

Standard Models for Finite Domain Constraint Solving

Helmut Simonis

COSYTEC SA

7, rue Jean Rostand
F-91893 Orsay Cedex
France
simonis@cosytec.fr

Aims

- ◆ Present typical applications and their models
- ◆ Step towards a methodology of finite domain modeling
- ◆ Help user of constraint systems
- ◆ Realistic examples
- ◆ Examples of evaluation of models

Overview

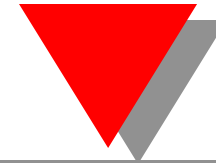
- ◆ **Introduction**
 - Problem classification scheme
 - Finite domain constraint solving
 - Global constraints
 - Search techniques
- ◆ **Models**
 - 4 case studies
 - Model in CHIP
 - Experimental results
- ◆ **Evaluation**
 - Limits and possibilities
 - Other models

Caveats

- ◆ **Work in progress**
 - models evolve in time
 - new constraints, new strategies appear
- ◆ **Each problem is different**
 - additional constraints
 - core of model remains the same
- ◆ **There are no general solutions**
 - need to adapt constraints, strategies
 - verify solutions against real data
- ◆ **Models take advantage of CHIP features**
 - heavy use of global constraints, partial search

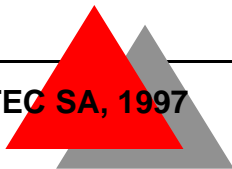
What it does not show

- ◆ **Simplified view of problem**
 - typical projects take 6-12 months
- ◆ **Only some constraints selected**
 - limit complexity
- ◆ **Project issues**
 - knowledge acquisition
 - interfaces
 - data model
- ◆ **Test data given**
 - usually big task in project
- ◆ **No dedicated graphical user interface**
 - standard tools in CHIP



Part 1

Introduction



Problem classification scheme

- ◆ **Last year's tutorial**
 - Pact96
- ◆ **Long paper version available**
 - ASIAN96 workshop
- ◆ **Shows which areas are susceptible to approach**
 - overview of published attempts
 - mentions full systems
- ◆ **Identifies strong areas for finite domain constraints**

Overview

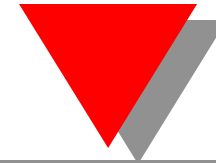
- ◆ Hardware design
- ◆ Compilation
- ◆ Financial problems
- ◆ Placement
- ◆ Cutting problems
- ◆ Stand allocation
- ◆ Air traffic control
- ◆ Frequency allocation
- ◆ Network configuration
- ◆ Product design
- ◆ Production step planning
- ◆ Production sequencing
- ◆ Production scheduling
- ◆ Satellite tasking
- ◆ Maintenance planning
- ◆ Product blending
- ◆ Time tabling
- ◆ Crew rotation
- ◆ Aircraft rotation
- ◆ Transport
- ◆ Personnel assignment
- ◆ Personnel requirement planning

Four central topics

- ◆ **Scheduling**
 - Production scheduling
 - Project planning
- ◆ **Assignment**
 - Parking assignment
 - Platform allocation
- ◆ **Transport**
 - Lorry, train, airlines
- ◆ **Crew assignment**
 - Train, airlines

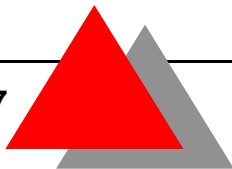
This year's models

- ◆ **Assignment**
 - stand allocation for airport
- ◆ **Scheduling**
 - resource restricted scheduling
- ◆ **Transport**
 - airline fleet rotation
- ◆ **Personnel time tabling**
 - nurse scheduling
 - nice model, good results
- ◆ **No crew assignment**
 - too complex for given time
 - data too sensitive



Incomplete finite domain solver

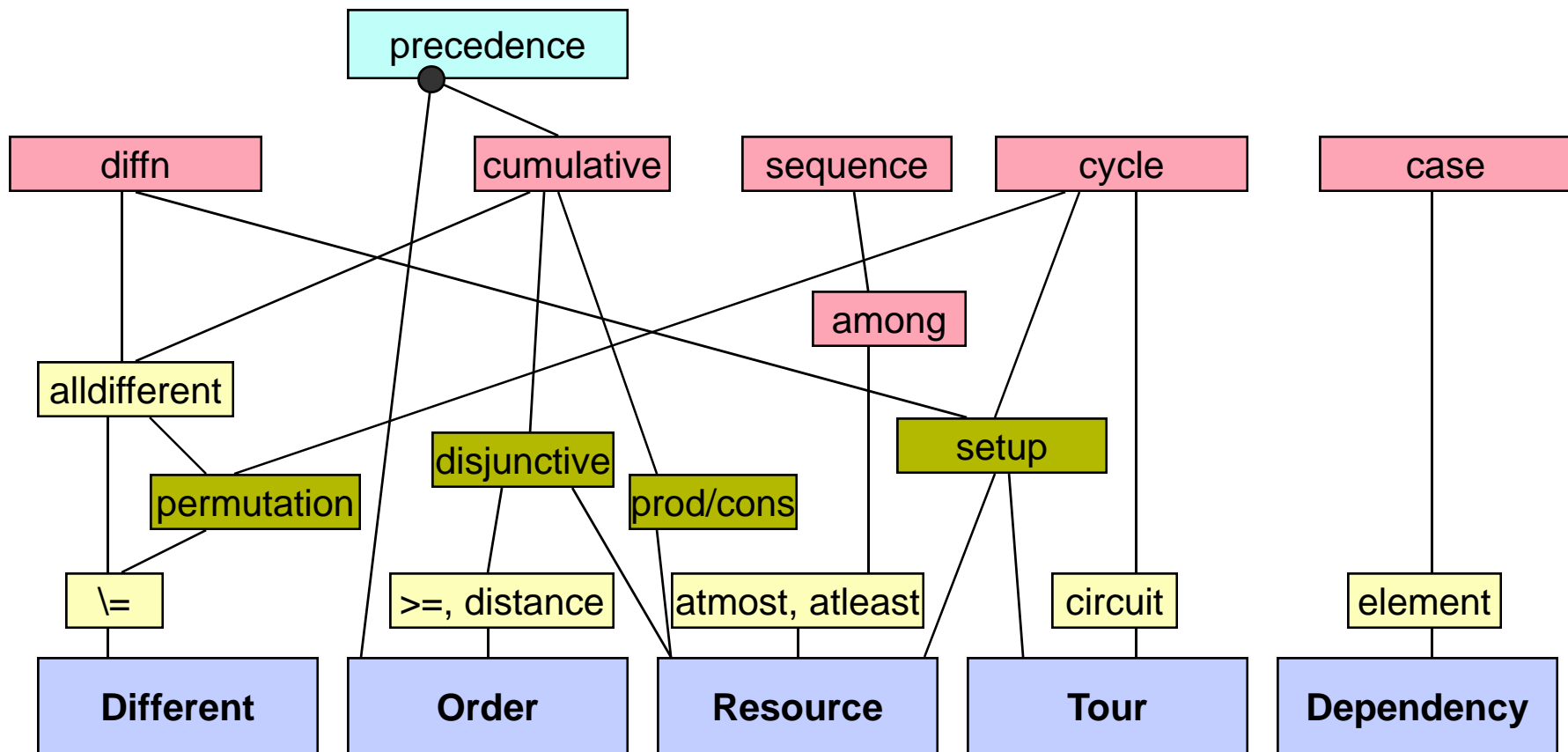
- ◆ **Domain**
 - finite sets of values
 - subsets of natural numbers
- ◆ **Need for enumeration**
- ◆ **Classification criteria**
 - constraint granularity
 - richness of constraint sets
 - propagation results
 - user definable constraints/control
- ◆ **Methods**
 - explicit domain representation
 - bound propagation/ removal of interior values
 - heuristics based on domains



