Search Tree Visualization Tool

H. Simonis
A. Aggoun

COSYTEC SA
4, rue Jean Rostand
F-91893 Orsay Cedex

1. Abstract

This paper specifies a visual tool to display and manipulate search procedures and constraints in finite domain constraint programming. It is based on requirements collected from different users of CHIP and on experiments performed inside the DISCIPL project. This specification will form the basis for the visual search tree tool [D97] to be developed by COSYTEC for its CHIP [C96] constraint programming system.

2. Overview

In this section we describe some background information. The reader should be familiar with finite domain constraint programming, in particular using the global constraints in CHIP[C96][AB93][BC94][SC95].

2.1 Finite domain program structure

Typical finite domain programs consist of three, well separated parts:

- Variable Definition
- Constraint Statement
- Search Procedure

The visualization tool should help to understand all 3 parts of a programs, with special emphasis on the visualization of the search procedure[D97].

2.2 Expected Users

The tool should be useable at different levels of expertise[F97], from a novice constraint programmer trying to understand how constraints work, to the expert programmer developing and debugging large applications, but also by the tool developer to understand existing and to help find new propagation mechanisms. These different users have different requirements. The beginner will want to debug the program with a minimum of effort, while the expert will want to concentrate on parts of his program only without displaying too much irrelevant information. The tool developer may be interested in the internal process of constraint propagation, which will normally not interest the application programmer.

2.3 Method of Debugging

The visualization tool should work on-line. This means that the constraint program is run to generate information about the search tree which is then displayed in a graphical form inside the constraint tool. The user can then interact with this representation of the search tree and re-instantiate the state of the search process at different points in the search tree.
In order to generate the search tree, the constraint program must terminate. Four termination criteria can be used (the search stops when any of the criteria is met).

- Number of solutions found
- Number of failures in the search
- Number of nodes explored in the search tree
- Execution time limit exceeded

Finite domain constraint programs must use a search procedure to fix variables to values, since the constraint solver is not complete. In principle any type of program can be used to perform this search, in practice we find that value and variable choice predicates are most common. In value choice, the selection of the next variable is deterministic, the non-deterministics search is performed over different values, in variable choice search methods the selection of the variable is non-deterministic, but only one value is used to fix the variable to a value. The tool must work with both types of search methods.
In the following, we will assume a value choice procedure for simplicity.

3. Interface to application program

The interface to the visualization tool should be as simple as possible, while allowing the user to control all aspects of the display.
For a novice user with small example programs it can be possible to extract automatically all variables and all constraints and to display the complete search tree generated by a standard search routine. This behavior should be enabled/disabled with a debugger switch.
For more complex programs, the user must annotate the source code to indicate which variables and which constraints should be displayed. The user must also annotate/rewrite custom search procedures to indicate which choices should be displayed in the search tree.

3.1 Marking Variables

Two methods for marking variables are possible. One approach consists in indicating the variables which should be handled in the visualization tool. The other approach consists in passing all variables that should be handled as arguments to the search tree procedure. In the visualization tool we use the second alternative.

3.2 Marking Constraints

The user should be able to indicate that all constraints should be included, or should be able to individually mark/unmark constraints. This will be done in the form of meta-predicate wrappers around the constraints in the source code.

3.3 Annotating Search Procedures

3.3.1 Built-in search procedure

The easiest case is the annotation of built-in search procedures using the predicate labeling/4[C96]. The labeling routine performs variable selection and value assignment in a value choice scheme, where heuristics and choice functions are given by the user. In order to visualize this search scheme, the call to labeling/4 should be replaced by a visualize_labeling predicate which takes the same arguments and which automates the generation of the search tree.

3.3.2 User defined procedures

For user defined procedures two annotations are required. One is a wrapper around the search part of the program, which indicates when the search starts and when it ends. This predicate takes a list of domain variables as an argument. The other predicate is a wrapper around a non-deterministic part inside the search routine. The user can decide to see each choice individually or combine several choices into one. This gives additional control over the width and depth of the search tree.

3.4 Restricting Displayed Information

The key to success for debugging large-scale application will be the restriction of the displayed information to important parts. The user must be able to control the volume of information both when generating the search tree and inside the visualization tool. At each point, it is possible to reduce or to increase the amount of displayed information under user control.

4. Views

The visualization tool provides a number of views into the search procedure. These views are based on four concepts

State

Information is displayed for one particular state of the search tree, e.g. the domains of all variables at a particular node of the search. Moving from one state to another updates all information.

Path

Information is displayed for a path in the search tree from the root node to another node. This shows the change of constraints and variables over all choices leading to the selected node.

