Chapter 10: Customizing Search (Progressive Party Problem)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- Problem decomposition
  - Decide which problem to solve
  - Not always required to solve complete problem in one go
- Modelling with bin packing
- Customized search routines can bring dramatic improvements
- Understanding what is happening important to find improvements

Problem Definition

Progressive Party

The problem is to timetable a party at a yacht club. Certain boats are to be designated hosts, and the crews of the remaining boats in turn visit the host boats for several successive half-hour periods. The crew of a host boat remains on board to act as hosts while the crew of a guest boat together visits several hosts. Every boat can only host a limited number of guests at a time (its capacity) and crew sizes are different. The party lasts for 6 time periods. A guest boat cannot not revisit a host and guest crews cannot meet more than once. The problem facing the rally organizer is that of minimizing the number of host boats.

Data

Boat	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Capacity	6	8	12	12	12	12	12	10	10	10	10	10	8	8
Crew	2	2	2	2	4	4	4	1	2	2	2	3	4	2
Boat	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Capacity	8	12	8	8	8	8	8	8	7	7	7	7	7	7
Crew	3	6	2	2	4	2	4	5	4	4	2	2	4	5
Boat	29	30	31	32	33	34	35	36	37	38	39	40	41	42
Capacity	6	6	6	6	6	6	6	6	6	6	9	0	0	0
Crew	2	4	2	2	2	2	2	2	4	5	7	2	3	4

High Level Problem Decomposition

- Phase 1: Select minimal set of host boats
  - Manually
- Phase 2: Create plan to assign guest boats to hosts in multiple periods
  - Done as a constraint program

Idea

- Decompose problem into multiple, simpler sub problems
- Solve each sub problem in turn
- Provides solution of complete problem
- Challenge: How to decompose so that good solutions are obtained?
- How to show optimality of solution?

Selecting Host boats

- Some additional side constraints
  - Some boats must be hosts
  - Some boats may not be hosts
- Reason on total or spare capacity
- No solution with 12 boats (with side constraints)

Cork Constraint Computation Centre

Solution to Phase 1

- Select boats 1 to 12 and 14 as hosts
- Many possible problem variants by selecting other host boats

Cork Constraint Computation Centre

Phase 2 Sub-problem

- Host boats and their capacity given
- Ignore host teams, only consider free capacity
- Assign guest teams to host boats

Model

- Assign guest boats to hosts for each time period
- Matrix (size *NrGuests* $\times$ *NrPeriods*) of domain variables x_{ij}
- Variables range over possible hosts 1..*NrHosts*

Constraints

- Each guest boat visits a host boat atmost once
- Two guest boats meet at most once
- All guest boats assigned to a host in a time period fit within spare capacity of host boat

Each guest visits a hosts atmost once

- The variables for a guest and different time periods must be pairwise different
- `alldifferent` constraint on rows of matrix
- $\text{alldifferent}(\{x_{ij} | 1 \leq j \leq \text{NrPeriods}\})$

Two guests meet at most once

- The variables for two guests can have the same value for atmost one time period
- Constraints on each pair of rows in matrix
- $x_{i_1 j} = x_{i_2 j}, i_1 \neq i_2 \Rightarrow x_{i_1 k} \neq x_{i_2 k} 1 \leq k \leq \mathrm{NrPeriods}, k \neq j$

All guests assigned to a host in a time period fit within spare capacity of host boat

- Capacity constraint expressed as bin packing for each time period
- Each host boat is a bin with capacity from 0 to its unused capacity
- Each guest is an item to be assigned to a bin
- Size of item given by crew size of guest boat

Bin Packing Constraint

- Global constraint `bin_packing(Assignment,Sizes,Capacity)`
- Items of different sizes are assigned to bins
- Assignment of item modelled with domain variable (first argument)
- Size of items fixed: integer values (second argument)
- Each bin may have a different capacity
- Capacity of each bin given as a domain variable (third argument)