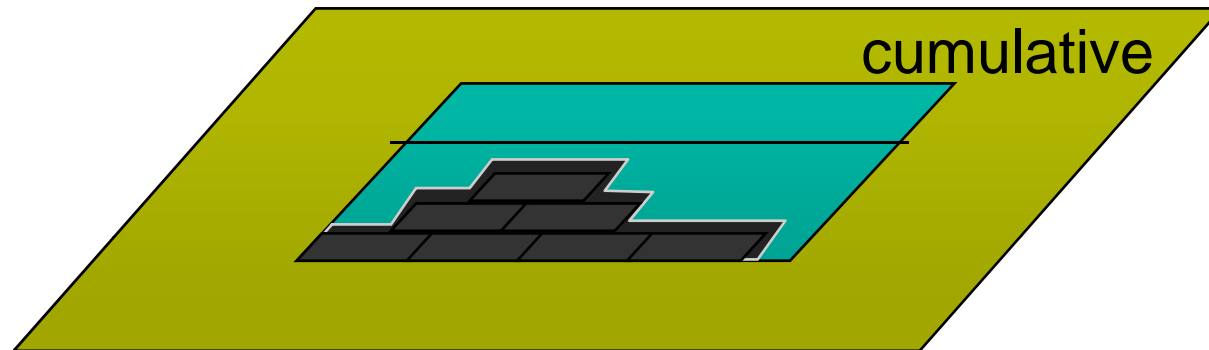
Global constraints

- ◆ **Work on sets of variables**
 - global conditions, not local constraints
- ◆ **Semantic methods**
 - Operations Research
 - spatial algorithms
 - graph theory
 - network flows
- ◆ **Building blocks (high-level constraint primitives)**
 - as general as possible
 - multi-purpose
 - very strong propagation (within acceptable algorithmic complexity)

Constraint morphology



The Cumulative global constraint



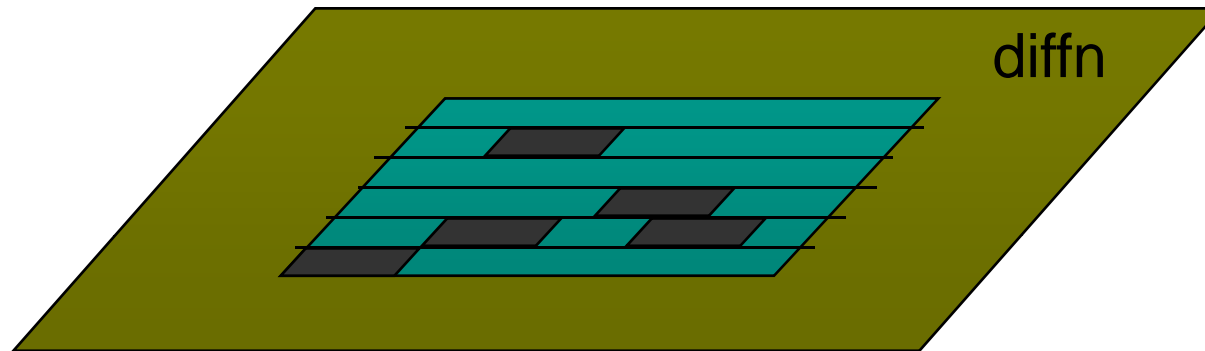
◆ Cumulative constraint

- Resource limits over periods of time
- Upper/lower limits
- Soft/hard limits
- Gradual constraint relaxation

◆ Application

- Resource restrictive scheduling, producer consumer constraints, disjunctive schedule, manpower constraints, overtime

The Diffn global constraint



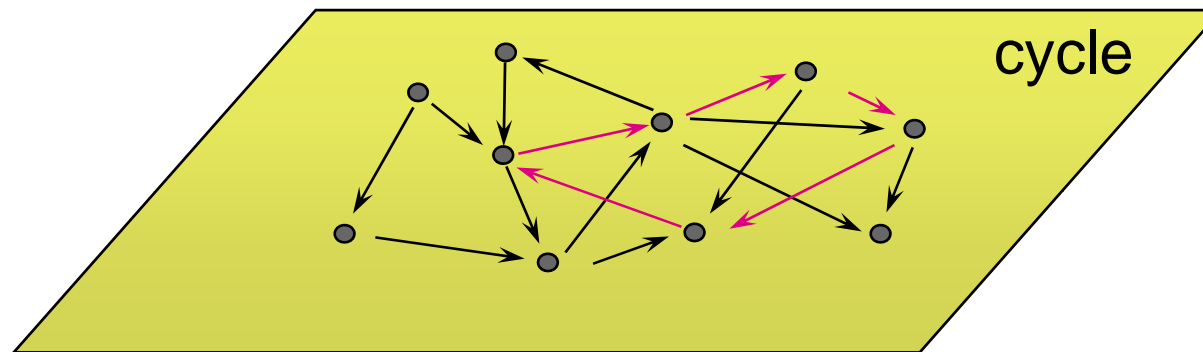
◆ Diffn constraint

- non overlapping areas on n-dimensional rectangles
- distances between rectangles
- limit use of areas

◆ Application

- layout, packing, resource assignment, setup, distribution planning, time-tabling

