Visualization of Global Constraints

A. Aggoun
N. Beldiceanu
E. Bourreau
H. Simonis

COSYTEC SA
4, rue Jean Rostand
91893 Orsay Cedex
France

1 Abstract

In this report we describe visualization tools for global constraints in the CHIP system. These tools help to understand the behavior of different global constraints, the core feature of the CHIP constraint language. The tools follow a general classification scheme for the use of global constraints in global constraint concepts. Each concept captures the use of a constraint for a particular type of problem, which can be visualized in a different way. The different visualizers form a class structure of CHIP++ objects, and can be extended or modified by rewriting some callback predicates.

2 Introduction

Debugging of constraint programs take can quite different forms and requires a wide variety of tools. Depending on the host language, the programmer may use a standard debugger or use trace facilities built into the language. It has been reported that debugging with such tools is often time consuming and not very effective [Fab97]. To overcome this problem, ad hoc graphical output has been generated already for the very first constraint programs. In recent years, graphical tools which simplify this task and which make the debugging process more systematic have been developed, in particular for Eclipse [Mei95], OZ [Sch97] and CHIP [SA97]. We can distinguish the following variants:

- Search tree tools represent the search tree generated in a finite domain program in a graphical form and allow user interaction to query the state of the engine at each point. The OZ explorer also provides interactive search facilities, the user can expand nodes in the search tree under his control and thus solve a problem "by hand" while the system performs the constraint propagation. The CHIP search tree tool generates the search tree in a first phase, and allows interaction by re-creating the state of a particular node by re-instantiating variables to certain values.
- The domains of domain variables can often be graphically displayed. Most often this display takes the form of an incidence matrix variables-values, but some systems also use a textual representation.

- Display of the propagation steps in the constraint solver is less common. In the CHIP search tree tool, each propagation step after a value assignment is shown in a graphical form, which indicates the variables affected and the type of update.
- Failure analysis, an important aspect of understanding constraint search behavior, can also be supported by graphical tools.
- The general form of the constraint network is often interesting. Little work has been reported on the display of a constraint network in graph form. The CHIP search tree tool uses an incidence matrix representation, which is particular useful for the representation of non-binary, global constraints.

In this paper we describe another approach to visualization, the concept based representation of global constraints. Global constraints form the core of the CHIP system, they are high-level abstractions which can be used to express complex conditions between sets of variables in a simple way, and which also contain powerful propagation methods to reduce domains and to check consistency.
For the development of the visualization tools, the following design aims were stated:

- Multiple use: Each tool must be applicable in multiple instances, so that it can be re-used for many different application problems.
- Use in the search tree tool: The tools must work together with the search tree tool. If a node in the search tree tool is selected, the constraint visualizer should display the state of the constraint in this node.
- Use as debugging tool without the search tree tool: At the same time, the tools will be used outside the search tree tool. An API is provided so that users can update the display at particular points in the application.
- Use as part of application framework: The tools must integrate into the application framework developed by COSYTEC. For simple demos, they can be embedded into applications as sub windows which are defined via the panel package. For complex applications, they can be added as development phase extensions of the application framework.
- Explanation facilities: The tools should try to explain which constraint propagation method has caused a change in the display, so that users can understand problems in models more efficiently.
- Visualize concepts, not code: The global constraints are powerful abstractions, but they do not always correspond to the programmer's view of a particular problem. The same constraint can be used to express many different application concepts. The visualization tools should try to show the constraints in the form of these concepts. This means that one global constraint may have more than one visualization.
- Work with Visual CHIP (concept based): The tools use the same constraint concepts that the VISUAL CHIP modeling tool is using. This allows a unified representation of problems at the modeling and the problem solving level.
- User extendable by rewriting some callbacks: The tools can easily be extended by re-writing/extending some basic callbacks. Rather generic tools can be adapted more closely to particular application domains without major rewrites.

3 Global constraint concepts

In this section we briefly review the difference between global constraints and constraint concepts. Global constraints are abstractions which look for a compromise between expressive power and structure. To increase expressive power, the constraint should be as general and flexible as possible, while the structure is needed to allow efficient constraint propagation. In practice, global constraints are typical as general as the methods used inside allow. While this generality increases the usefulness of the constraint and limits the number of different constraints in a system, it also increases the difficulty of learning and effectively using the constraint. For many applications, it is not necessary for example to understand all 25 pages of the cycle documentation to use the constraint.
The constraint concepts see the problem from another perspective. Each typical use of a constraint forms a constraint concept, these concepts can be combined in an application in different ways to solve a problem. Some concepts are simply different interpretations of the same constraint, some other are restrictions of a constraint, for example limiting degrees of freedom in some parameter.
We will now describe the constraint concepts used in the CHIP environment.

3.1 Cumulative

Some of the concepts for cumulative were already discussed in the original paper introducing the constraint [AB93]. At the moment, we use the following concepts:

- cumulative resource,
- disjunctive resource,
- bin packing,
- producer consumer,
- redundant projection

The cumulative resource is the most commonly used concept for cumulative. The items in the constraints are seen as tasks with start time, duration and resource consumption. The x-axis is the time axis, the y-axis is the resource usage. Limits can be placed on the overall end and/or the maximal resource utilization.
The disjunctive resource case is a special case of the cumulative resource with a resource limit of 1. Each task has a resource usage of 1.
The bin packing concept for cumulative sees items as items to be packed into bins. The x-axis denotes the different items, the y-axis the height of the bins. Each item is specified by its bin assignment, a width which is equal to 1 and a height which corresponds to the size of the item.
Producer/Consumer constraints were introduced in [SC95] to describe the behavior of consumable resources. Tasks either start from time 0 and block a resource amount up a time point on the x-axis (producers) or the start from a time point and stretch to the end of the time period (consumers) and consume a given amount of resource.
The redundant projection is used to strengthen a constraint model by projecting tasks of a (2D) placement problem onto one axis.

3.2 Diffn

For the diffn [BC94] constraint, a large number of concepts is known.

- disjunctive tasks,

- machine scheduling,
- assignment,
- placement 2d,
- placement with spare,
- placement 3D,
- placement 3D + assignment

The diffn constraint can be used to express disjunctive scheduling problems or multiple machine scheduling problems. The x-dimension denotes time, the y-dimension machine allocation. All tasks have height 1 to indicate that they use one machine during their execution. Related is the assignment problem, where tasks fixed in time must be allocated to different machines. The diffn constraint can also be used for placement problems. The easiest variant is a two-dimensional placement. More complex are the cases where we can use a spare dimension for relaxation, or where we place objects in a three-dimensional space.

3.3 Cycle

The cycle constraint [BC 94] also is used in many different ways:

- oriented graph,
- non oriented graph,
- geographical tours,
- tours with machine assignment,
- tours with fixed times,
- tours with time windows,
- loading/unloading
- scheduling with product dependent set-up times,

In its basic form, it is used to find cycles in oriented or non-oriented graphs. If the locations and distances are derived from geographical data, we want to build geographical tours. Additional parameters allow the introduction of machine assignment to the problem. In other problems, start times are linked to the nodes and we are interested in tour planning with time windows. A significant special case is the tour planning with fixed time windows, for example in the case of railway/aircraft rotations. Another extension handles capacity of the tours together with load/unloading operations at each node. A completely different model uses the cycle constraint to express set-up times in multi-machine scheduling problems.

3.4 Among

We also use the among constraint [BC94] in a number of different ways:

- single among,
- overlapping sequences,
- extending sequences,
- multiple among

In the basic constraint, the number of occurrences of a value set is limited. This constraint can also be applied to overlapping subsequences or to increasing sequences

from a start. In its final form, multiple among constraints on different value sets are handled together in the multiple among concept.

3.5 Sequence, precedence, inverse

At the moment, there are no detailed concepts for these constraints.

4 Principles of operation

In this section we describe the basic implementation structure of the visualizer tools. If you are not interested in extending/modifying some visualizer tool, you may skip this section.

4.1 Class Structure

The class visualize is obtained by adding the list_entry attributes to the visualize1 class. All abstract classes are shown italic in the table below, they should never be used by the application programmer. You can add new classes either by deriving a new class from a virtual one, or by specializing an existing class.

visualize
- *visualize_ltsrsb*
  - *visualize_gantt*
    - visualize_disjunctive_tasks,
    - visualize_machine_scheduling,
    - visualize_assignment,
    - visualize_placement_2d,
    - visualize_placement_remains
  - *visualize_profile*
    - visualize_cumulative_resource,
    - visualize_disjunctive_resource,
    - visualize_redundant_projection,
    - visualize_bin_packing,
    - visualize_producer_consumer
  - *visualize_domain*
    - visualize_variable
  - visualize_measure
    - *visualize_stack*
      - visualize_local_stack
      - visualize_global_stack
    - visualize_time
- *visualize_srsb*
  - *visualize_drawing*
    - *visualize_graph*
      - visualize_oriented_graph,
      - visualize_non_oriented_graph,
      - visualize_geographical_tours
      - visualize_graph_lines

The classes visualize_ltsrsb and visualize_srsb differ mainly in their window layout. The first has a main drawing area, drawing areas to the left and on top and scrollbars to the right and at the bottom. Most structured displays are derived from this class, the left and top area being used for scales or resource display.