Main Program

```
top:-
    top(10,6).

top(Problem,Size):-
    problem(Problem,Hosts,Guests),
    model(Hosts,Guests,Size,Matrix),
    writeln(Matrix).
```

Data

```
problem(10,
    [10,10,9,8,8,8,8,8,8,7,6,6,4],
    [7,6,5,5,5,4,4,4,4,4,4,4,4,4,3,
     3,2,2,2,2,2,2,2,2,2,2,2,2,2]).
```


Creating Variables

```
model(Hosts,Guests,NrPeriods,Matrix):-
    length(Hosts,NrHosts),
    length(Guests,NrGuests),
    dim(Matrix,[NrGuests,NrPeriods]),
    Matrix[1..NrGuests,1..NrPeriods] :: 1..NrHosts,
    ...
```

Setting up `alldifferent` constraints

```
...
(for(I,1,NrGuests),
 param(Matrix,NrPeriods) do
    ic:alldifferent(Matrix[I,1..NrPeriods])
),
...
```


Setting up `bin_packing` constraints

```
...
(for(J,1,NrPeriods),
 param(Matrix,NrGuests,Guests,Hosts) do
    make_bins(Hosts,Bins),
    bin_packing(Matrix[1..NrGuests,J],
                Guests,Bins)
),
...
```

Each pair of guests meet atmost once

```
...
(for(I,1,NrGuests-1),
 param(Matrix,NrGuests,NrPeriods) do
    (for(I1,I+1,NrGuests),
     param(Matrix,NrPeriods,I) do
        card_leq(Matrix[I,1..NrPeriods],
                 Matrix[I1,1..NrPeriods],1)
    )
),
...
```


Call search

```
...
extract_array(col,Matrix,List),
assign(List).
```

Make Bin variables

```
make_bins(HostCapacity,Bins):-
    (foreach(Cap,HostCapacity),
     foreach(B,Bins) do
        B :: 0..Cap
    ).
```

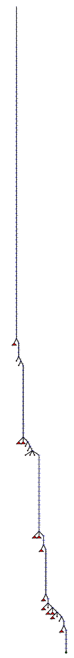

Each pair of guests meet atmost once

```
card_leq(Vector1,Vector2,Card):-
    collection_to_list(Vector1,List1),
    collection_to_list(Vector2,List2),
    (foreach(X,List1),
     foreach(Y,List2),
     fromto(0,A,A+B,Term) do
        #=(X,Y,B)
    ),
    eval(Term) #=< Card.
```

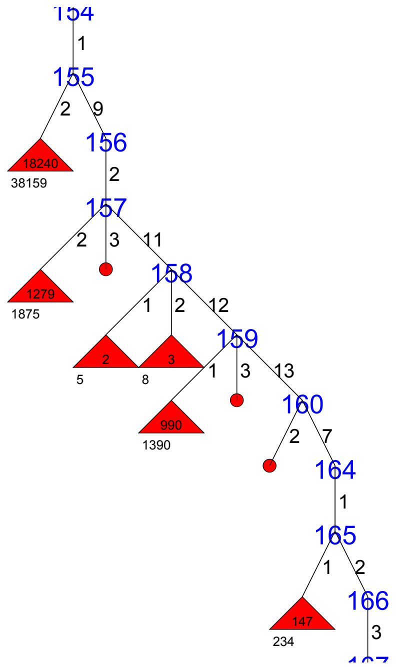
Naive Search

```
assign(List):-
    search(List,0,input_order,indomain,
           complete,[]).
```


Naive Search (Compact view)



Naive Search (Zoomed)



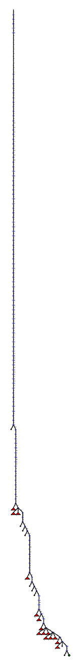
Observations

- Not too many wrong choices
- But very deep backtracking required to discover failure
- Most effort wasted in “dead” parts of search tree