Tree

Information is displayed for all nodes below the current node. It can be shown either as the intersection or as the union of the individual nodes. This concept is used for advanced features described at the end of the report.

Evolution

Information is displayed for all nodes in the search tree explored before the current node. This concept is useful for statistics on failures, calls of predicates, number of propagation steps. Some of the information may only be easily available for the complete search tree, but quite difficult to obtain for each node in the tree without re-running the search procedure.

We now discuss the different views in more detail.

4.1 Tree View

The search process is represented in tree form with each connection from parent to child indication a separate choice. The tree view is used to analyze and to navigate through the search space.

4.1.1 Nodes

The tree consists of different types of nodes, which are detailed below. Nodes are color coded with a user modifiable color scheme. A typical scheme would use colors to show the number of alternatives in each interior node.

4.1.1.1 root

The root node corresponds to the state of the constraint system after the constraint set-up, before the enumeration process. At this point it is possible to see the domains of all variables after the initial constraint propagation.

4.1.1.2 interior nodes

The interior nodes correspond to states in the search tree where not all choice points have been explored and the constraint system has not failed.

4.1.1.3 failure leafs

This type of leaf node is displayed when the constraint system fails. Failure nodes can be shown in two ways:

The more explicit representation shows each failed choice in the search tree. If for example, a indomain call tested 10 different values and each of them failed, then this representation should display 10 failure nodes. This has the advantage that there is a well defined unique failure cause, i.e. one constraint that failed during propagation. The disadvantage is the possibly large number of failure nodes.

The more compact representation shows one node for every decision predicate (like indomain) which failed. One failure node here abstracts possibly many failures, where individual values have been tested and the constraint system failed. The advantage is the more compact form of the search, the disadvantage the difficulty to explain the propagation which led to the failure.

The system must be flexible enough to switch between the two representations, depending on the use of the tool.

4.1.1.4 success leafs

Success leafs are reached when all variables in the problem have been assigned and a solution was found. Note that quite often the last steps of the search procedure are induced by constraint propagation, i.e. the last real choice in the search may be several levels above the success leaf and all nodes below this choice are deterministic, i.e. the variables selected already have been assigned by constraint propagation.

4.1.1.5 failure trees

In order to present the search tree in a more compact form, it is useful to compact failed sub-trees into a single node, a failure tree. This tree is marked with the number of failure leafs it contains. The user can interact with the system to collapse or expand parts of the tree by hand. The system also has options to automatically collapse all failure trees or to expand all nodes.

4.1.1.6 success trees

Similar to failure trees, the system can display several solutions as one success tree node. Again, the system indicates how many failure nodes and how many success nodes it contains. The user can collapse/expand any node to the corresponding failure or success tree.

4.1.2 User Interaction

4.1.2.1 Basic Interaction

The use can zoom to any part of the search tree by either using scrollbars and/or selecting an area in the current display area with the mouse.
The use can navigate through the search tree from node to node by selecting nodes with the mouse.
The use can collapse the sub-tree originating at any node or expand a currently collapsed sub-tree.

4.1.2.2 Selecting nodes

Selecting a node sets the current state of the search process to the state represented by the node. This requires to re-instantiate all variables on the path from root to the selected node to obtain the path information. The system can show information about the selected node, like the level of the node in the search tree and the index of the selected variable. There are commands to show all other nodes which assign the selected variable, or to find isomorphic sub-trees to the sub-tree below the current node.

4.2 Variable View

The different variable views are intended to help understand the impact of the search procedure on the different domain variables.

4.2.1 Update View

This state shows the change of the variables on the different steps on the current path from root to selected node. In x direction, all variables are shown, in y directions the levels of the search tree. Each entry in this table marks the change of the variable with different colors. The following updates are recognized

- variable is set at this level by search procedure
- variable is ground
- min and max is updated
- min is updated
- max is updated
- size is changed, i.e. interior value was removed

4.2.2 State View

This view shows the domains of all variables at a given state. In x direction, values in the domain are shown. Each line corresponds to a variable. The entries in this view show which values are currently in the domain of the variables. They can also show which values were removed in the last assignment step. In a future extension, the view can be used to show all values which are in the domain in all sub-trees and values which are removed in all leaf-nodes below the current node.
For large domains, a textual representation of the domain may be more compact than a graphical one. It must also be possible to restrict the display to a user-defined interval of values.

4.2.3 Spy View

This view allows to follow the change of individual domain variables through the path from root to current node. In x-direction, values are shown, in y-direction the different levels of the path. Changes from one level to the next are displayed in each line, as well as all values currently in the domain.
The user can specify which variables should be spied within the graphical environment.