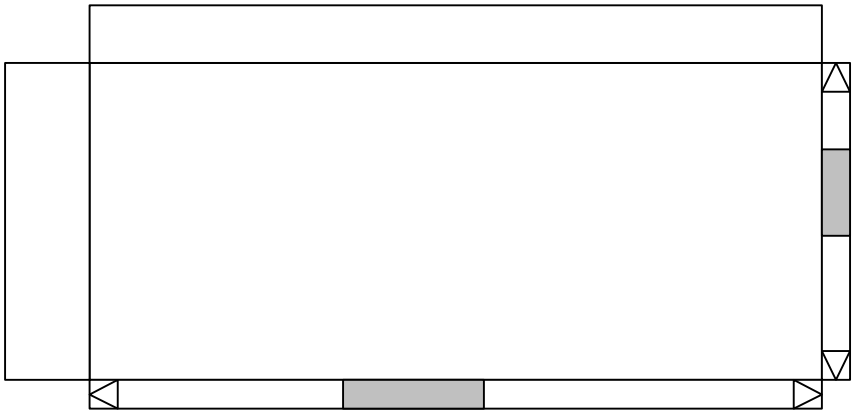

Figure 1 Visualize_ltsrsb window layout

The visualize_srsb class has a main drawing area which is affected by the scrollbars to the right and at the bottom. It is mainly used for drawing concepts of a simpler form.

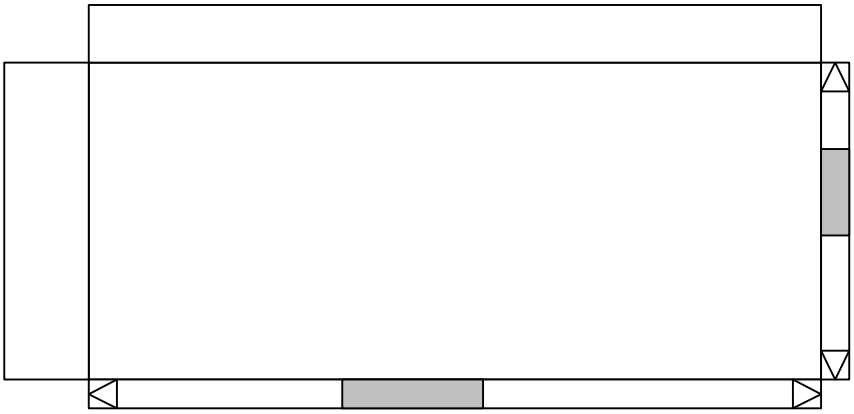

Figure 2 Visualize_srsb window layout

4.2 Callbacks

The callbacks are stored in attributes of the objects in the form of predicate calls pred(X), where pred/1 is the callback name and X is the name of the object affected. Overwriting such a callback with a new routine means that the new code is executed instead of the old one. This means that the new code should emulate all actions of the old code that should be kept. Alternatively, the code can be structured in such a way that the new code calls the callback of the parent class before/after executing any of its own routines.

4.2.1 create_windows

This code creates the object, sets attributes and creates all windows of the visualizer. It does not draw in the new windows.

4.2.2 resize_callback

This code is executed if the back window of the visualizer is resized. This is used to keep the scrollbars, title and resource windows in constant size after a resize.

4.2.3 reset_callback

This code is executed when an existing visualizer is re-created during program execution. It should make sure that default values are set correctly

4.2.4 init_callback (user code)

The init callback is responsible to calculate the displayed area for the constraint. This area should be stored in the x1,y1,x2,y1,ix1,iy1,ix2,iy2 attributes. The code should not attempt to change the vdc of the windows, that is done in another part of the program. Most visualizers want to modify this code.

4.2.5 draw_callback (user code)

The callback is responsible to draw the visualization. The drawing can use all attributes of the visualizer, but most important will be the items attribute which contains the current set of items. These values must not be modified by code.

4.2.6 resource_draw_callback (ltsrsb only)

This code draws the resource scale. The default is a numeric scale dependent on the vdc space of the visualizer. This callback only exists for ltsrsb subclasses.

4.2.7 title_draw_callback (ltsrsb only)

This code draws the title scale. The default is a numeric scale dependent on the vdc space of the visualizer. This callback only exists for ltsrsb subclasses.

4.3 API

This section describes how to use the visualizer library as an application programmer.

4.3.1 Creation

A visualizer object is created by a wrapper around a constraint call inside a CHIP program. The wrapper has the general form

```
visualize(Constraint, Visualizer_type, Attribte_list, Name)
```

The first argument is the call to the constraint, the second argument is the type of visualizer used (an atom), the third is a list of attribute value pairs of the form A<-B where A is an attribute of the visualizer and B is the value used. The last argument is the name of the visualizer object. If the constraint call is executed only once in the application, this name should be an atom. If the constraint call is executed multiple times in a recursive loop, then a new name must be generated each time.
A typical example is the call

```
visualize(cumulative(Start, Dur, Res, unused, unused, Limit, End,
unused), visualize_cumulative_resource, [winw<-500, winh<-400],
cumulative1)
```

which generates the cumulative_resource visualizer cumulative1 working on a cumulative constraint and setting values for the window width and height in the tool.

When the constraint is called, the following actions are performed:

- The program generates the visualizer object of the correct class.
- It then creates all windows required.
- All the parameters are set either to their default value or their value specified in the attribute list.
- The constraint is posted. If the constraint fails immediately, the state of the constraint at this time is shown. If the constraint delays, nothing is drawn. This particular mechanism is used to allow failure analysis at set-up time.
- The internal constraint number is remembered. This number is created internally in the constraint solver and is increased every time a new constraint is posted. Remembering the value allows us to refer to a constraint from the search tree tool.
- The constraint call is remembered by storing the call as an variable attribute in the object. Whenever the domain variables in the call change, the new information can be extracted and displayed. If the program ever backtracks over the creation of the constraint visualizer, no further updates of the display are possible.

Note that the objects are not removed when the query terminates. If the same program is run again, the system recognizes the names of existing visualizers and attaches the constraints to the existing windows. In this way, settings on window positions and sizes are not lost from one query to the next. During a debugging session, the same code can be executed multiple times without destroying and recreating the visualization tools.

4.3.2 Update

To update the tools, two predicates have been defined. Note that the visualizer tools are also updated automatically inside the search tree tool. In that case, none of these predicates need to be called in the application program.

```
Visualize_update
```

This predicate updates all visualizer tools that are active at this moment. This is the normal way to update the display at a given point in the search.

```
Visualize_draw(X)
```

This predicate updates only the visualizer with the name X.

5 Interface to search tree tool

The CHIP search tree tool is described in [SA97]. Its function is to capture and visualize the form of a search tree generated by a CHIP search routine. In the most simple case, a built-in search procedure (labeling) is replaced by another built-in, which automatically calls the correct primitives. In a more complex situation, a user written search routine must be annotated with some special predicates to identify how the search should be displayed.
The interface from the search tree tool to the global constraint visualizers is transparent. If a visualization tool is active during the search, it will be updated

whenever a new node in the search tree is selected. The state of the constraint will be displayed for the node which has been selected.

5.1 New view

A new view has been added in the search tree tool to list all current visualizers. You may change the state of the tool by enabling/disabling updates, or showing or hiding its windows.

5.2 Action on selecting a node

We now describe what is happening in detail when a node is selected. When the callback on the selection of anode is executed, the following steps are performed:
The state of the search in this node is re-instated. The path for the root to the node is scanned to find the variable bindings that must be restored. By constraint propagation, the state of all domain variables and constraints is updated.
The node specific information is displayed in the search tree tool. depending on the selected view, this may be the display of all variables or of all constraints, etc.
All the currently enabled visualizers are redrawn in the current state of the search.
The binding of all variables is undone, we fail until the root of the tree.

6 Use outside search tree tool

The global constraint visualizers can be used outside the search tree tool. In that case, the programmer is responsible of calling the update routine (visualize_update/0) at the right point in the search process.
If called at every point in tree search, i.e. after each indomain, the complete evolution of the constraint is given. This is useful to obtain a first impression of the behavior of the constraint, but a more detailed analysis will be required to understand failures or missing propagation at some particular step in the search.

7 Global constraint visualizers

This section will discuss each visualizer on it own, showing the different parameters which can be set. We also describe the display in the visualizer, and how the user can interact with the system. For each tool, we also show an example from a demo program, which uses different options and which was used to generate the display example.

7.1 Cumulative resource

The tool class is called

`visualize_cumulative_resource.`

7.1.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
guess_color	color to draw guessed parts	seagreen
obli_color	color to draw obligatory part profile	wheat
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
limit_low_color	color to draw minimum limit or end value	yellow
limit_high_color	color to draw maximum limit or end value	red
solid_color	color to draw solid part profile	black

7.1.2 Constraint arguments handled

The tool works with the cumulative constraint and uses arguments

Argument	Name	Usage
1	Start	A list of start times
2	Dur	A list of duration values
3	Res	A list of resource usage
6	Limit	The overall resource limit
7	End	The overall end