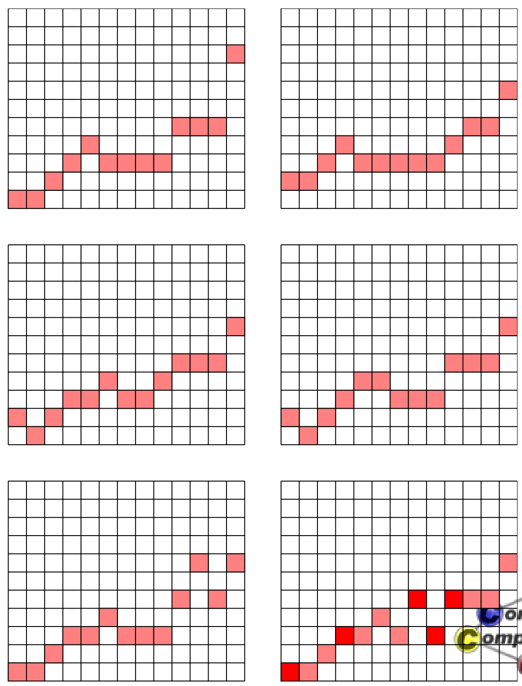
First Fail strategy

```
assign(List):-
    search(List,0,first_fail,indomain,
           complete,[]).
```

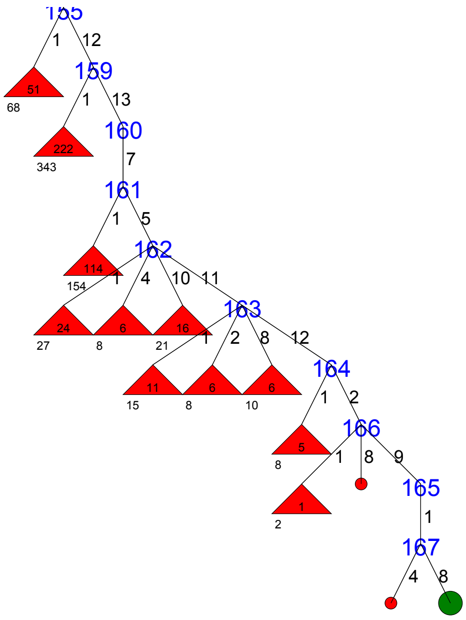

First Fail Search



1	2	3	4	5	6
2	5	1	6	3	4
3	1	6	2	4	5
4	6	2	5	1	3
5	3	2	1	6	7
2	8	4	3	7	1
3	4	5	1	7	2
6	7	1	9	8	10
6	8	10	7	9	3
7	4	9	2	8	12
7	1	5	8	9	11
8	3	4	7	1	2
8	7	11	3	2	9
9	10	7	8	2	13
1	9	12	5	4	7
4	2	12	10	3	5
5	6	8	12	2	11
9	11	3	6	10	12
9	12	6	10	11	2
10	12	13	11	6	1
10	5	7	12	11	9
10	11	9	13	12	8
11	12	8	13	10	9
11	13	7	9	12	1
11	9	10	12	13	8
12	13	9	11	10	4
12	10	13	9	11	8
12	11	8	10	13	1
13	9	8	11	12	10



First Fail Search (Zoomed)



Problem
Program
Search
A Further Decomposition
First Fail Strategy
Layered Search
Layered with Credit Search
Randomized with Restart

Observations

- Assignment not done in row or column mode
- Tree consists of straight parts without backtracking
- ... and nearly fully explored parts