The Cycle global constraint



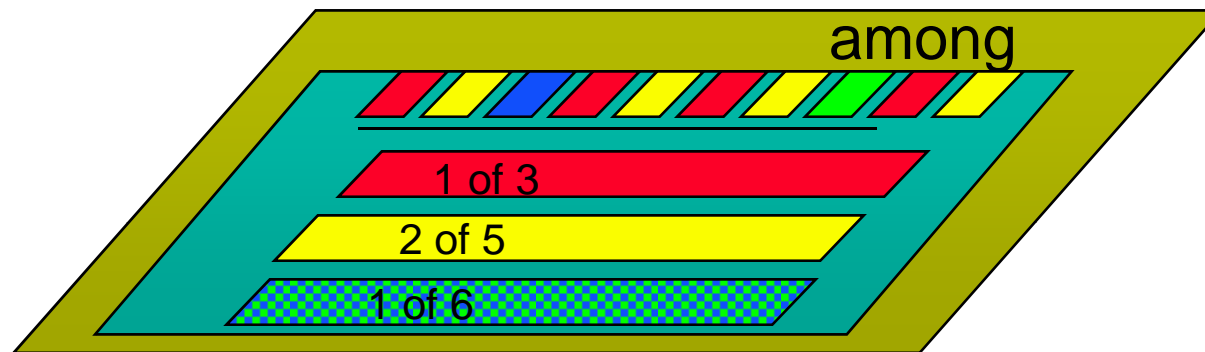
◆ Cycle constraint

- Finds cycles in directed graphs with minimal cost
- Assign resources, find compatible start dates

◆ Applications

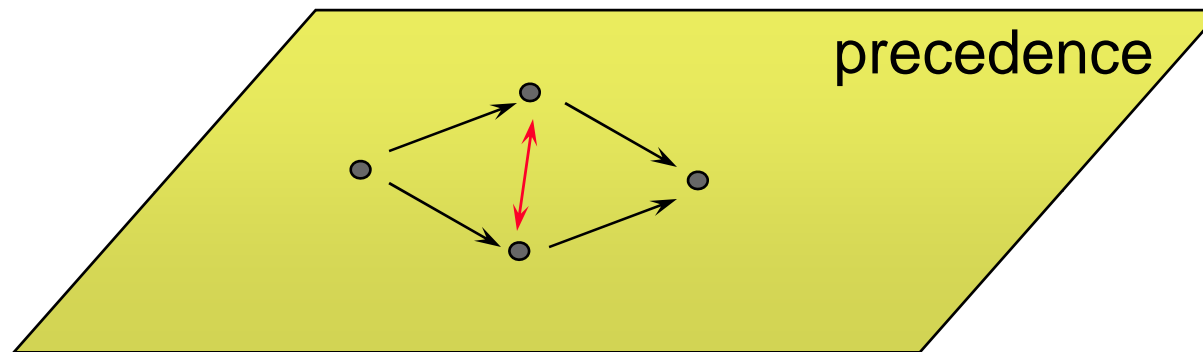
- Tour planning, personnel rotation, distribution problems, production sequencing

The Among global constraint



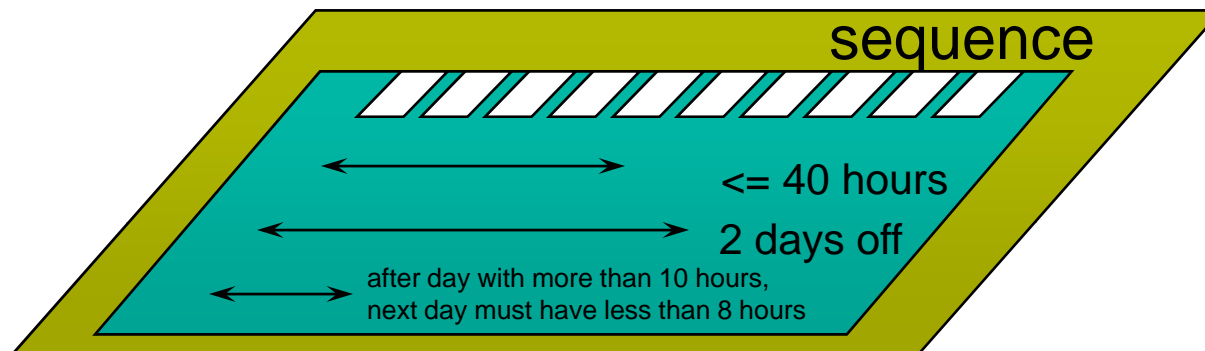
- ◆ **Among constraint**
 - How often do values occur in (sub)sequences
 - based on counting arguments
 - interaction between sequences
- ◆ **Applications**
 - production sequencing, time tabling, coloring problems, set covering

The Precedence global constraint



- ◆ **Precedence constraint**
 - Combine resource constraints and precedence networks
 - Reasoning on latency (position in network)
 - Co-operation between multiple resources
- ◆ **Applications**
 - resource restricted scheduling, channel routing, frequency allocation