4.3 Constraint View

The constraint views show either the complete constraint network of the program, or show the evolution of particular constraints within the search process. The views are useful to understand the problem structure and the overall constraint reasoning.

4.3.1 All Constraints

Some of the constraint views can be used for all types of constraints, others are specialized for the global constraints in CHIP. We first discuss the general views.

4.3.1.1 Incidence Matrix

All constraints can be shown as a constraint variable incidence matrix. In x direction, all variables are displayed, in y direction, all constraints are shown. If required, this matrix can be compressed by grouping constraints into lines which count occurrences of variables. This display gives an indication of the impact of constraints on variables, which is also used for the variable selection in certain strategies.

4.3.1.2 Update View

This view shows the acitivity of the constraints in the different levels of the search tree. In x-axis, all constraints are shown, in y-axis, the different levels of the search tree. An entry shows what happens to a constraint at this level in the search tree. Different colors are used to encode whether

- the constraint is woken
- the constraint causes some update (min,max, remove)
- the constraint binds some value
- the constraint is solved

4.3.1.3 State View

This view displays all active constraints at a given point, i.e. all constraints which have not been decided. The representation is derived from the incidence matrix, with variables along the x-axis and constraints on the y-axis.

4.3.1.4 Spy View

For some constraints, in particular those involving many variables, it can be interesting to see the evolution of the constraint in the search path. In x-axis, all the variables in the constraint are shown. On y-axis, the different levels of the search tree are displayed. Entries show which variables are updated either by the selected constraint and in a different color those updated by some assignment or other constraint propagation.
The user can interactively select/deselect constraints to be spied.

4.3.1.5 Constraint Count

This simple view shows a graph with the level of the search tree on the x-axis and the number of active constraints on the y-axis. This display shows the progress of constraint propagation in a very simple form.

4.3.1.6 Other Representation

It may be possible to represent particular constraint networks in graphical form, i.e. as graphs with variables as nodes and hyperlinks for constraints, or as particular interval graphs. These possibilities must be further investigated before fixed requirements can be stated.

4.3.2 Global Constraints

The global constraints [C96][AB93][BC94][SC95] form the centerpiece of the CHIP system. They are very high level constraint abstactions, often working on a large number of variables at the same time. Many complex constraint problems can be easily expressed by a small number of different global constraints. Debugging global constraints has its own problems and possibilities, which are quite different from the debugging of basic constraints.
Many of the global constraints have different, optional parameters, which state additional conditions to be satisfied.
The internal reasoning of the constraint is based on multiple, complex algorithms. It is useful to get information about the effectiveness of these methods for particular problems without understanding all details of the method.

Global constraints have useful visual representations, derived from the intended use of the constraint.
On the implementation level, global constraints are defined via multiple constraints on one variable each. During propagation, these individual constraints are woken multiple times, leading to a large number of invocations of the global constraint.
All variants of global constraint views are state based, i.e. they show the stateof the computation at a given node in the search tree.

4.3.2.1 cumulative

The cumulative constraint[AB93] expresses cumulative resource limits over time. It is used primarily to express resource constraints in scheduling problems, but has many other uses for problems like placement or bin packing.
The basic cumulative constraint view is a profile, which shows the current utilization of the resource. It does not show individual tasks, but only the overall resource consumption.
Another cumulative constraint view shows individual tasks and their impact on resource availability. This display can be used for example to visualize producer/consumer constraints[SC95].

4.3.2.2 diffn

The diffn constraint [BC94] states that a n-dimensional rectangles do not overlap in a n-dimensional space. It is often used to express assignment, scheduling or packing problems. In the simplest case, the diffn constraint is visualized by one or multiple 2D placement views.

4.3.2.3 cycle

The cycle constraint [BC94] finds cycles in directed graphs, represented in a domain form. The cycle constraint is shown as a collection of completed cycles, unclosed lines and completely connected components. A visualization of the initial graph, while interesting for problem analysis, is outside the scope of the debugging tool.

4.3.2.4 precedence

The precedence constraint [C96] is displayed as a collection of cumulative constraints together with a representation of the precedence network. The graphical representation of the network is deduced by the program.

4.3.2.5 among

The among constraint [C96] is shown as a line of variables, where particular values are color coded.

4.3.2.6 sequence

The sequence constraint [C96] is displayed in a form similar to the among constraint.

4.4 Propagation View

The propagation views are used to understand the propagation for one assignment step. This view is for the advance user which tries to understand the interaction of the different constraints with each other.
This display shows all propagation steps resulting from the current assignment. The view displays variables in x direction, and propagation steps in y direction. Each line of the display corresponds to one constraint. In each line, all variables which are used in the constraint are marked. Any variables which are updated are shown in a coding according to their update type. It is possible to restrict the display to show each woken constraint only once in the propagation view and to ignore all constraints which are woken, but do not further restrict variables.