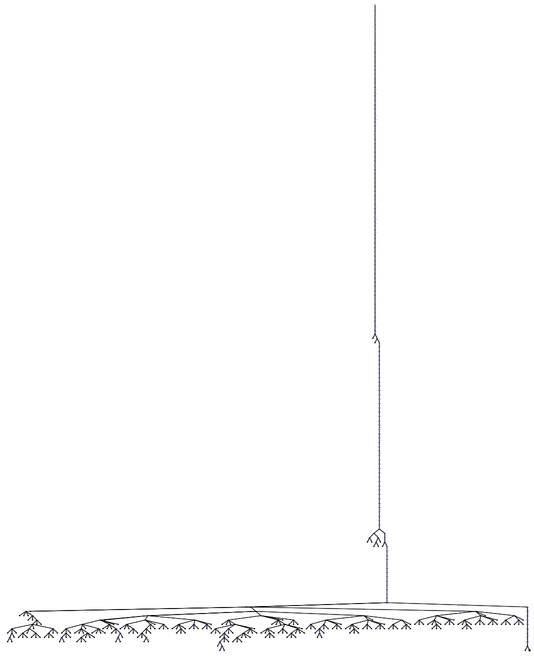
Idea

- Assign variables by time period
- Within one time period, use `first_fail` selection
- Solves bin packing packing for each period completely
- Clearer impact of disequality constraints
- Serial composition of search procedures

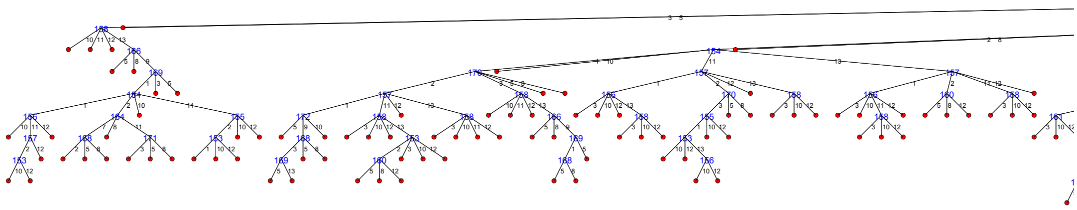
Layered Search

```
assign(Matrix,NrPeriods,NrGuests):-
    (for(J,1,NrPeriods),
     param(Matrix,NrGuests) do
        search(Matrix[1..NrGuests,J],0,
               first_fail,indomain,
               complete,[])
    ).
```

Layered Search



Layered Solution (Zoomed)



Observations

- Deep backtracking for last time period
- No backtracking to earlier time periods required
- Small amount of backtracking at other time periods

Idea

- Use credit based search
- But not for complete search tree
- Loose too much useful work
- Backtrack independently for each time period
- Hope to correct wrong choices without deep backtracking

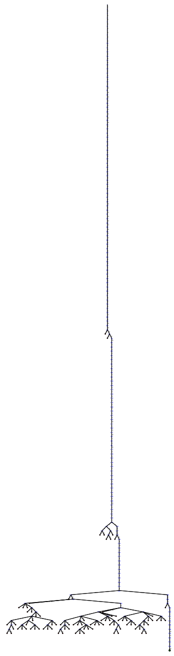
Reminder: Credit Based Search

- Explore top of tree completely, based on credit
- Start with fixed amount of credit
- Each node consumes one credit unit
- Split remaining credit amongst children
- When credit runs out, start bounded backtrack search
- Each branch can use only *K* backtracks
- If this limit is exceeded, jump to unexplored top of tree