The Sequence global constraint



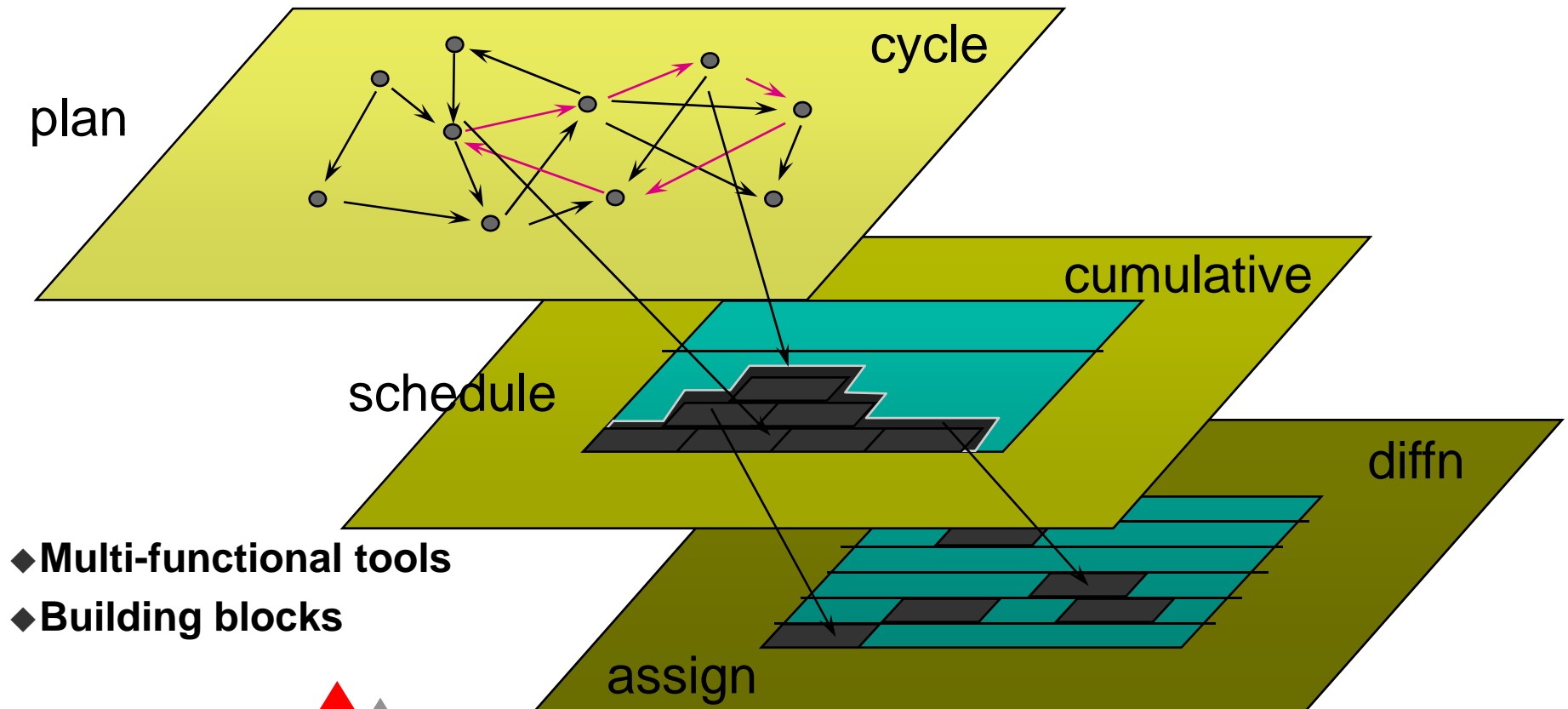
- ◆ **Sequence constraint**

- constraints on pattern inside sequences
- combinatorial pattern matching
- counting arguments

- ◆ **Applications**

- Time tabling, personnel assignment,
- work rules, scheduling with daily working time limits

The power of global constraints

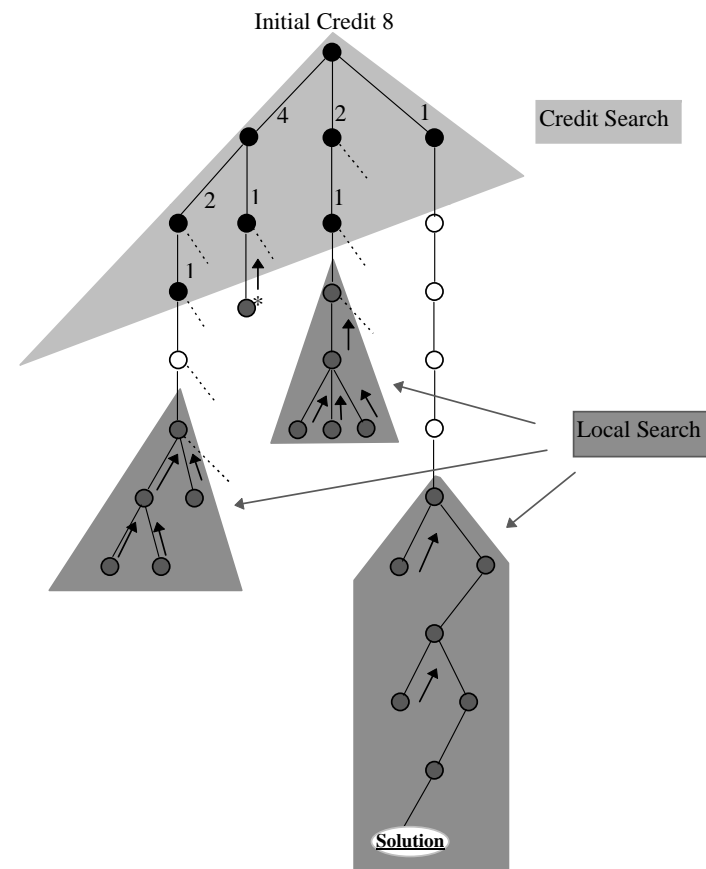


Search strategies

- ◆ **How to find values for variables**
- ◆ **Central to application of strategies/heuristics**
- ◆ **Chronological backtracking**
 - explores full search tree
 - complete
 - often stuck in one part of tree
- ◆ **Partial search**
 - combination of credit based search with nearly deterministic local search
 - not complete
 - polynomial complexity
 - used to explore different parts of search tree in systematic fashion
 - uses normal variable and value selection criteria

Example of partial search tree

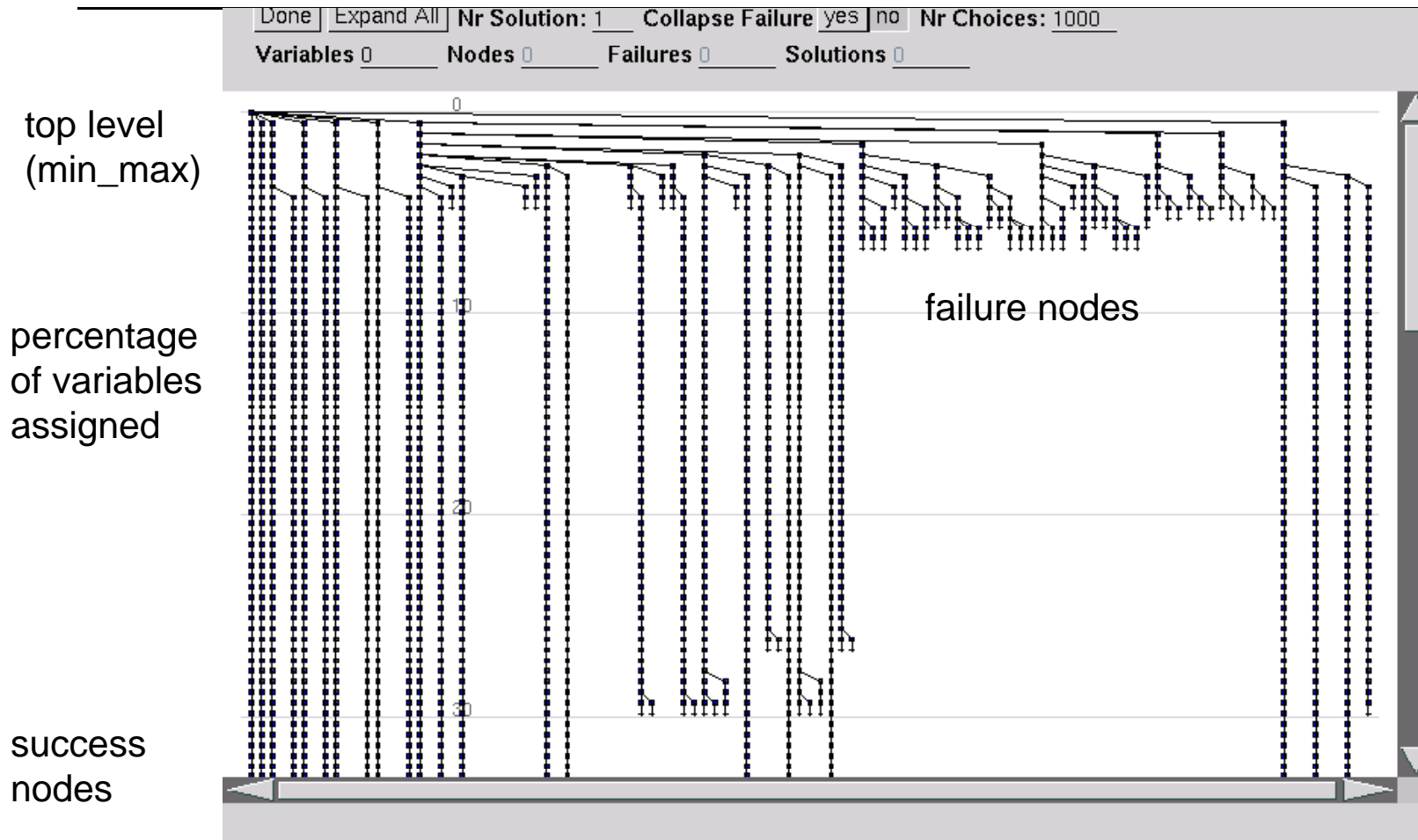
```
partial_dfs([X1..X10],  
            8,  
            10,  
            my_delete,  
            my_indomain,  
            4,  
            part(1,2)),
```

