times an attempt was made to check a constraint between identical situation elements. This situation arises when template slots have identical node characteristics.
- *Spatical Cohesion* Identification of the cost in terms of additional constraint checks and savings in terms of removed domain values incurred by applying the spatial cohesion heuristic.
- *Constraint Breakdown* A breakdown for each inter-slot constraint of how many types it was checked during the entire propagation and search process.

Two additional output functions are available demonstrating some of the system flexibility and future use for explanation purposes. First, during node consistency checking, a record is kept of failing nodes and why they failed to qualify for assignment to a template slot. This structure can be queried by "(explain)" which shows all reasoning behind removal, or "(explain-node-fail slot instance)" which will tell why a particular instance was not a candidate for a particular slot. Second, the function "(show-cons-fail)" details each application of a failed arc constraint indicating which situation elements failed the constraint.

Acknowledgement

This document has been prepared by Steven Woods, Defence Scientist, DREV. Sections 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and Section 10.8 were written by Denis McGonigal while a summer student under the supervision of Steven Woods.

Thanks to Guy Vezina of DREV for discussion about parallelization and terrain analysis, to Stephane Lapointe and Luc Lamontagne also of DREV for interesting comments on my work, to Graham Williams of CSIRO Australia for discussing the relationship of this work with user expectations, and to Qiang Yang, University of Waterloo for providing some initial lisp code, and for interesting discussion about abstraction and problem decomposition.

Bibliography

[1] John Anataki. Analysis of the division icac for an artificial intelligence / expert system current enemy situation prototype development. Contractor W7701-8-3535/01-XSK, Defence Research Establishment Valcartier, 1989. Unclassified.

[2] Canadian Army. Programme d'enseignement personnnel sur le regiment de chars d'assaut fantasien. training manual PEP 498-1 to 4, Centre d'instruction au combat bureau tactique, May 1992. Not classified.

[3] Martin C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[4] A. Dechter and R. Dechter. Removing redundancies in constraint networks. In *Proceedings of the 6th AAAI*, pages 105–109, 1987.

[5] R. Dechter. From local to global consistency. In *Eighth Canadian Conference on Artificial Intelligence*, 1990.

[6] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34, 1987.

[7] R. Dechter and J. Pearl. Tree clustering for constriant networks. *Artificial Intelligence*, 38:353–366, 1989.

[8] Rina Dechter. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[9] Rina Dechter. From local to global consistency. *Artificial Intelligence*, 55:87–107, 1992.

[10] Rina Dechter and Itay Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. *unknown*, pages 271–277, 1988?

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

[12] L. Des Groseilliers, J.-C. Pare, and D. Soucy. Tactical information fusion prototype: Enemy order of battle database. Technical Report 3066/91, Defence Research Establishment Valcartier, June 1991. Unclassified.

[13] Army Doctrine and Tactics Board. Combat intelligence, first draft. Technical Report FMC 2910-CFP 315(2), National Defence, Canada, Mobile Command Headquarters, St-Hubert, Quebec, August 1988. Unclassified.

[14] Canadian Forces. Tactics troop aide memoire. Technical Report QL 6B 8901 YO 9001 PH III 8902, Canadian Forces School of Military Engineering, 1988.

[15] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, September 1982.

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

[17] E.C. Freuder. A sufficient condition of backtrack-free search. *Journal of the ACM*, 29(1):23–32, 1982.

[18] E.C. Freuder and M.J. Quinn. Parallelism in an algorithm that takes advantage of stable sets of variables to solve constraint satisfaction problems. Technical Report 85-21, Dept. of Computer Science, University of New Hampshire, Durham, NH, 1985.

[19] Eugene Freuder and Alan Mackworth. Introduction to the special volume on constraint-based reasoning. *Artificial Intelligence*, 58:1–2, 1992.

[20] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the 9th AAAI*, volume 2, pages 227–233, 1991.

[21] M. Gauvin, L. Des Groseilliers, and D. Roy. Tactical information fusion prototype: Research project definition. Memorandum 3068/91, Defense Research Establishment Valcartier, Valcartier, Quebec, May 1991. Unclassified.

[22] United States Government. Intelligence analysis. Field Manual FM34-3, Headquarters, Department of the Army, Washington, DC, January 1986. Unclassified.

[23] Qi Guan and Gerhard Friedrich. Extending constraint satisfaction problem solving in structural design. In F. Belli and F.J. Radermacher, editors, *Lecture Notes in Computer Science*, volume 604 of *Lecture Notes in Computer Science*, pages 341–350. Springer-Verlag, 1992.

[24] R.M. Haralick and G.L Elliott. Increasing tree-search efficiency for constraint satisfaction problmes. *Artificial Intelligence*, 14:263–313, 1980.

[25] Robert Holte, Robert Zimmer, and Alan MacDonald. When does changing representation improve problem-solving performance ? In *Proceedings of the Workshop on Change or Representation and Problem Reformulation*, April 1992. NASA Ames Technical Report FIA-92-06.

[26] Robert Holte, Robert Zimmer, and Alan MacDonald. A study of the representation-dependency of abstraction techniques. In *Proceedings of the 1993 Workshop on Knowledge Compilation and Speedup Learning*, 1993.