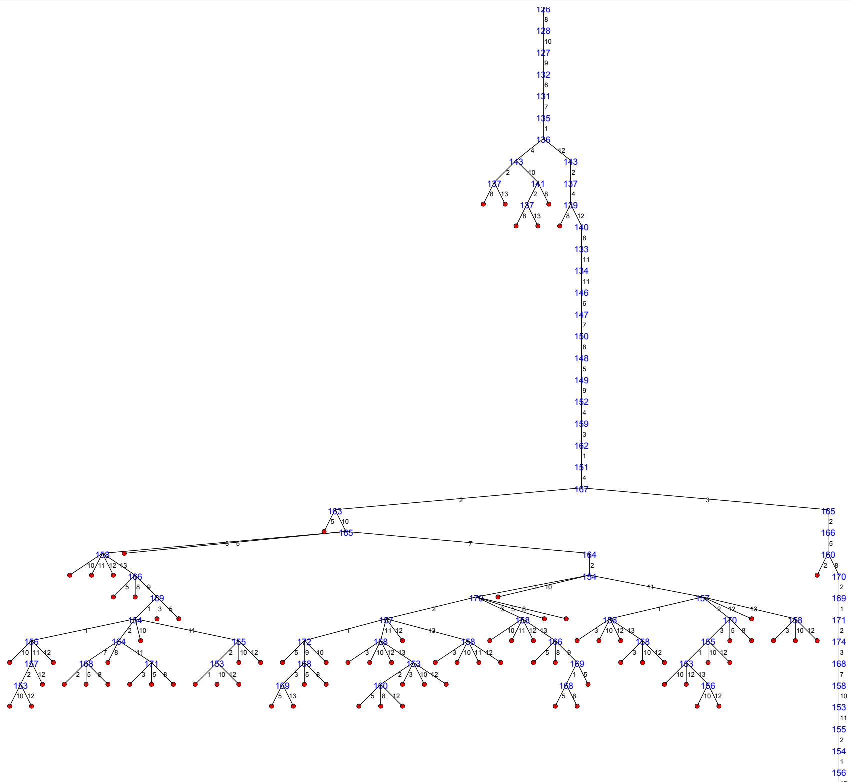
Layered with Credit

```
assign(Matrix,NrPeriods,NrGuests):-
     (for(J,1,NrPeriods),
      param(Matrix,NrGuests) do
          NSq is NrGuests*NrGuests,
          search(Matrix[1..NrGuests,J],0,
                 first_fail,indomain,
                 credit(NSq,10),[])
     ).
```

Layered with Credit Search



Layered with Credit Search (Zoomed)



Observations

- Improved search
- Need more sample problems to really understand impact

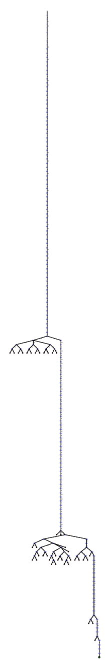
Idea

- Randomize value selection
- Remove bias picking bins in same order
- Allows to add restart
- When spending too much time without finding solution
- Restart search from beginning
- Randomization will explore other initial assignments
- Do not get caught in “dead” part of search tree

Randomized with Restart

```
assign(Matrix,NrPeriods,NrGuests):-
    repeat,
    (for(J,1,NrPeriods),
     param(Matrix,NrGuests) do
        NSq is NrGuests*NrGuests,
        once(search(Matrix[1..NrGuests,J],0,
                    first_fail,indomain_random,
                    credit(NSq,10),[]))
    ),
    !.
```


Randomized Search



Observations

- Avoids deep backtracking in last time periods
- Perhaps by mixing values more evenly
- Impose fewer disequality constraints for last periods
- Easier to find solution
- Should allow to find solutions with more time periods

Changing time periods

Problem	Size	Naive	FF	Layered	Credit	Random
10	5	0.812	1.453	1.515	0.828	1.922
10	6	14.813	2.047	2.093	1.219	2.469
10	7	79.109	3.688	50.250	3.234	3.672
10	8	-	-	141.609	55.156	6.328
10	9	-	-	-	-	10.281

Observations

- Randomized method is strongest for this problem
- Not always fastest for smaller problem sizes
- Restart required for size 9 problems
- Same model, very different results due to search
- Very similar results for other problem instances

Further Improvement

- Idea: There is no real effect of including later time periods in constraint model
- Only current time period matters
- Decomposition: Set up model for one period at a time

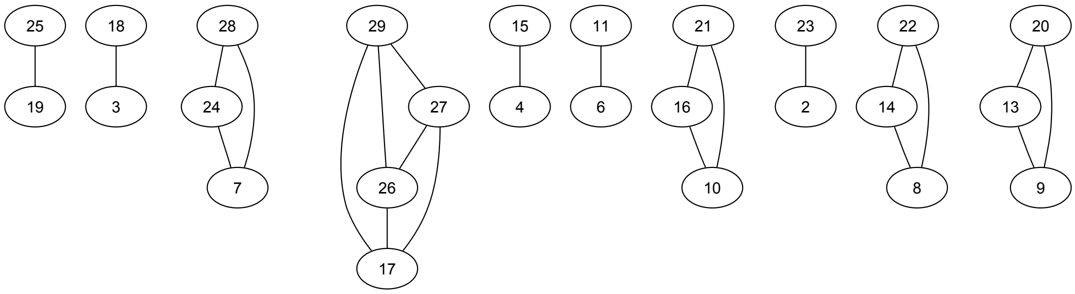
Fine Grained Decomposition

Old	New
Bin packing	Bin packing
Alldifferent	Domain restrictions
Meet at most once	Disequalities between guest boats