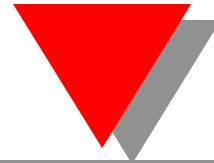


Search tree visualization

- ◆ **Generation of search tree representation at run-time**
- ◆ **Shows parent child relation, failed sub-trees, success nodes**
- ◆ **In examples here**
 - leaf failure nodes suppressed
 - failure trees not collapsed
- ◆ **Interface a set a simple meta-call predicates**
- ◆ **Supported by work in DISCIPL Esprit project**
 - Declarative debugging
 - Visualization of constraint programming results

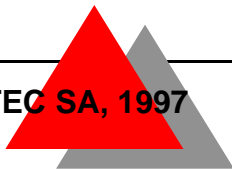
Typical search tree visual





Part 2

Models



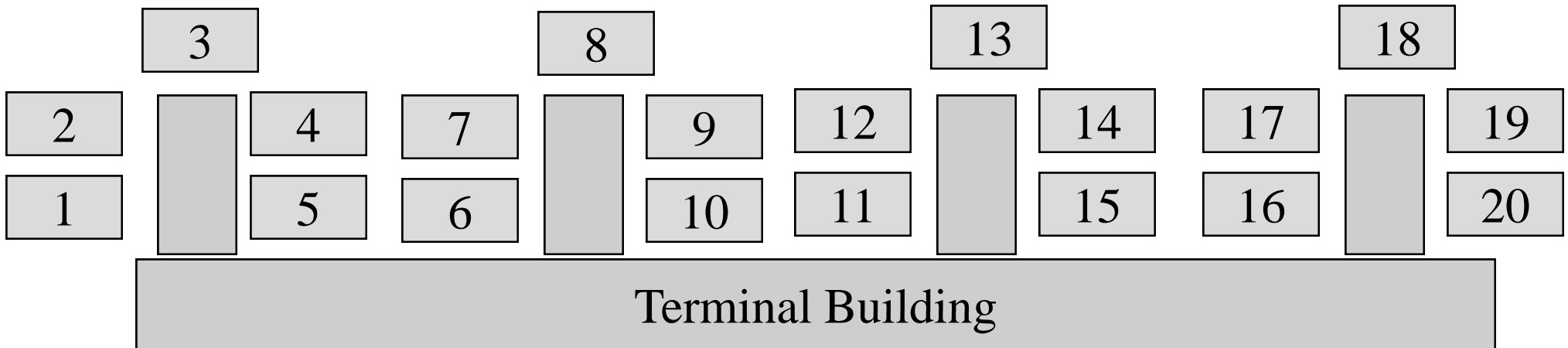
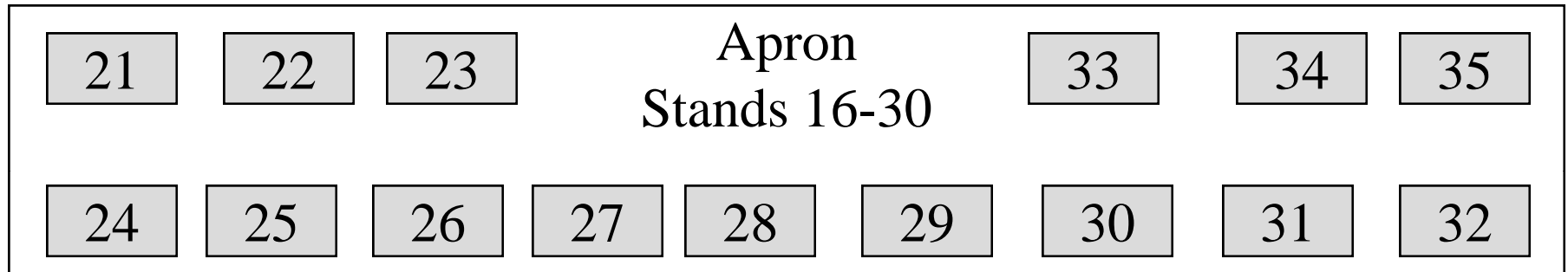
Example 1: Stand allocation

- ◆ **Stand allocation at an airport**
- ◆ **Aircraft arrive and depart at different times**
- ◆ **Must find parking for each plane during its stay**
- ◆ **Depending on aircraft type, some stands are not permitted**
- ◆ **Contact stands connect directly to terminal, passengers can board easily**
- ◆ **Apron stands need passenger busses**

Objectives

- ◆ Maximize number of passengers which can board directly
- ◆ Minimize pax in aircraft parked on apron
- ◆ Park all aircraft on compatible stands
- ◆ Never allow two aircraft on same stand at the same time