4.5 Failure Display

An important aspect of the search tree analysis is understanding the cause of failure at a certain point. The system must display information about the failed constraint for each failure node. If multiple failures are collapsed, it should show all causes annotated with the frequency of causing failures.

5. System Requirements

5.1 Variables

The system must be able to identify each domain variable. This is achieved by an internal index number stored within each variable.

The domain of variables must be accessible in a compact external representation, for example as a union of intervals to avoid generating lists of all possible values like in the current dom/2.

5.2 Constraints

Constraints must be uniquely identified in the system. It will be necessary to access constraints from the identifier, and vice versa. In CHIP, the identifier is a number stored in the constraint frame. For constraints with multiple internal implementation constraints, only the number of the global constraint will be used.

5.3 Constraint list

A list of all active constraints currently attached to a variable is available as a term of the call pattern and as a list of integer constraint numbers.

5.4 Last Failure Knowledge

When propagation fails, we must pass the cause of the failure back to the Prolog system, so that we can display it in the debugging tool. This functionality is provided with the propagation_event_log/2 predicate, which can also return all woken constraints and all updates caused by these constraints. This event-log is generated on demand inside the constraint solver and can be accessed as a list of events from the Prolog engine.

The log records the type of event, the predicate which is currently executed, the variable which caused the propagation and variables which are modified. It also records a reason for the event. This value returns the propagation method currently being used in the constraint reasoning. For global constraints, this reason indicates which of the internal methods was responsible for the event.

6. Current State

A first version of the visualization tool has been implemented. This tool was used to investigate a number of design decisions, and also serves as a proof of concept for large-scale debugging.
At the moment, the following tools and views are available,most in a restricted form.

- Search tree display
- Variable update
- Variable state
- Variable spy
- Constraint incidence matrix
- Constraint count
- Cumulative global constraint visualization
- Cycle global constraint visualization

The required primitives for failure analysis have been implemented, the propagation event log is available in a textual form.

The visualization tool prototype has been used in preparation of the Standard Models tutorial at the PACT 97 conference in London [S97]. The tool was used to explain the difference between search methods and strategies, in particular partial search methods. Problems studied in this tutorial were based on real-life problems, with special emphasis on realistic problem sizes.

7. Summary

In this report we have described the expected functionionality of a search tree visualization tool for the finite domain constraint solver in CHIP. This tool provides a graphical representation of the search tree explored by the constraint solver coupled with views of the variable and constraint evolution during the search process. A number of extension of the underlying constraint engines were described. A prototype of the tool with restricted functionality was developed and tested on realistic examples.

8. Bibliography

[AB93] A. Aggoun, N. Beldiceanu
Extending CHIP in Order to Solve Complex Scheduling Problems.
Journal of Mathematical and Computer Modelling, Vol. 17, N. 7, pages 57-73
Pergamon Press, 1993.

[BC94] N. Beldiceanu, E. Contejean

Introducing Global Constraints in CHIP
Journal of Mathematical and Computer Modeling, Vol. 20, N. 12, pages 97-123
Pergamon Press, 1993.

[BM95] J. Boye and J. Maluszynski
Directional types for logic programs and the annotation method, Rapport INRIA, N°2471, January 1995, France.

[C96] COSYTEC
CHIP V5 Reference Manual.
COSYTEC SA. Paris.

[Ek] F. Eklund
Declarative Error Diagnosis of GAPLog Programs, Thesis N° 627, University of Linkôping.

[D97] DiSCiPl Group
CP Debugging Tools, June, 1997.

[DEC95] P. Deransart, A. Ed-Dbali, L. Cervoni
Prolog the standard, Reference Manual, 1995.

[F97] Massimo Fabrice
Debugging Needs, DiSCiPl Report, ICON, 1997.

[PBH97] G. Puebla, F. Bueno, M. Hermebegildo
UPM, Draft 30 June 1997, A Proposal for an assertion Language for Debugging of Constraint Logic Programs.

[SC95] H. Simonis, T. Cornelissens
Modelling Producer/Consumer Constraints
Principles and Practice of Constraint Programming CP'95, Cassis, France, Sept. 95,
Lecture Notes in Computer Science 976, pages 449-463.

[S97] H. Simonis
Standard Models
Tutorial Notes, Practical Application of Constraint Technology Conference, London, April 1997

[V94] E. Vetillard
Using Declarations in CLP. Thesis, Marseille, 1994.