7.1.3 Display

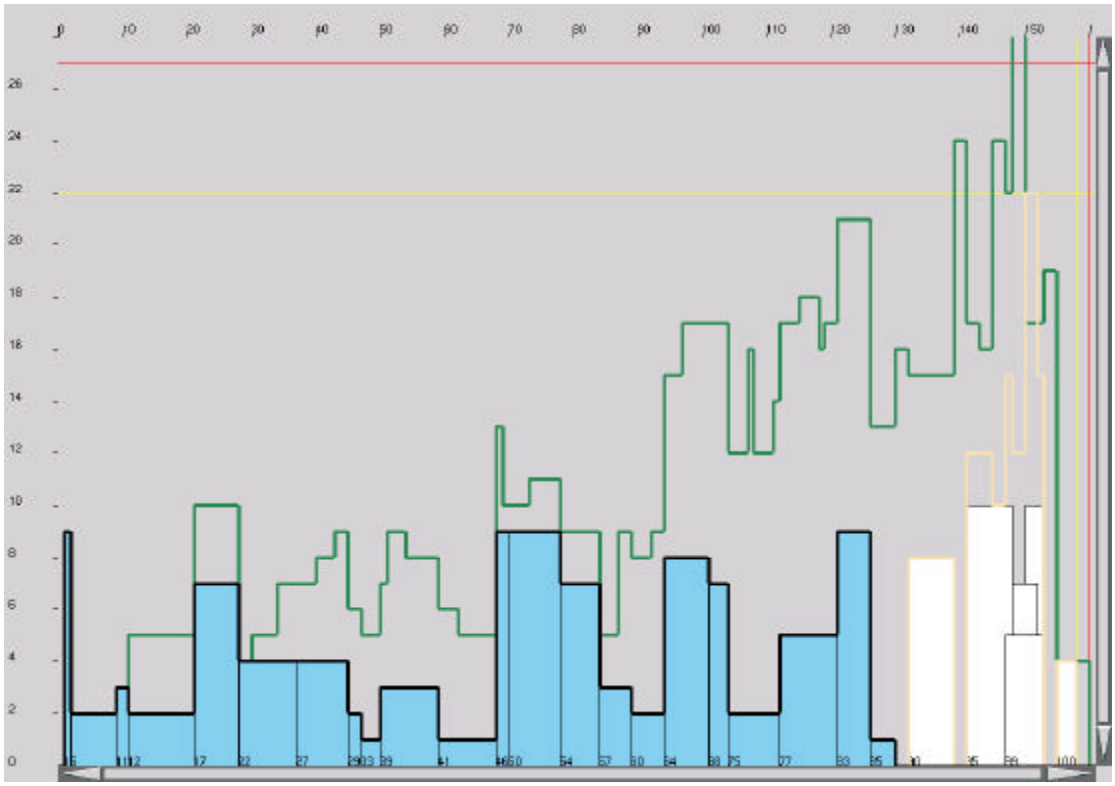

Figure 3: Cumulative Resource

7.1.3.1 Displayed area

The displayed area is limited by the upper domain limits of the Limit and End parameters of cumulative.

7.1.3.2 Fixed parts

A fixed part, drawn in blue, is allocated if Start, Duration, and Resource use are known. The fixed parts are drawn overlapping from the bottom up.

7.1.3.3 Obligatory parts

Obligatory parts are created if the domains of start and duration are sufficiently restricted. They are drawn in white from the bottom up, and they may overlap other parts. An obligatory part indicates that a task is using the resource in this time period, but that the time window and/or the resource use may increase as domains are further restricted.

7.1.3.4 Fixed profile

The fixed profile, drawn in black, is obtained by adding up all fixed parts over the time interval.

7.1.3.5 Obligatory profile

The obligatory profile, drawn in wheat, is obtained by adding up all fixed and obligatory parts over the time interval.

7.1.3.6 Expected profile

The expected profile shows an estimated resource profile by assuming an average resource consumption of all tasks over their possible domain. If the average profile is equal to zero at a time point, then no task can possibly be placed at this time point. The profile drawn in this way is not incremental, further domain reductions will reduce the profile in some part, but increase it in others. The profile may also exceed the limit value in some time period.

7.1.3.7 Limits

The limits in x and y dimension are also displayed in the tool. A yellow line denotes the lower bound, a red line the upper bound of the Limit or End parameters.

7.1.4 Example

The following example generates one visualizer instance for each constraint that is generated. The names of the instances are generated automatically.

```
gen_schedule([],_,_,_,_,_). % no cumulative without tasks
gen_schedule([H|T],Lduration,Lresource,Limit,Max,End) :-
      High :: 1..Limit,
      Endr :: 1..Max,
      Endr #<= End,
      incval(cumul,N),
      sprintf(Name,"cumul_%d",[N]),
      visualize(cumulative([H|T], Lduration, Lresource,
                  unused,     unused,     High,     Endr,     unused),
            visualize_cumulative_resource, [],Name).
```

7.1.5 Demo File

alvarez.pl

7.2 Disjunctive resource

The tool class is called

`visualize_disjunctive_resource.`

7.2.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
guess_color	color to draw guessed parts	seagreen
obli_color	color to draw obligatory part profile	wheat
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
limit_low_color	color to draw minimum limit or end value	yellow
limit_high_color	color to draw maximum limit or end value	red
solid_color	color to draw solid part profile	black

7.2.2 Constraint arguments handled

The tool works with the cumulative constraint and uses arguments

Argument	Name	Usage
1	Start	A list of start times
2	Dur	A list of duration values
3	Res	A list of resource usage
6	Limit	The overall resource limit
7	End	The overall end

In a disjunctive problem, all resource usage values must have value 1 and the resource limit must have limit 1 as well.

7.2.3 Display

as for cumulative resource

7.2.4 Example

```
gen_cum_cliques(unused,_):-
      writeln(unused).
gen_cum_cliques([],_).
gen_cum_cliques([_],_).
gen_cum_cliques([LO,LD|R],End) :-
      length(LO,N),
      length(LR,N),
      LR :: 1..1,
      Er :: 1:9000,
      Er #<= End,
      incval(cumul,Clique),
      sprintf(Name,"clique_%d",[Clique]),
      visualize(cumulative(LO,LD,LR,unused,unused,1,Er,unused),
                visualize_disjunctive_resource,[],Name),
      gen_cum_cliques(R,End).
```

7.2.5 Demo File

alvarez.pl

7.3 Bin packing

The tool class is called

`visualize_bin_packing.`

7.3.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
guess_color	color to draw guessed parts	seagreen
obli_color	color to draw obligatory part profile	wheat
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
limit_low_color	color to draw minimum limit or end value	yellow
limit_high_color	color to draw maximum limit or end value	red
solid_color	color to draw solid part profile	black

7.3.2 Constraint arguments handled

The tool works with the cumulative constraint and uses arguments

Argument	Name	Usage
1	Start	A list of bin assignments
2	Dur	A list of duration values
3	Res	A list of item sizes
6	Limit	The overall bin size
7	End	The overall number of bins

For a bin packing problem, the list of durations must all have the value 1.

7.3.3 Display

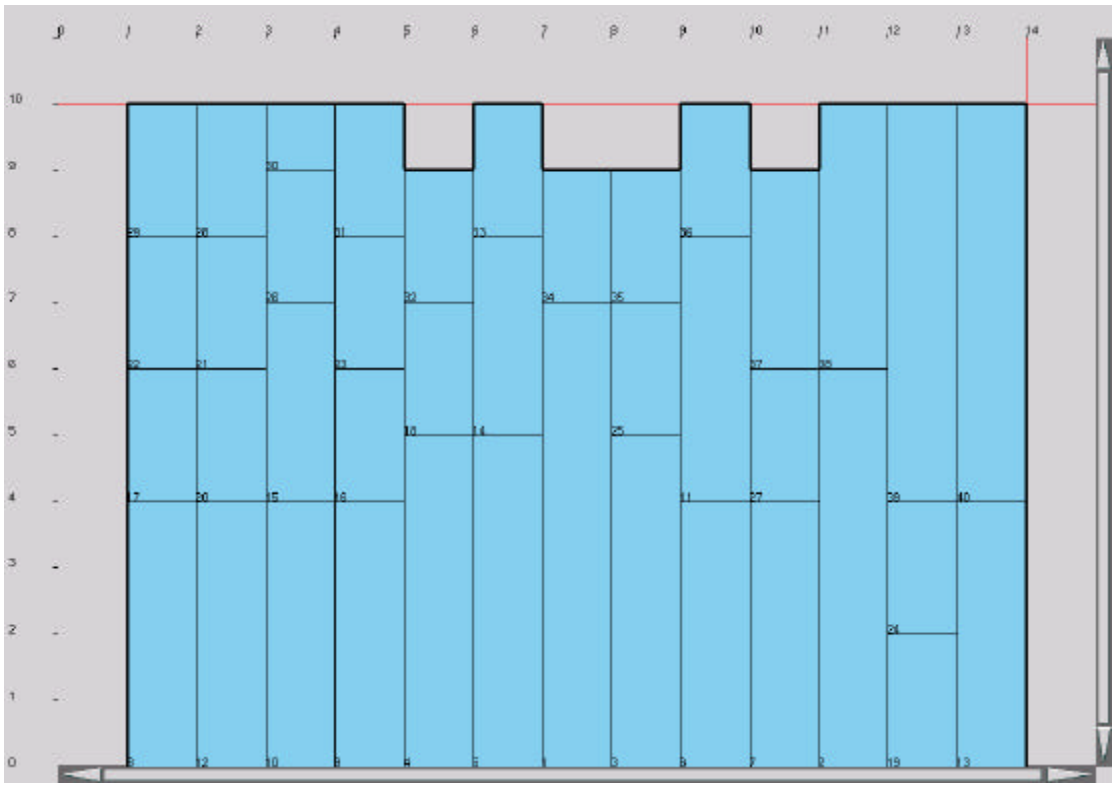

Figure 4: Bin Packing

The main difference to the cumulative resource visualizer is that the items are drawn in a non-overlapping fashion. This is possible as in the bin_packing concept all items have width (x-dimension) of one. Items placed in one bin do not interact with items in another bin. Items in the same bin can be stacked to reach the bin limit.
Another difference is that in the bin_packing case no obligatory parts are generated, as the width of each item is one. In the bin packing visualizer, only fixed items are shown.