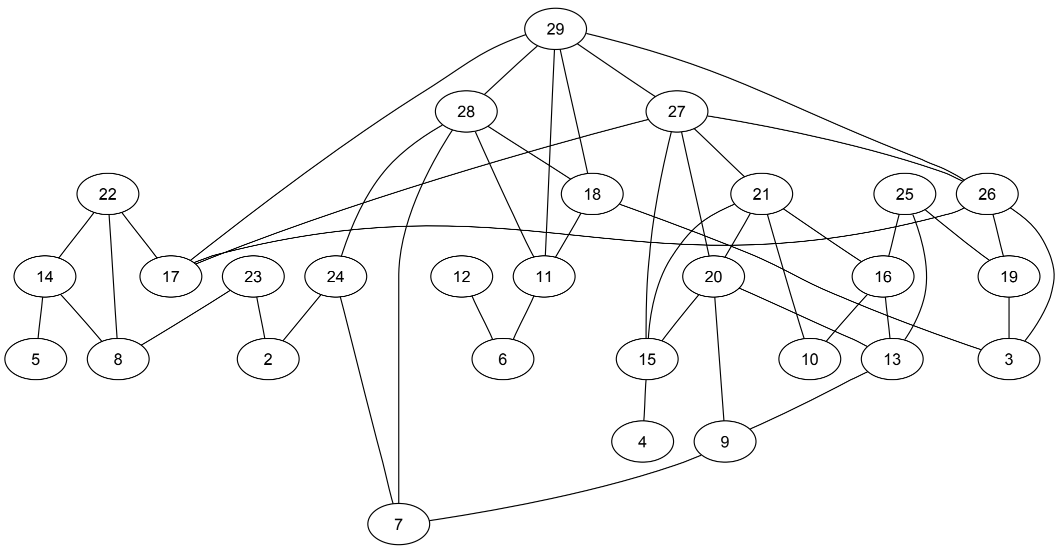
Generated Graph Coloring Problem

- Guest boats = Nodes
- Host boats = Colors
- Disequality constraints = Edges in graph

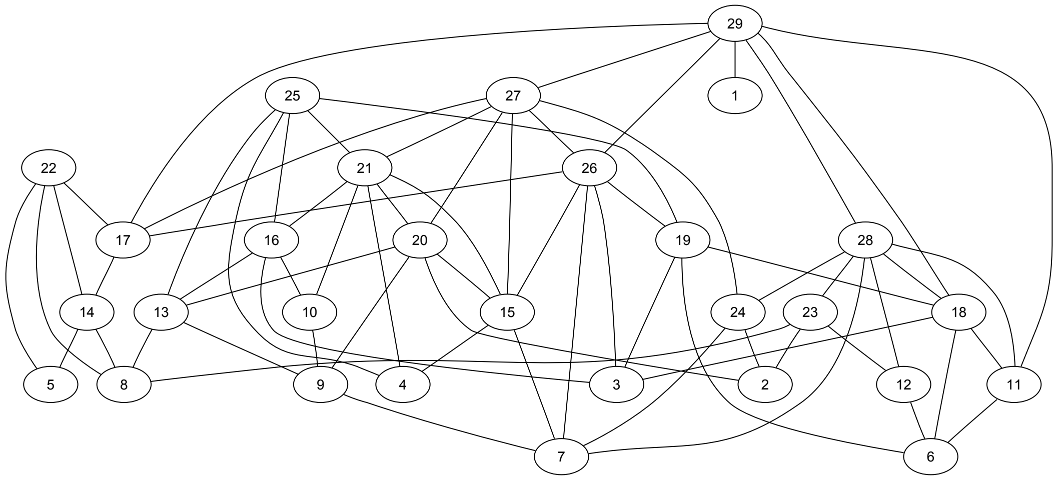
Visualization (Time period 2)