[27] Robert C. Holte. Automatic change of representation. presentation slides, parts 1 and 2, December 1992.

[28] Paul Hubbe and Eugene Freuder. An efficient cross product representation of the constraint satisfaction problem search space. In *Proceedings of the 10th AAAI*, pages 421–427, 1992.

[29] Acquired Intelligence Inc. Knowledge acquisition for combat intelligence. Contract W7701-8-5202/01-XSK, Defense Research Establishment Valcartier, May 1990. Unclassified.

[30] DMR Group Inc. Knowledge representation for tactical information fusino. Contract W7701-1-0659/01-XSK, Defense Research Establishment Valcartier, May 1992. Unclassified.

[31] Simon Kasif. On the parallel complexity of some constraint satisfaction problems. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 349–353, 1986.

[32] Simon Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.

[33] Craig Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the 9th AAAI*, volume 2, pages 686–691, 1991.

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

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

[36] Stephane Lapointe. Statistical identification of templates in message data. Combat Intelligence Automation Group, Defence Research Establishment Valcartier, 1992.

[37] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[38] A.K. Mackworth. Consistency in networks of relations. In Webber and Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Morgan Kaufmann Publishers Inc., 1981.

[39] A.K. Mackworth. Constraint satisfaction. In S.C. Shaprio, editor, *Encylopedia of Artificial Intelligence*, volume 1, pages 205–211. John Wiley and Sons, 1987.

[40] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 125:65–74, 1985.

[41] Alan Mackworth. The logic of constraint satisfaction. *Artificial Intelligence*, 58:3–20, December 1992.

[42] Alan Mackworth and Eugene Freuder. The complexity of constraint satisfaction revistited. *Artificial Intelligence*, 59:57–62, 1993.

[43] J.T. McCall, J.G. Tront, F.G. Gray, and R.M. Haralick. Parallel computer architectures and problem solving strategies for the consistent labeling problem. *IEEE Transactions on Computers*, C-24(11):973–980, 1985.

[44] J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomporphisms. *Information Sciences*, 19:229–250, 1979.

[45] 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.

[46] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[47] U Montanari. Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

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

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

[50] National Defence, Base des forces Candiennes de Gagetown, Oromocto, NB Canada. *Programme D'enseignement Personnel Sur Le Regiment De Chars D'Assaut Fantasien*, May 1992.

[51] National Defence, Base des forces Candiennes de Gagetown, Oromocto, NB Canada. *Programme D'enseignement Personnel Sur Le Regiment De Fusiliers Motorises Fantasien*, May 1992.

[52] National Defence, Base des forces Candiennes de Gagetown, Oromocto, NB Canada. *Programme D'enseignement Personnel Sur Les Operations Offensives Fantasiennes*, May 1992.

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

[54] Peter Norvig. *Paradigms of Artificial Intelligence Programming, Case Studies in Common Lisp*. Morgan Kaufmann, 1992.

[55] Ministry of Defence. Tactics of the soviet ground forces. Technical Report B-GL-050-INT/FT-135, Ministry of Defence, Chief of the General Staff. Amendment No.1, Army Code 71031.

[56] Ministry of Defence. Notes on the soviet ground forces. Technical Report B-GL-050-INT/PT-121, Ministry of Defence, Chief of General Staff, 1979. Amendment No. 4.

[57] W.J. Park and B. Cote. An artificial intelligence representatio of the fantasian ground forces. Technical Report 2947/88, Defence Research Establishment Valcartier, October 1988. Uclassified.

[58] I. Rigoutsos and R. Hummel. Massively parallel model matching. *Computer Magazine*, 25(2):33–42, February 1992.

[59] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint-satisfaction problems. Technical Report ACT-AI-22-89, MCC Corporation, Austin, Texas, 1989.

[60] R. Sosic and J. Gu. A polynomial time algorithm for the n-queens problem. *SIGART*, 1(3), 1990.

[61] H. Tolba, F. Charpillet, and J.P. Haton. Representing and propagating constraints in temporal reasoning. *AI Communications : The European Journal on Artificial Intelligence*, 4(4):145–151, December 1991.

[62] Hany Tolba, Francois Charpillet, and Jean-Paul Haton. Representing and propagating constraints in temporal reasoning. In *Proceedings of the International Conference on Tools for Artificial Intelligence*, pages 181–185, November 1991.

[63] Hany Tolba, Francois Charpillet, and Jean-Paul Haton. A new temporal model combining qualitative and quantitative information. In *Research and Development for Expert Systems (series)*. December 1992.

[64] P Van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[65] P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, December 1992.

[66] Guy Vezina. Virtual battlefield applications. work in progress at DREV, June 1993.

[67] D. Waltz. Understanding line drawings of scenes with shadows. In P.H. WInston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw Hill, Cambridge, Massachusets, 1975.

[68] 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.

[69] Steven Woods. A method of interactive recognition of spatially defined model deployment templates using abstraction. to appear, DND workshop on advanced technologies in Knowledge Based Systems and Robotics, August 1993.

[70] 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.

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

[72] Qiang Yang and Phillip Fong. Csp sample source code. personal communication, 1993.

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