7.3.4 Example

```
bin(Start,Dur,Res,profile(Start1,Dur1,Res1,Limit)):-
      append(Start,Start1,S),
      append(Dur,Dur1,D),
      append(Res,Res1,R),
      L :: 0..Limit,
      incval(cumul,N),
      sprintf(Name,"cumul_%d",[N]),
      visualize(cumulative(S,D,R,unused,unused,L,14,unused),
                visualize_bin_packing,[],Name).
```

7.3.5 Demo File

party.pl

7.4 Redundant projection

The tool class is called

`visualize_redundant_projection.`

7.4.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
guess_color	color to draw guessed parts	seagreen
obli_color	color to draw obligatory part profile	wheat
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
limit_low_color	color to draw minimum limit or end value	yellow
limit_high_color	color to draw maximum limit or end value	red
solid_color	color to draw solid part profile	black

7.4.2 Constraint arguments handled

The tool works with the cumulative constraint and uses arguments

Argument	Name	Usage
1	Start	A list of start times
2	Dur	A list of duration values
3	Res	A list of resource usage
6	Limit	The overall resource limit
7	End	The overall end

7.4.3 Display

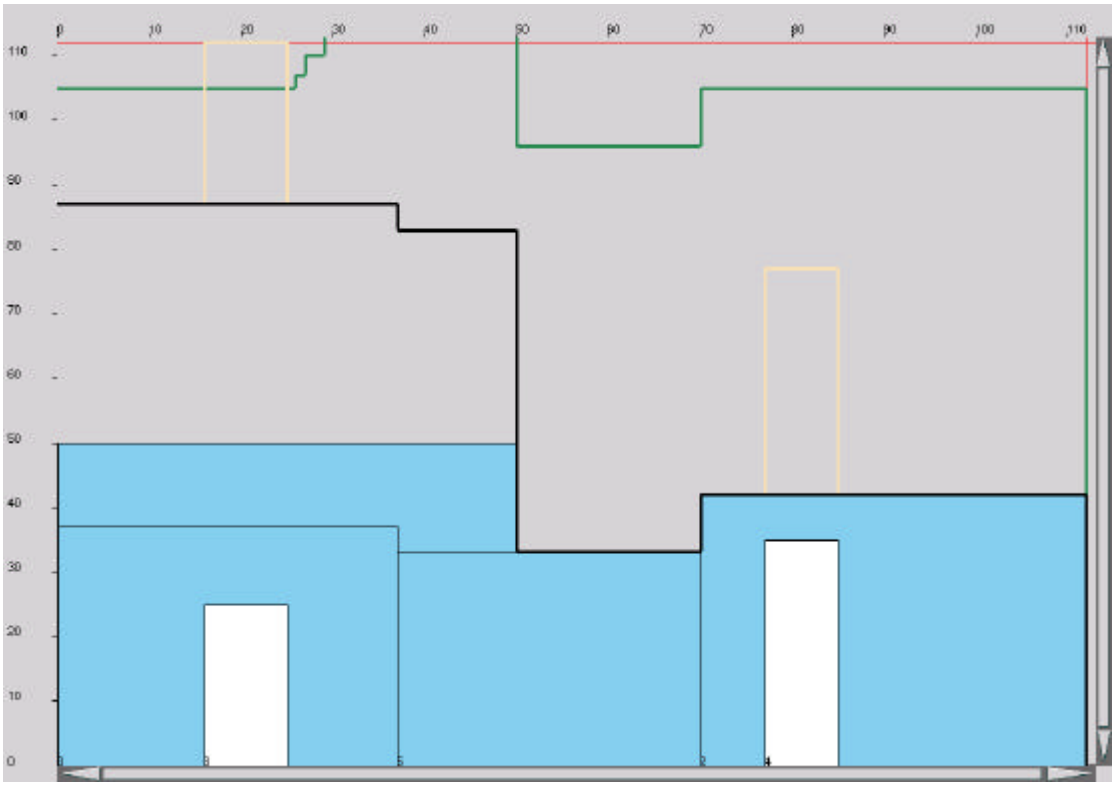

Figure 5: Redundant Projection

The display corresponds to the cumulative constraint concept display.

7.4.4 Example

```
solve(Nr,Data,W,H):-
      create_vars(Data,X,Y,Rectangles,W,H, Vars),

      visualize(cumulative(X,Data,Data,unused,unused,H,W,unused),
            visualize_redundant_projection,
            [winx<-0,winy<-0,winw<-300,winh<-200],cumul1),
      visualize(cumulative(Y,Data,Data,unused,unused,W,H,unused),
            visualize_redundant_projection,
            [winx<-0, winy<-220, winw<-300, winh<-200], cumul2),
      visualize(diffn(Rectangles,unused,unused,[W,H]),
            visualize_placement_2d,
            [winx<-0, winy<-580, winw<-300, winh<-300], diffn),
      visualize(diffn(Rectangles),
            visualize_placement_remains,
            [winx<-320, winy<-580, winw<-300, winh<-300,
            post<-no],diffn2),
      show_solution(Rectangles,W,H),
      search_start(Vars,labeling(Rectangles,1)).
```

7.4.5 Demo File

squarediff_search.pl

7.5 Producer consumer

The tool class is called

`visualize_producer_consumer.`

7.5.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
guess_color	color to draw guessed parts	seagreen
obli_color	color to draw obligatory part profile	wheat
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
limit_low_color	color to draw minimum limit or end value	yellow
limit_high_color	color to draw maximum limit or end value	red
solid_color	color to draw solid part profile	black

7.5.2 Constraint arguments handled

The tool works with the cumulative constraint and uses arguments

Argument	Name	Usage
1	Start	A list of start times
2	Dur	A list of duration values
3	Res	A list of resource usage
4	Ends	A list of end times
6	Limit	The overall resource limit
7	End	The overall end

7.5.3 Display

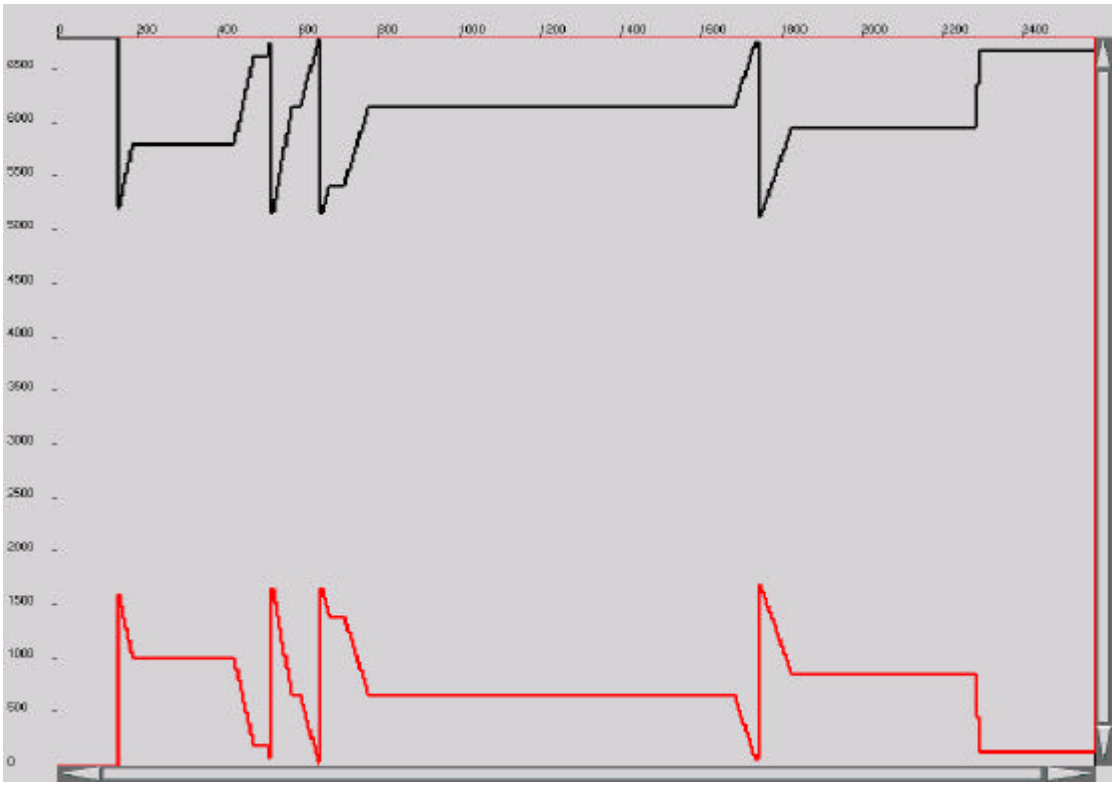

Figure 6: Producer Consumer

The display shows two stock curves. The top-most (black) shows the profile in the cumulative constraint obtained by adding all resource consumptions of all tasks. The bottom one (red) shows the stock level at each point in time. This is the complementary part to the black profile, both added together fill the available space.
To understand the cumulative constraint reasoning, the black profile should be considered, to understand the producer consumer concept, the red profile is more useful.

7.5.4 Example

```
cons_prod([], []).
cons_prod([P|Ps], [N|Ns]):-
      cons_tasks(P, Prod_Tasks, Cons_Tasks),
      append(Prod_Tasks, Cons_Tasks,  Tasks),
      stock(P,Initial_Stock),
      Max1 is Initial_Stock + N*mode@tank_capacity + 1,
      High :: 0..Max1,
      take_end_time(End_T),
      take_task_parameters(Tasks, Starts, Durations, Resources,
                           Ends),
      sprintf(Name,"%d",[P]),
      visualize(cumulative(Starts, Durations, Resources,
            Ends, unused, High, End_T, unused),
            visualize_producer_consumer,[],Name),
      cons_prod(Ps, Ns).
```

7.5.5 Demo File

filter.pl

7.6 Assignment

The tool class is called

`visualize_assignment.`

7.6.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10

7.6.2 Constraint arguments handled

The tool works with the diffn constraint and uses the following arguments of the constraint:

Argument	Name	Usage
1	Rect	A list of rectangles [X,Y, Width, Height]

For an assignment problem, the start times X and duration Width of the rectangles must be fixed, the height of each rectangle Height must be one.