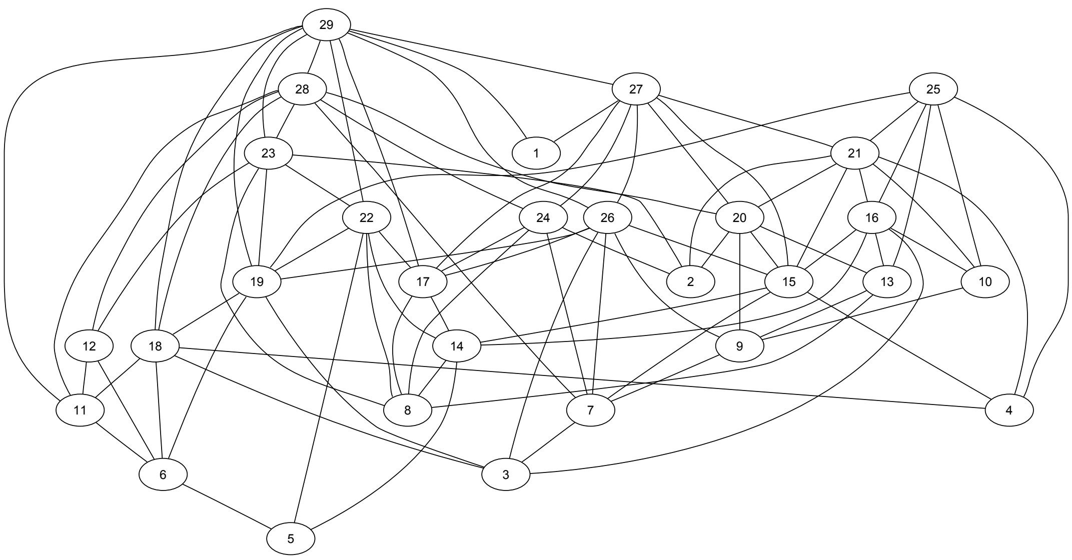
Visualization (Time period 3)



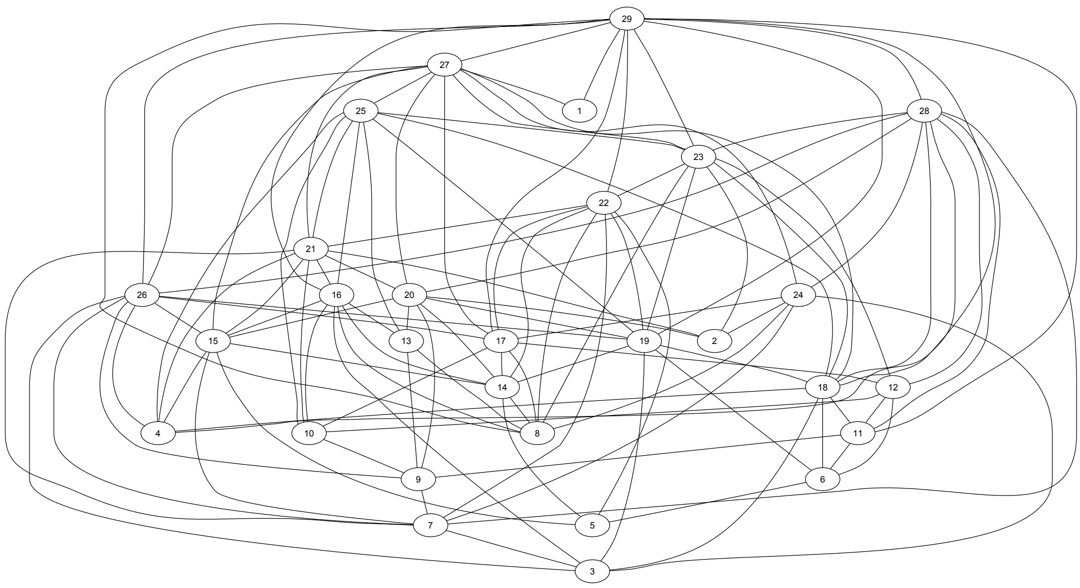
Visualization (Time period 4)