Airport layout



Aircraft type compatibility

- ◆ 747: stands 3, 8, 13, 18 and apron
- ◆ 74M: stands 3, 8, 13, 18,19 and apron
- ◆ 310,AB3: stands 2, 3, 4, 7, 8, 9, 12, 13, 14 and apron
- ◆ 737,727,320,M81: on all stands
- ◆ SSC: stands 7,8,9 (first class only)
- ◆ ATR, etc: only on apron (door too low for ramp)

Flight data

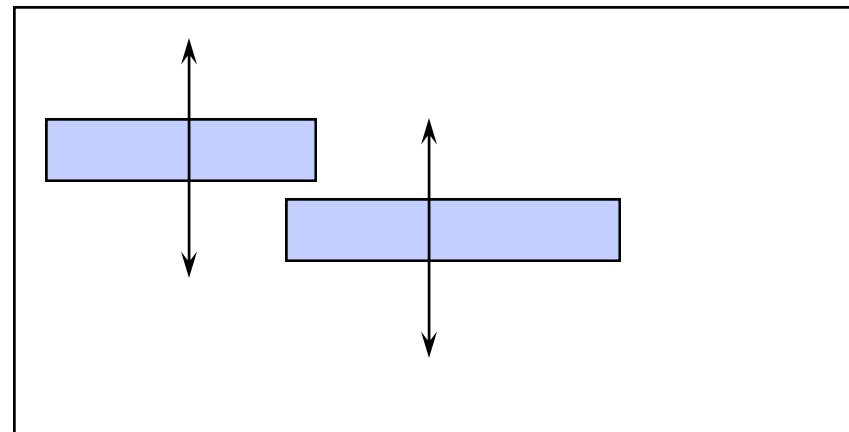
- ◆ Describe aircraft type, arrival and departure time and number of passengers (arrival + departure)
- ◆ park('AB3',0510,1015,132).
- ◆ Table of flight information inside program

Model

- ◆ **One variable per flight**
 - domain: all stands which are allowed for flight
- ◆ **Constraints**
 - two different flights either
 - ◆ do not overlap in time, or
 - ◆ are not one the same stand
 - overlap can be checked statically, as arrival and departure time is known
- ◆ **Expression of constraints**
 - sets of disequalities between flights
 - sets of alldifferent (one per time point)
 - ◆ all flights which are parked at this time
 - ◆ subset reduction possible

Expression with diffn

- ◆ Each flight is represented as rectangle
- ◆ Start, duration and end are fixed (x-dimension)
- ◆ Stand assignment is free (y-dimension), height is 1
- ◆ Non-overlap condition is expressed by one diffn constraint



Cost

- ◆ **Element constraint expresses link between stand assignment and cost**
 - if parked at terminal, cost is 0
 - if parked on apron, cost is #pax
 - link generated statically
- ◆ **element(Stand,[0,0,0,0.....0,Pax,Pax.....Pax], Cost)**

all terminal stands have cost 0

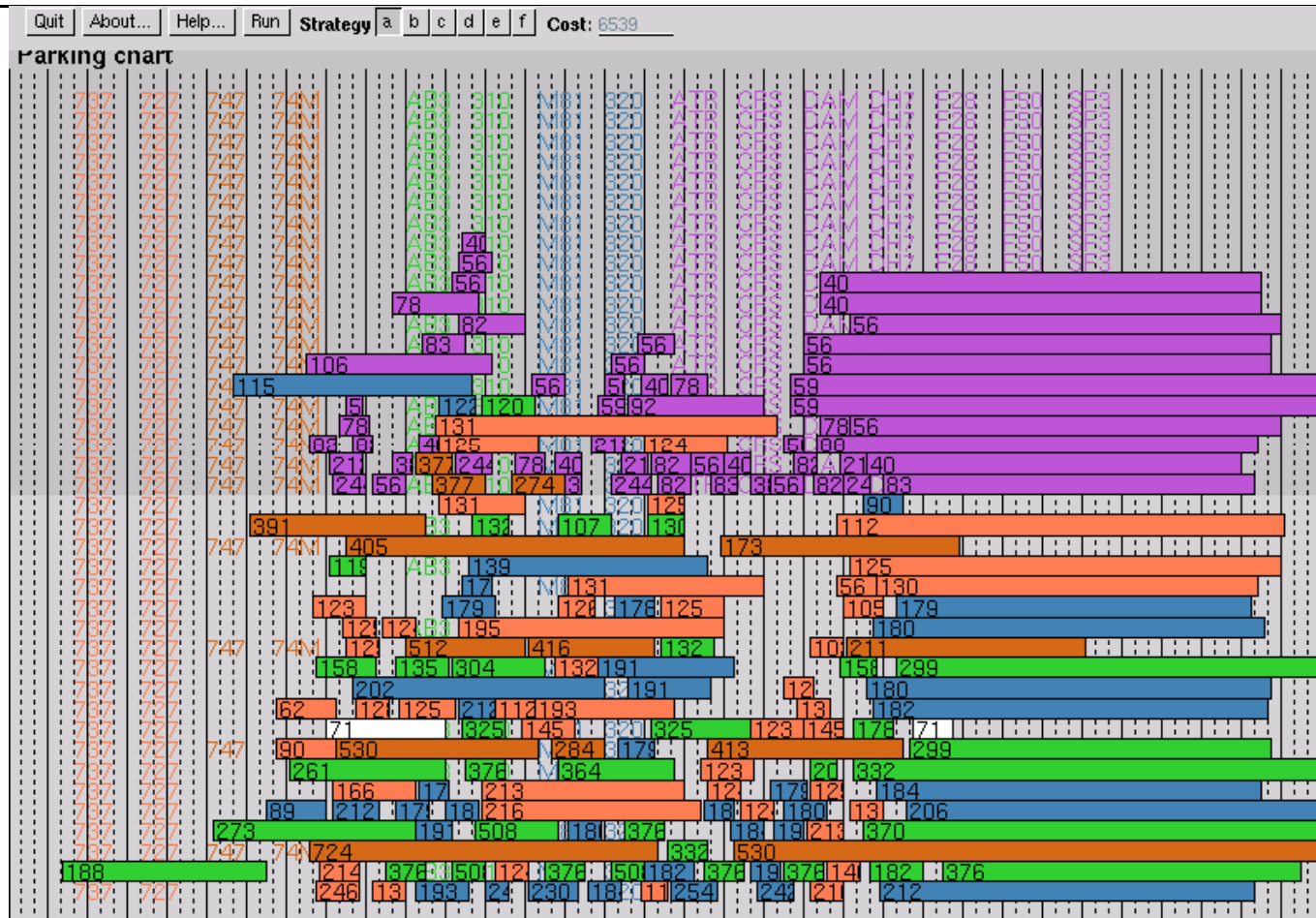
all apron stands have cost Pax

Total cost is sum of costs of all flights

Data

- ◆ **Airport configuration fictitious**
 - no domestic/international distinction
 - no conflict between stands
- ◆ **Flight data from an existing airport**
 - all flights planned for one day
- ◆ **Snapshot of real data**
 - problem changes very rapidly
 - estimated 20 seconds average time between updates