7.6.3 Display

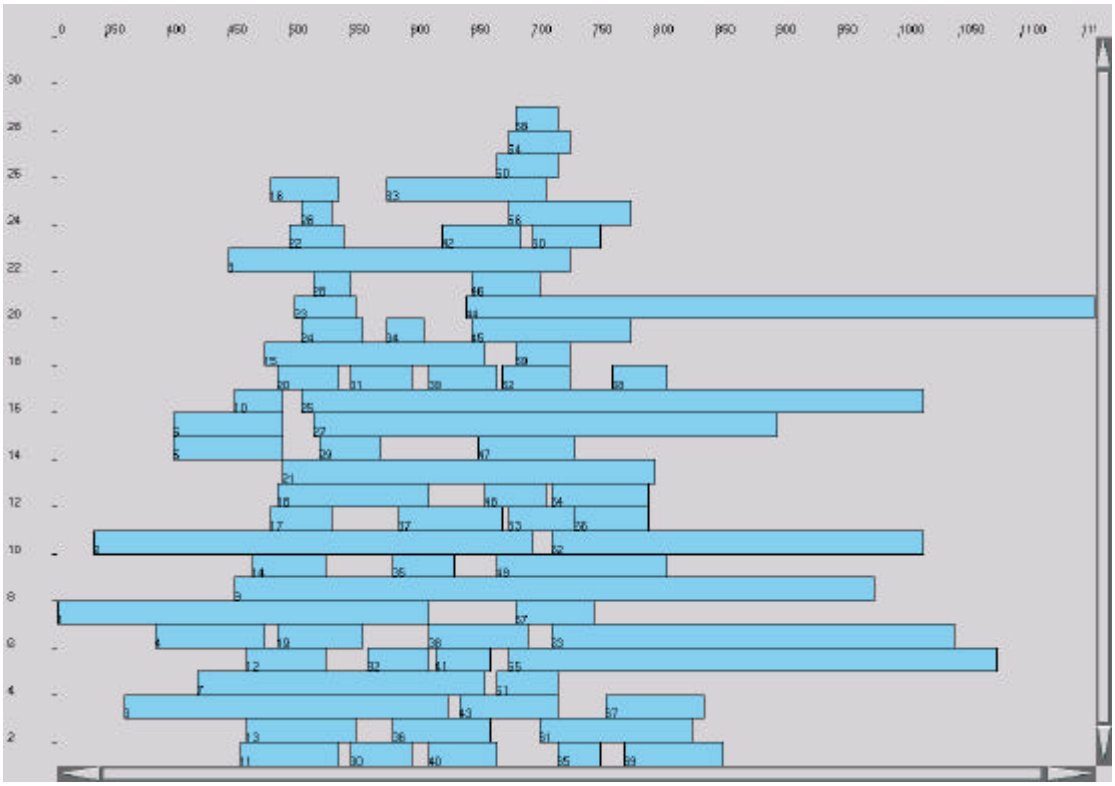

Figure 7: Assignment

7.6.3.1 Displayed Area

The displayed are is obtained by the minimal and maximal values in the domains of the rectangle arguments.

7.6.3.2 Fixed Area

A fixed rectangle (in blue) is drawn if a task is placed on a machine.

7.6.3.3 Obligatory Parts

There are no obligatory parts in this display, as all items have height 1 and can only move in the y-dimension.

7.6.4 Example

```
run:-
      findall(park(A,B,C,D),park(A,B,C,D),L),
      create_objects(L,Obj),
      create_diffn(Obj,Rect),
      visualize(diffn(Rect),visualize_assignment,[],diffn1),
      extract_cost(Obj,0,Sum),
      extract_terms(Obj,Terms),
      sort(Terms,Terms1,2),
      reverse(Terms1,Terms2),
      min_max((labeling_with_preference(Terms2),
               visualize_update,
               grid,
               show_diffn(Obj,Sum)),Sum,0,10000,5,20),
```

```
      grid,
      show_diffn(Obj,Sum).
```

7.6.5 Demo File

stand.pl

7.7 Placement 2D

The tool class is called

`visualize_placement_2d.`

7.7.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10

7.7.2 Constraint arguments handled

The tool works with the diffn constraint and uses the following arguments of the constraint:

Argument	Name	Usage
1	Rect	A list of rectangles

7.7.3 Display

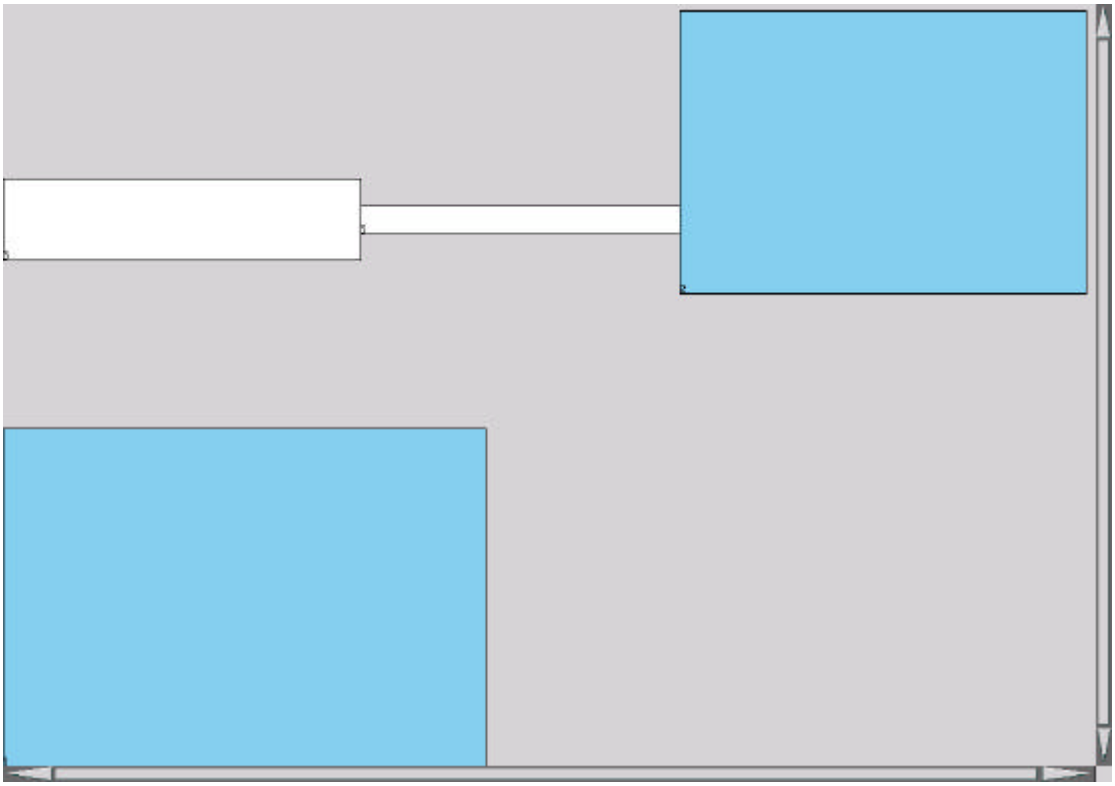

Figure 8: Placement 2D

7.7.3.1 Displayed area

The displayed area is bounded by the minimal and the maximal values of all domains.

7.7.3.2 Fixed parts

A fixed part is drawn if X, Y, Width and Height of the rectangle are known.

7.7.3.3 Obligatory parts

An obligatory part is drawn if an area is known to used by a rectangle regardless of its final placement.

7.7.3.4 Possible placement

The possible placement area shows the area in which the rectangle could currently be placed. This does not take holes in the domain into account.

7.7.4 Example

```
solve(Nr,Data,W,H):-
      create_vars(Data,X,Y,Rectangles,W,H, Vars),
      visualize(cumulative(X,Data,Data,unused,unused,H,W,unused),
            visualize_redundant_projection,
            [winx<-0,winy<-0,winw<-300,winh<-200],cumul1),
      visualize(cumulative(Y,Data,Data,unused,unused,W,H,unused),
            visualize_redundant_projection,
            [winx<-0, winy<-220, winw<-300, winh<-200], cumul2),
      visualize(diffn(Rectangles,unused,unused,[W,H]),
            visualize_placement_2d,
            [winx<-0, winy<-580, winw<-300, winh<-300], diffn),
```

```
visualize(diffn(Rectangles),
      visualize_placement_remains,
      [winx<-320, winy<-580, winw<-300, winh<-300,
      post<-no],diffn2),
show_solution(Rectangles,W,H),
search_start(Vars,labeling(Rectangles,1)).
```

7.7.5 Demo File

squarediff_search.pl

7.8 Placement remains

The tool class is called

`visualize_placement_remains.`

7.8.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
ground_color	color to draw ground items	skyblue
obligatory_color	color to draw obligatory parts	white
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
guess_color	color to draw unplaced rectangles	seagreen

7.8.2 Constraint arguments handled

The tool works with the diffn constraint and uses the following arguments of the constraint:

Argument	Name	Usage
1	Rect	A list of rectangles

7.8.3 Display

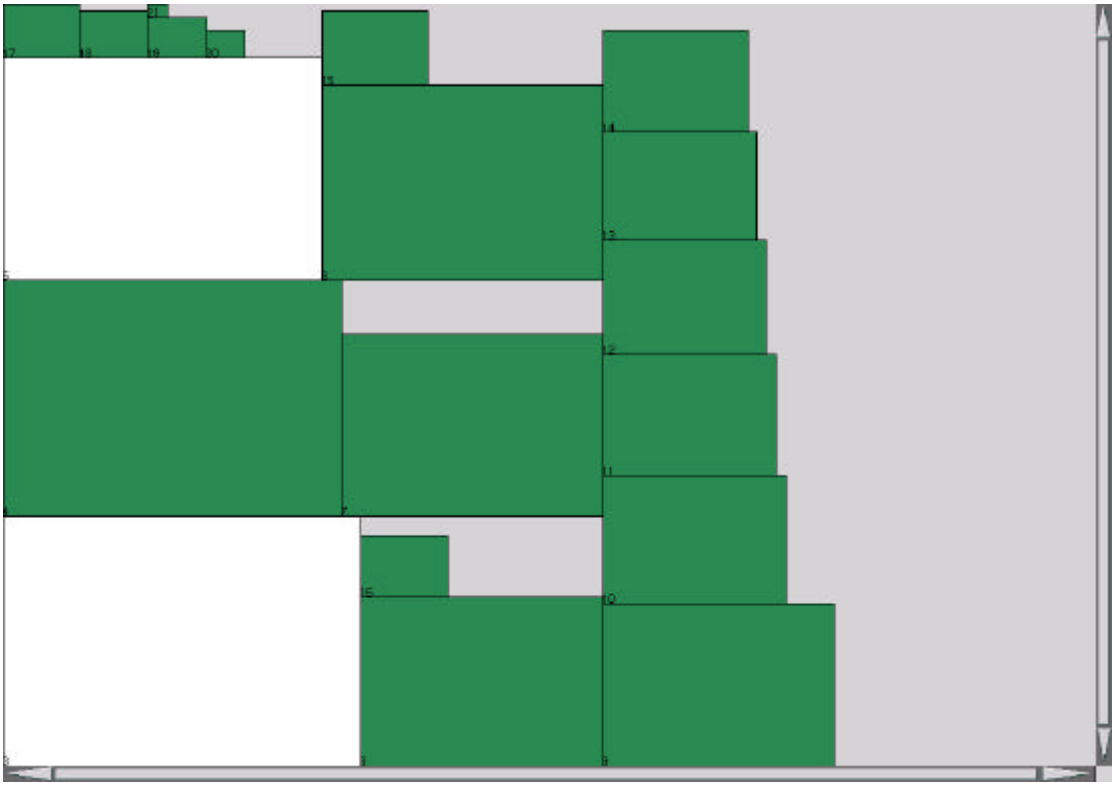

Figure 9: Placement remains

7.8.3.1 Displayed area

The displayed area is bounded by the minimal and the maximal values of all domains.

7.8.3.2 NonFixed parts

A fixed part is drawn if X, Y, Width and Height of the rectangle are known.

7.8.3.3 Obligatory parts

An obligatory part is drawn if an area is known to used by a rectangle regardless of its final placement.

7.8.4 Example

```
solve(Nr,Data,W,H):-
      create_vars(Data,X,Y,Rectangles,W,H, Vars),
      visualize(cumulative(X,Data,Data,unused,unused,H,W,unused),
            visualize_redundant_projection,
            [winx<-0,winy<-0,winw<-300,winh<-200],cumul1),
      visualize(cumulative(Y,Data,Data,unused,unused,W,H,unused),
            visualize_redundant_projection,
            [winx<-0, winy<-220, winw<-300, winh<-200], cumul2),
      visualize(diffn(Rectangles,unused,unused,[W,H]),
            visualize_placement_2d,
            [winx<-0, winy<-580, winw<-300, winh<-300], diffn),
      visualize(diffn(Rectangles),
            visualize_placement_remains,
            [winx<-320, winy<-580, winw<-300, winh<-300,
            post<-no],diffn2),
```

```
      show_solution(Rectangles,W,H),
      search_start(Vars,labeling(Rectangles,1)).
```

7.8.5 Demo File

squarediff_search.pl

7.9 Oriented graph

The tool class is called

`visualize_oriented_graph`.

7.9.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
show_domain	show successors only if domain is smaller than value	1
radius	size of circle for node	30
marker_color	color or marker	blue
marker_style	style of marker	plus
marker_size	size of marker	5
assigned_link_color	color of assigned successor link	red
unassigned_link_color	color of unassigned successor link	blue
node_locations	list of pairs X-Y, specifying node locations	0

7.9.2 Constraint arguments handled

The tool works with the cycle constraint and uses the following arguments of the constraint:

Argument	Name	Usage
2	Succ	A list of successors
3	Weight	A list of weight values

7.9.3 Display

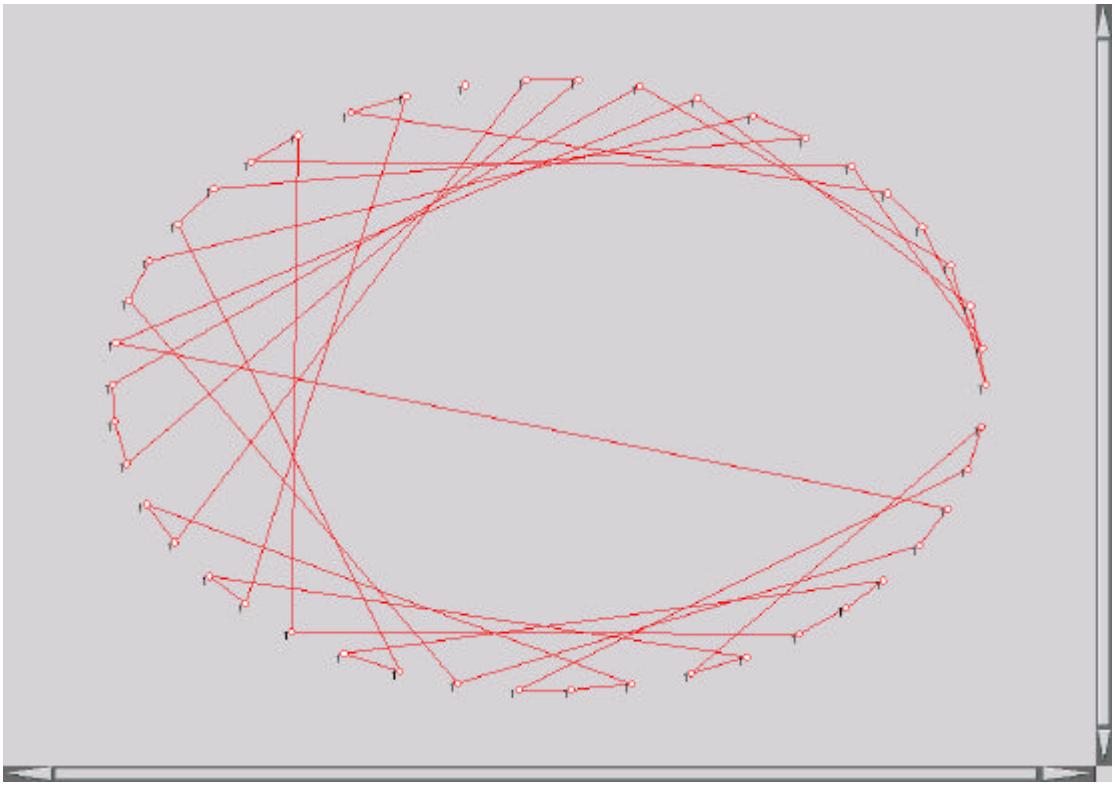

Figure 10: Oriented Graph

The display shows a layout of the nodes and the connections between the nodes. If the attribute node_locations is not given, node positions on a regular polygon will be calculated by the system. Depending on the attribute show_domain, connections will be drawn if the domain of the successor variable is smaller or equal to the attribute value. This allows to avoid unnecessary clutter in the display by showing successors only, if the domains have been sufficiently reduced. If the successor is assigned, the connection is drawn in the assigned_link_color, otherwise the unassigned_link_color is used.

7.9.4 Example

```
run:-
        get_parameters(SizeX,SizeY,NbCycle,Min,Max),
        get_chess_board(L),
        SquareSize is SizeX * SizeY,
        length(Ones,SquareSize),
        Ones :: 1..1,
        get_locations(1,SquareSize,Locations),
        visualize(cycle(NbCycle,L,Ones,Min,Max),visualize_oriented_graph,
                [winw<-400,winh<-400,radius<-5,show_domain<-1000],cycle1),
        visualize(cycle(NbCycle,L,Ones,Min,Max),visualize_graph_lines,
                [post<-no,winw<-400,winh<-400],cycle2),
        find_solution(L).
```

7.9.5 Demo File

mknight.pl

7.10 Geographical tours

The tool class is called

`visualize_geographical_tours.`

7.10.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
show_domain	show successors only if domain is smaller than value	1
radius	size of circle for node	30
marker_color	color or marker	blue
marker_style	style of marker	plus
marker_size	size of marker	5
assigned_link_color	color of assigned successor link	red
unassigned_link_color	color of unassigned successor link	blue
node_locations	list of pairs X-Y, specifying node locations	0

7.10.2 Constraint arguments handled

The tool works with the cycle constraint and uses the following arguments of the constraint:

Argument	Name	Usage
2	Succ	A list of successors
3	Weight	A list of weight values

7.10.3 Display

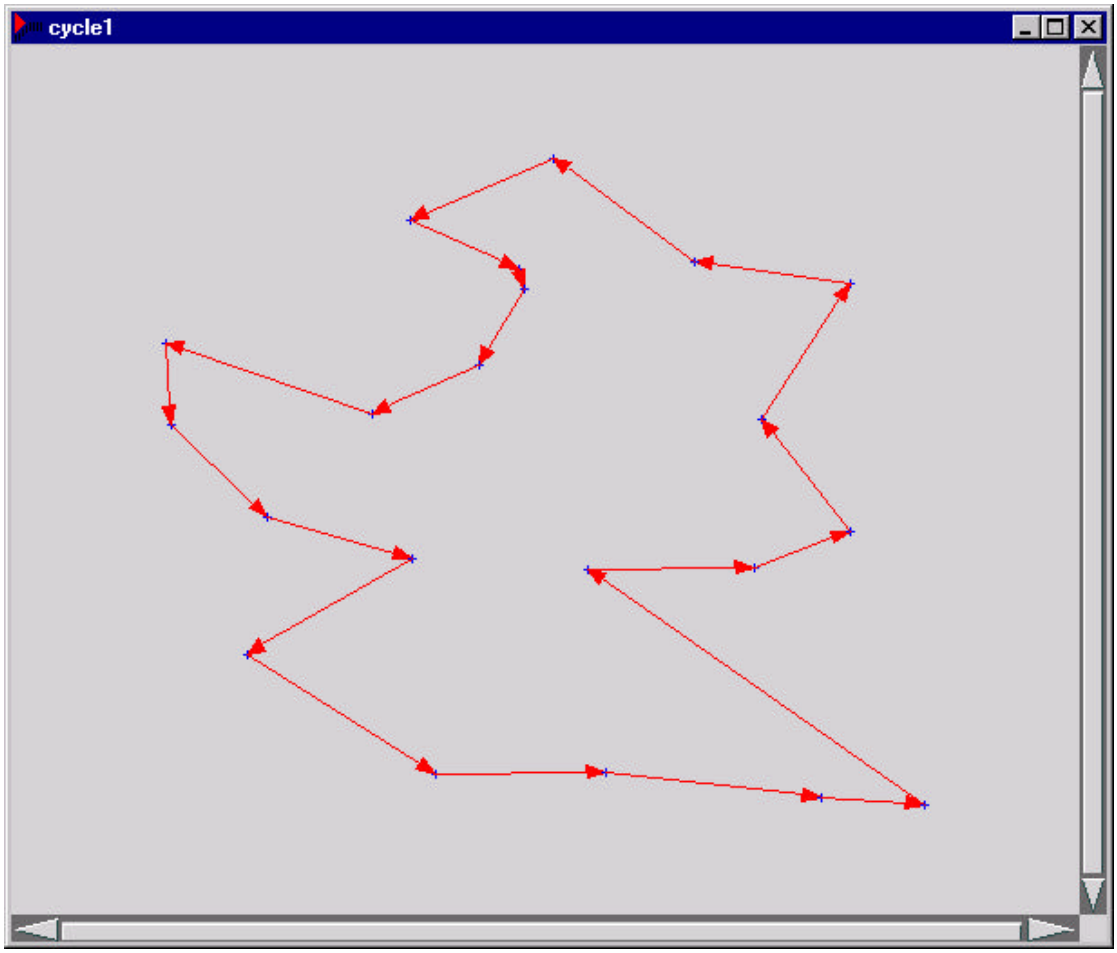

Figure 11: Geographical Tours

The display is similar to the oriented_graph visualizer. For geographical tours, the node location list must be given and should correspond to the locations visited by the tour.

7.10.4 Example

```
run(Balance,Number_tours):-
      variables(1, 21, L, X,C),
      distances(L, L, D),
      length(Ones,21),
      Ones :: 1..1,
      length(Capacity,Number_tours),
      Cost :: 0:2000000,
      location_data(Loc),
      visualize(cycle(Number_tours,X,C,0,99999,unused,unused,
            unused,unused,unused,[Cost,D]),
            visualize_geographical_tours,[node_locations<-Loc,
            winw<-400,winh<-400,show_domain<-1000],cycle1),
      visualize(cycle(Number_tours,X,C,0,99999,unused,unused,
            unused,unused,unused,[Cost,D]),
            visualize_graph_lines,[post<-no,winw<-400,
            winh<-400],cycle2),
      restrict_cost(C),
      min_max((labeling(L, 6, most_constrained, assign),
```

```
            display_solution(L, Cost)), Cost),
      visualize_update.
```

7.10.5 Demo File

mtravel.pl

7.11 Graph lines

The tool class is called

`visualize_graph_lines.`

7.11.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
post	Post constraint when called	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10

7.11.2 Constraint arguments handled

The tool works with the cycle constraint and uses the following arguments of the constraint:

Argument	Name	Usage
2	Succ	A list of successors
3	Weight	A list of weight values
6	Special	A list of special nodes

7.11.3 Display

Figure 12: Graph Lines

This visualizer uses a different view of the successor graph. The display shows all currently existing lines in the cycle graph. A line always starts with a node whose predecessor is not yet assigned. Each line then lists all nodes in sequence which are assigned as successors. If the line is closed, i.e. the last node is linked to the first node, the node is drawn in a gray color. If the successor is not assigned, the domain of the variable is written out.
Initially, most nodes will be placed in lines consisting of only one element. As the labeling continues, more and more lines are merged, until in the solution there are as many lines as cycles in the graph.

7.11.4 Example

```
run:-
      get_parameters(SizeX,SizeY,NbCycle,Min,Max),
      get_chess_board(L),
      SquareSize is SizeX * SizeY,
      length(Ones,SquareSize),
      Ones :: 1..1,
      get_locations(1,SquareSize,Locations),
      visualize(cycle(NbCycle,L,Ones,Min,Max),
            visualize_oriented_graph,[winw<-400,winh<-400,
            radius<-5,show_domain<-1000],cycle1),
      visualize(cycle(NbCycle,L,Ones,Min,Max),
            visualize_graph_lines,
            [post<-no,winw<-400,winh<-400],cycle2),
      find_solution(L).
```

7.11.5 Demo File

mknight.pl

8 Other visualizers

8.1 Variable

The tool class is called

`visualize_variable.`

8.1.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
fixed_color	color used to draw fixed values	yellow
free_color	color used to draw domains not yet fixed	brown

8.1.2 Constraint arguments handled

The tool expects a list of domain variables as its first argument.

8.1.3 Display

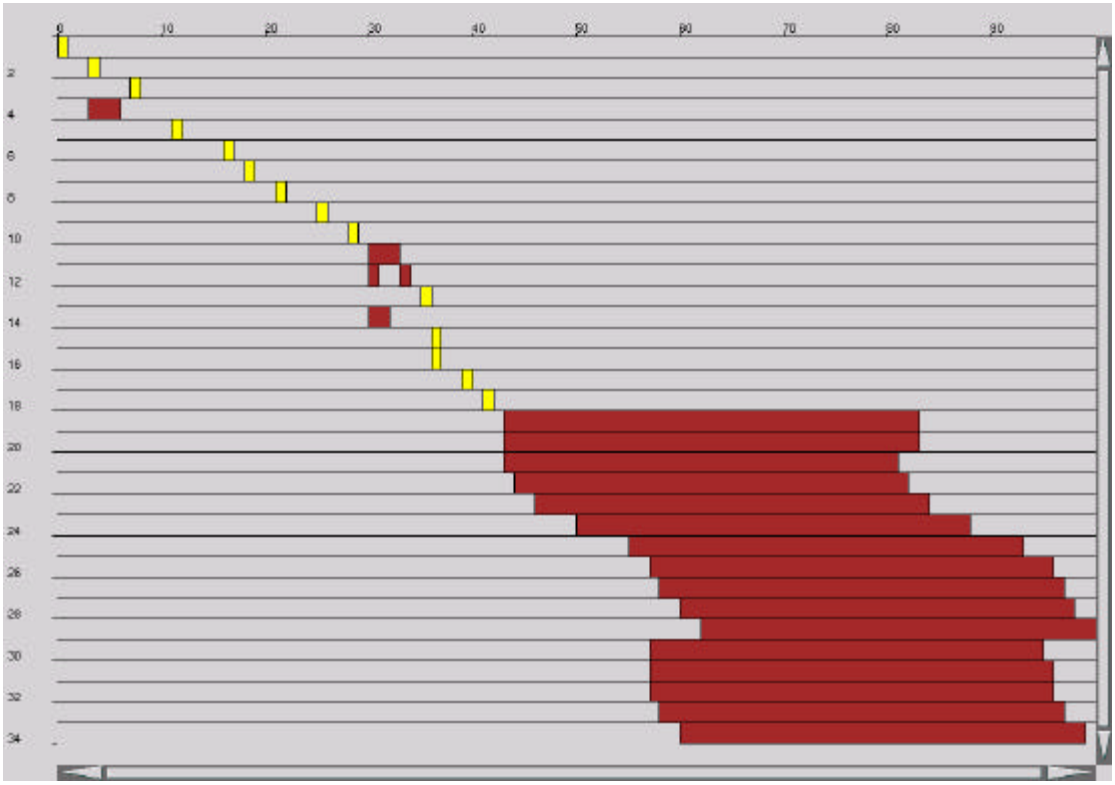

Figure 13: Variable

The display of the domains in this visualizer corresponds to the variable state view in the search tree tool. In the x-axis we display the domain values, and each line shows the domain of one variable. The yellow color is used if the variable is assigned, the brown color is used if the variable in not yet assigned.

8.1.4 Example

```
run(Upper,Last):-
      data(Nr,Dur,Use),
      length(Start,Nr),
      Start :: 0..Last,
      Limit :: 0..Upper,
      End :: 0..Last,
      precedences(L),
      set_precedences(L,Start,Dur),
      visualize(cumulative(Start,Dur,Use,unused,unused,Limit,End,
            unused),visualize_cumulative_resource,
            [winw<-500,winh<-200,wintitle<-test],cumul1),
      visualize(Start,visualize_variable,
            [winx<-0,winy<-230,winw<-500,winh<-400],'Start'),
      search_start(Start,min_max(labeling(Start),End)),
      layout(Start,Dur,Use,Y,Limit,Rectangles),
      visualize(diffn(Rectangles),visualize_placement_2d,[],diffn1),
      labeling(Y,0,most_constrained,indomain),
      show_rectangles(Rectangles,1),
      show_limits(Limit,End).
```

8.1.5 Demo File

ship_search.pl

8.2 Local Stack

The tool class is called

`visualize_local_stack.`

8.2.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
resolution	resolution in x direction, display starts from 0 when limit is exceeded	100
low_color	color used to draw lower stack usage	goldenrod
high_color	color used to draw upper stack usage	orchid

8.2.2 Constraint arguments handled

The tool does not use its first argument, it should be called with a free variable as first argument.