Visualization (Time period 5)



Visualization (Time period 6)



Solving the Graph Coloring Problem

- Use disequality constraints
  - Weak propagation
- Extract `alldifferent` constraints
  - Edge clique cover problem
  - Choice of consistency method
- Use `somedifferent` global constraint
  - Heavy
  - Interaction with `bin packing` constraint

Comparison to Comet

		ECLiPSe 6.0				Comet			
Nr	Size	Solved	Min	Max	Avg	Solved	Min	Max	Avg
1	6	100	0.187	0.343	0.226	100	0.33	0.38	0.35
1	7	100	0.218	0.515	0.271	100	0.39	0.49	0.44
1	8	100	0.250	2.469	0.382	100	0.50	0.72	0.57
1	9	100	0.266	9.906	1.253	100	0.74	1.46	1.01
1	10	100	0.375	136.828	23.314	100	1.47	41.72	4.68
2	6	100	0.218	2.375	0.624	100	0.37	0.52	0.43
2	7	100	0.266	3.453	1.117	100	0.47	1.64	0.73
2	8	100	0.297	15.421	2.348	100	0.75	7.16	2.69
2	9	100	0.469	107.187	20.719	99	4.41	162.96	33.54
3	6	100	0.219	3.266	0.551	100	0.37	0.56	0.43
3	7	100	0.250	3.734	0.889	100	0.49	1.45	0.74
3	8	100	0.296	21.360	2.005	100	0.84	11.64	2.85
3	9	100	1.078	173.266	34.774	96	4.41	164.44	40.10
4	6	100	0.219	9.922	2.443	100	0.39	0.72	0.47
4	7	100	0.360	25.297	3.531	100	0.55	2.33	0.87
4	8	100	0.438	53.547	8.848	100	1.23	11.38	3.68
4	9	63	3.062	494.109	206.324	94	8.35	166.90	59.55
5	6	100	0.203	7.922	1.498	100	0.53	5.29	1.67
5	7	100	0.266	28.000	5.889	100	1.77	132.82	29.72
6	6	100	0.219	15.219	2.147	100	0.58	31.84	2.74
6	7	100	0.407	64.312	11.328	88	3.24	152.37	56.92

More Information

- Brailsford, S., Hubbard, P., Smith, B., Williams, H.:
  Organizing a social event - a difficult problem of combinatorial optimization.
  Computers Ops Res **23** (1996) 845–856
- Beldiceanu, N., Bourreau, E., Chan, P., Rivreau, D.:
  Partial search strategy in CHIP.
  In: 2nd International Conference on Metaheuristics MIC97, Sophia Antipolis, France (1997)
- Van Hentenryck, P., Michel, L.:
  Constraint-Based Local Search.
  MIT Press, Boston (2005)

More Information

- Shaw, P.:
  A constraint for bin packing.
  In: Principles and Practice of Constraint Programming - CP 2004, Toronto, Canada (2004) 648–662
- Richter, Y., Freund, A., Naveh, Y.:
  Generalizing alldifferent: The somedifferent constraint.
  In: Principles and Practice of Constraint Programming - CP 2006, Nantes, France (2006) 468–483
- Philippe Galinier, Alain Hertz, Sandrine Paroz and Gilles Pesant:
  Using Local Search to Speed Up Filtering Algorithms for Some NP-Hard Constraints.
  CPAIOR 2008, Paris, France, May, 2008.

More Information

- Simonis, H.:
  Progress on the progressive party problem (Extended Abstract).
  In: CP-AI-OR 2009, Pittsburgh, PA (2009)