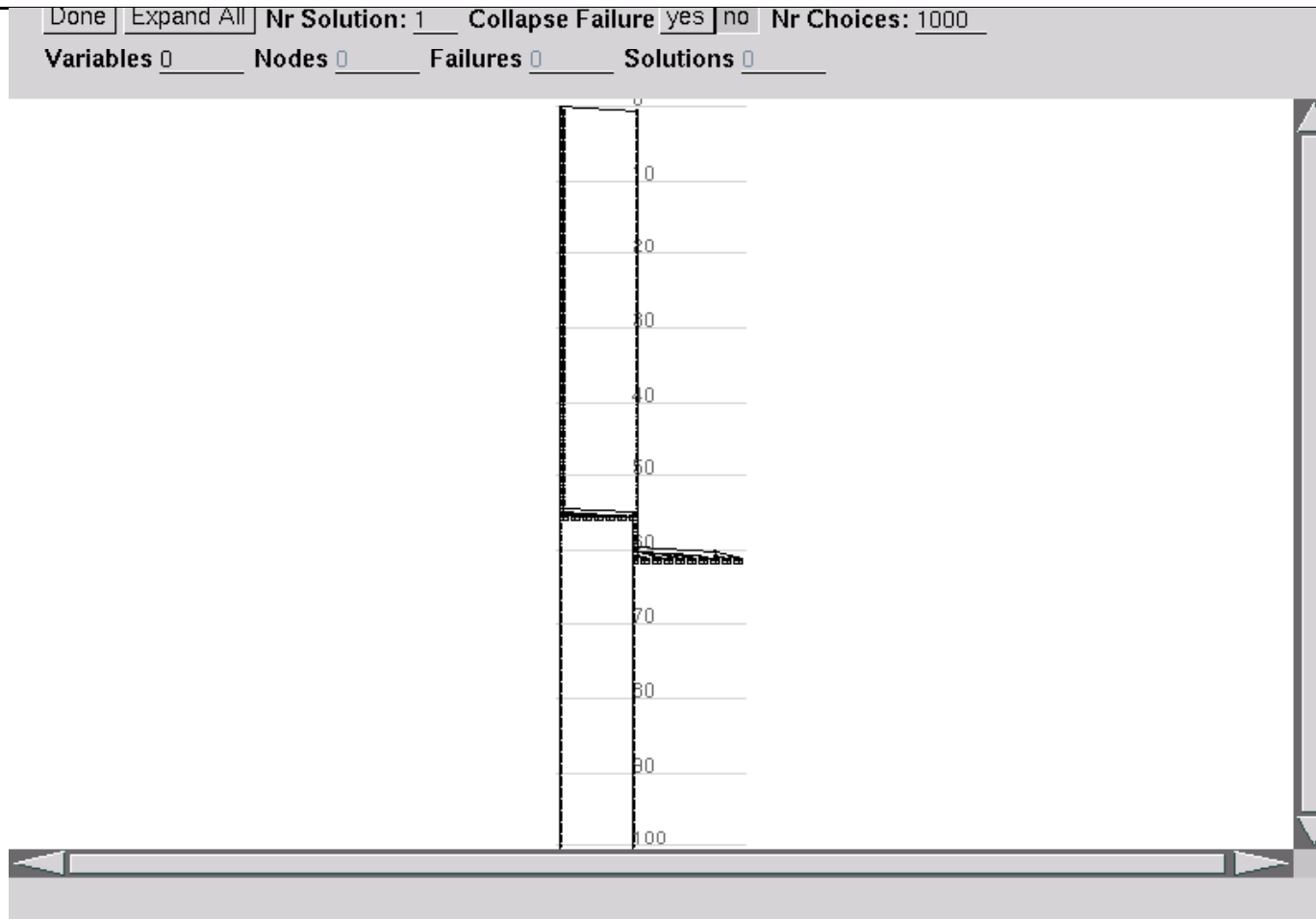
Solution



Search strategy A

- ◆ **Static variable sorting**
 - variable selection
 - ◆ sorted by decreasing number of pax
 - ◆ try to fit large aircraft first (low cost)
 - value selection
 - ◆ indomain/1 : assign stands on terminal first
- ◆ **First solution found without backtracking**
 - problem usually not too tight
 - spare capacity for problems/emergencies
- ◆ **Optimization not successful, shallow backtracking**
label([]).
label([Stand|Rest]):-
 indomain(Stand),
 label(Rest).

Search tree display (Strategy A)