8.2.3 Display

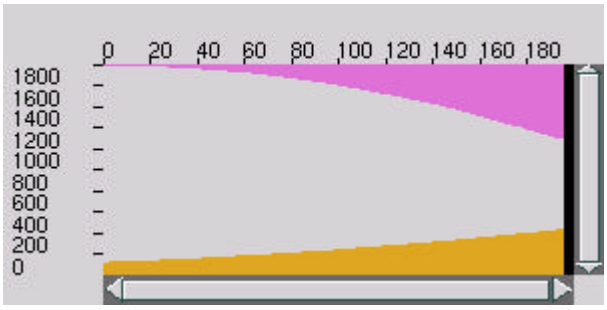

Figure 14: Local Stack

This display shows the current state of the local stack and trail. The local stack is growing from the bottom, the trail from the top. The scale in the y-direction is obtained from the local stack limit currently used. If the two measurements meet, the system is out of memory.

8.2.4 Example

```
all:-
      visualize(_,visualize_local_stack,
            [resolution<-200],local_stack),
      visualize(_,visualize_global_stack,
            [resolution<-200],global_stack),
      visualize(_,visualize_time,
            [resolution<-200,limit <- 500],utime),
      visualize(_,visualize_measure,
            [resolution<-200,limit <- 1000,
            measure_callback_name<-get_choice],backtrack_count),
      between(N, 4, 200),
      utime(_),
      setval(back,0),
      once(solve(N)),
      visualize_update,
      fail.
all.
```

8.2.5 Demo File

queen.pl

8.3 Global Stack

The tool class is called

`visualize_global_stack.`

8.3.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
resolution	resolution in x direction, display starts from 0 when limit is exceeded	100
low_color	color used to draw lower stack usage	goldenrod
high_color	color used to draw upper stack usage	orchid

8.3.2 Constraint arguments handled

The tool does not use its first argument, it should be called with a free variable as first argument.

8.3.3 Display

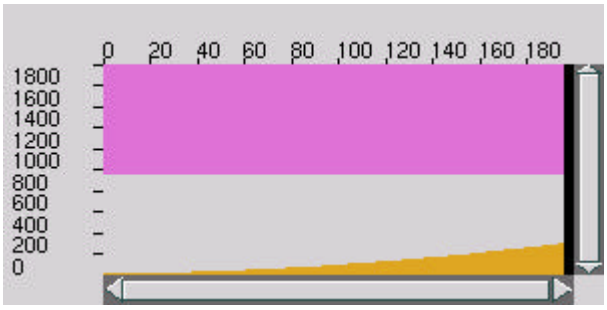

Figure 15: Global Stack

This display shows the current state of the global stack and heap. The global stack is growing from the bottom, the heap from the top. The scale in the y-direction is obtained from the global stack limit currently used. If the two measurements meet, the system is out of memory.
The display will wrap around after a number of measurements given in the resolution attribute.

8.3.4 Example

```
all:-
      visualize(_,visualize_local_stack,
            [resolution<-200],local_stack),
      visualize(_,visualize_global_stack,
            [resolution<-200],global_stack),
      visualize(_,visualize_time,
            [resolution<-200,limit <- 500],utime),
      visualize(_,visualize_measure,
            [resolution<-200,limit <- 1000,
            measure_callback_name<-get_choice],backtrack_count),
      between(N, 4, 200),
      utime(_),
      setval(back,0),
      once(solve(N)),
      visualize_update,
      fail.
all.
```

8.3.5 Demo File

queen.pl

8.4 Measure

The tool class is called

`visualize_measure.`

8.4.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
resolution	resolution in x direction, display starts from 0 when limit is exceeded	100
limit	resolution in y direction, display is scaled to fit into range 0 to limit	10000
measure_color	color used to draw measurement	red
measure_callback_name	name of the callback predicate with arity 2 used to calculate measurement	

8.4.2 Constraint arguments handled

The tool does not use its first argument, it should be called with a free variable as first argument.

8.4.3 Display

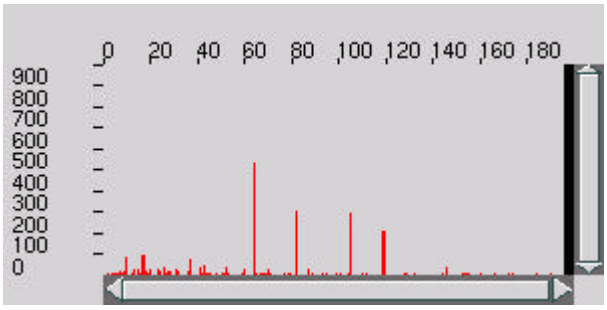

Figure 16: Measure

This display shows resolution many measurements and uses a scale in the y-direction of limit values. Whenever the visualizer is updated, the callback predicate is executed to get the current value to be displayed.

8.4.4 Example

```
all:-
      visualize(_,visualize_local_stack,
            [resolution<-200],local_stack),
      visualize(_,visualize_global_stack,
            [resolution<-200],global_stack),
      visualize(_,visualize_time,
            [resolution<-200,limit <- 500],utime),
      visualize(_,visualize_measure,
            [resolution<-200,limit <- 1000,
            measure_callback_name<-get_choice],backtrack_count),
      between(N, 4, 200),
      utime(_),
      setval(back,0),
      once(solve(N)),
      visualize_update,
      fail.
all.

get_choice(backtrack_count,B):-
      getval(back,B).
```

8.4.5 Demo File

queen.pl

8.5 Time

The tool class is called

`visualize_time.`

8.5.1 User modifiable attributes

The following attributes can be changed in the attribute list:

Attribute	Meaning	Default
winx	Window x position if framed	0
winy	Window y position if framed	0
winw	Window width if framed	300
winh	Window height if framed	150
wintitle	Window title if framed	Name
hide	reserved	no
enable	if no, visualizer is not updated	yes
framed	Window appears in its own frame	yes
scrollbar_width	width of scrollbar in pixel	15
scrollbar_height	height of scrollbar in pixel	15
title_height	height of title in pixel	30
resource_width	width of resource in pixel	50
background_color	background color of windows	lightgray
text_color	text color	black
text_size	text size	10
resolution	resolution in x direction, display starts from 0 when limit is exceeded	100
limit	resolution in y direction, display is scaled to fit into range 0 to limit	10000
measure_color	color used to draw measurement	red

8.5.2 Constraint arguments handled

The tool does not use its first argument, it should be called with a free variable as first argument.

8.5.3 Display

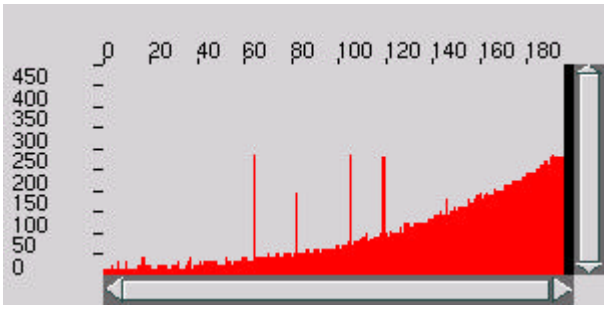

Figure 17: Time

This display is updated with a measurement of the utime (machine time since the last call), everytime the visualizer is updated. The application program should not contain any calls to utime/1, otherwise the measurements will be affected.

8.5.4 Example

```
all:-
      visualize(_,visualize_local_stack,
            [resolution<-200],local_stack),
      visualize(_,visualize_global_stack,
            [resolution<-200],global_stack),
      visualize(_,visualize_time,
            [resolution<-200,limit <- 500],utime),
      visualize(_,visualize_measure,
            [resolution<-200,limit <- 1000,
            measure_callback_name<-get_choice],backtrack_count),
      between(N, 4, 200),
      utime(_),
      setval(back,0),
      once(solve(N)),
      visualize_update,
      fail.
all.
```

8.5.5 Demo File

queen.pl

9 Summary

In this report we have described different visualizers for global constraint concepts. One global constraint, like cumulative, can be used in multiple contexts, to solve different types of constraint concepts. To understand the behavior of the constraint, it is best to describe the propagation in terms of these concepts as well. The global constraint visualizers are all derived from a fundamental class of visualizer objects. The user can overwrite different attributes and methods to adapt the tool to his particular needs. Working together with the search tree tool or as a standalone utility, these visualizers allow a better understanding of the global constraints and an easier exploitation of their functionality.

10 Acknowledgement

The work presented here is part of COSYTEC's work package in the DISCIPL project and was developed using ideas from a number of consortium partners, in particular from UPM. Feedback from a number of early users at COSYTEC was also very valuable.

11 Bibliography

[AB93] A. Aggoun, N. Beldiceanu
Extending CHIP in Order to Solve Complex Scheduling Problems
Journal of Mathematical and Computer Modelling, Vol. 17, No. 7, pages 57-73
Pergamon Press, 1993

[BC94] N. Beldiceanu, E. Contejean
Introducing Global Constraints in CHIP
Journal of Mathematical and Computer Modelling, Vol 20, No 12, pp 97-123, 1994

[Fab97] M. Fabris et al.
CP Debugging Needs and Tools
In Proc. Of the 3rd. Int'l Workshop on Automated Debugging-AADEBUG'97, Pages 103-122,
Linkoping, Sweden, May 1997.

[Mei95] M. Meier
Debugging Constraint Programs.
In Principles and Practice of Constraint Programming, page
204-221,CP'95, Cassis, France, September 1995, Springer, Lecture Notes In Computer Science 976.

[Sch97] C. Schulte
Oz Explorer: A Visual Constraint Programming Tool
Proceedings of the Fourteenth International Conference On Logic
Programming, Leuven, Belgium, pages 286-300. The MIT Press, July 1997.

[SC95] H. Simonis, T. Cornelissens
Modelling Producer/Consumer Constraints
Proc. Principles and Practice of Constraint Programming, Cassis, France, September 1995

[SA97] H. Simonis, A. Aggoun
Search Tree Debugging
Technical Report, COSYTEC SA, October 1997