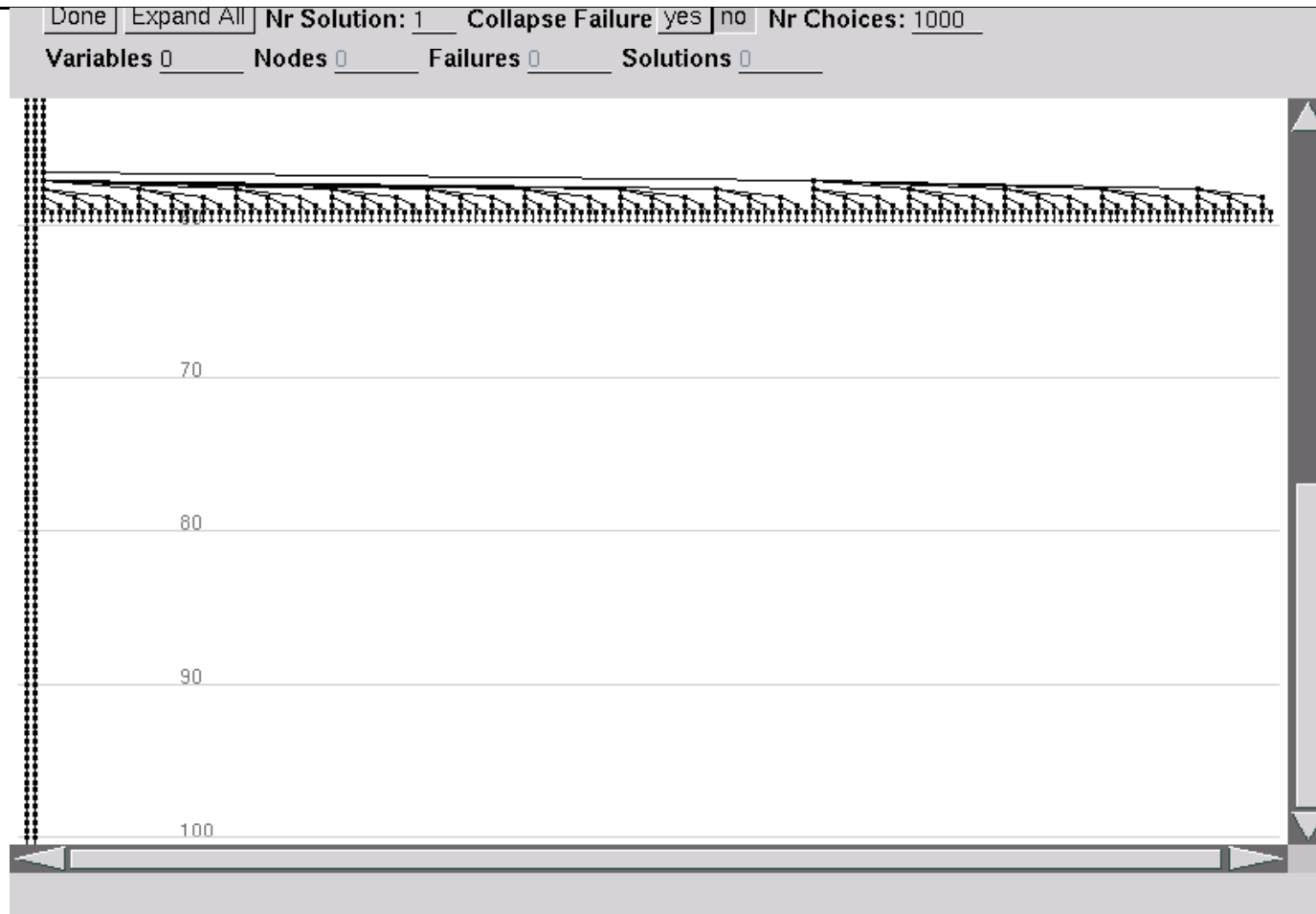


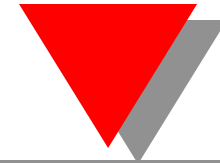
Search strategy B

- ◆ **Assign a preference to each stand**
 - high preference = many aircraft like to be parked there
 - low preference = place can not be used by many aircraft
- ◆ **Element constraint to link stand allocation to preference**
- ◆ **Try a low preference first, then assign stand**
- ◆ **Better first solution**
- ◆ **Still quite shallow backtracking**
- ◆ **Code:**

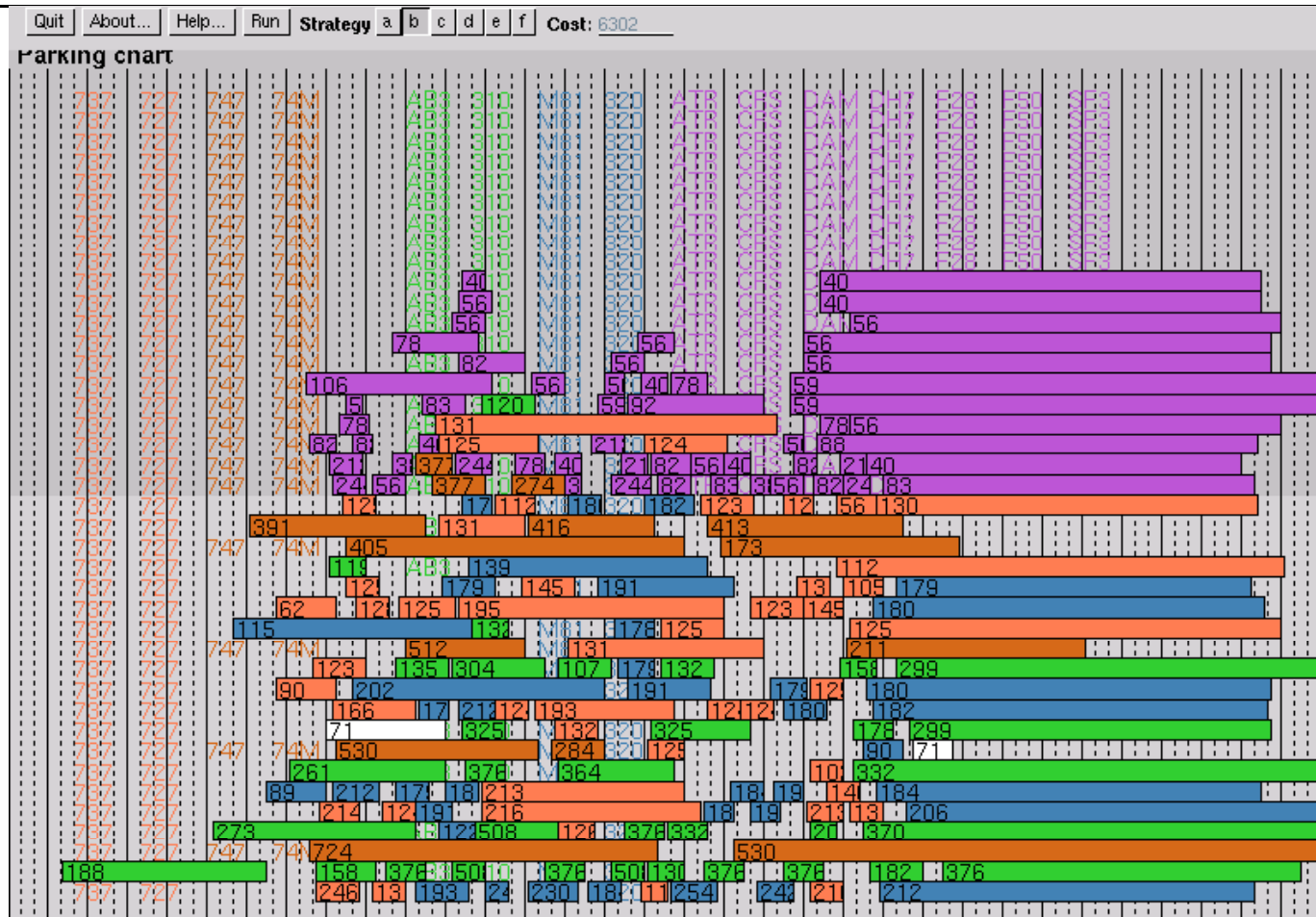
```
label([]).  
label([Pref-Stand|Rest]):-  
    indomain(Pref),  
    indomain(Stand),  
    label(Rest).
```

Search tree display (Strategy B)





Best solution strategy B

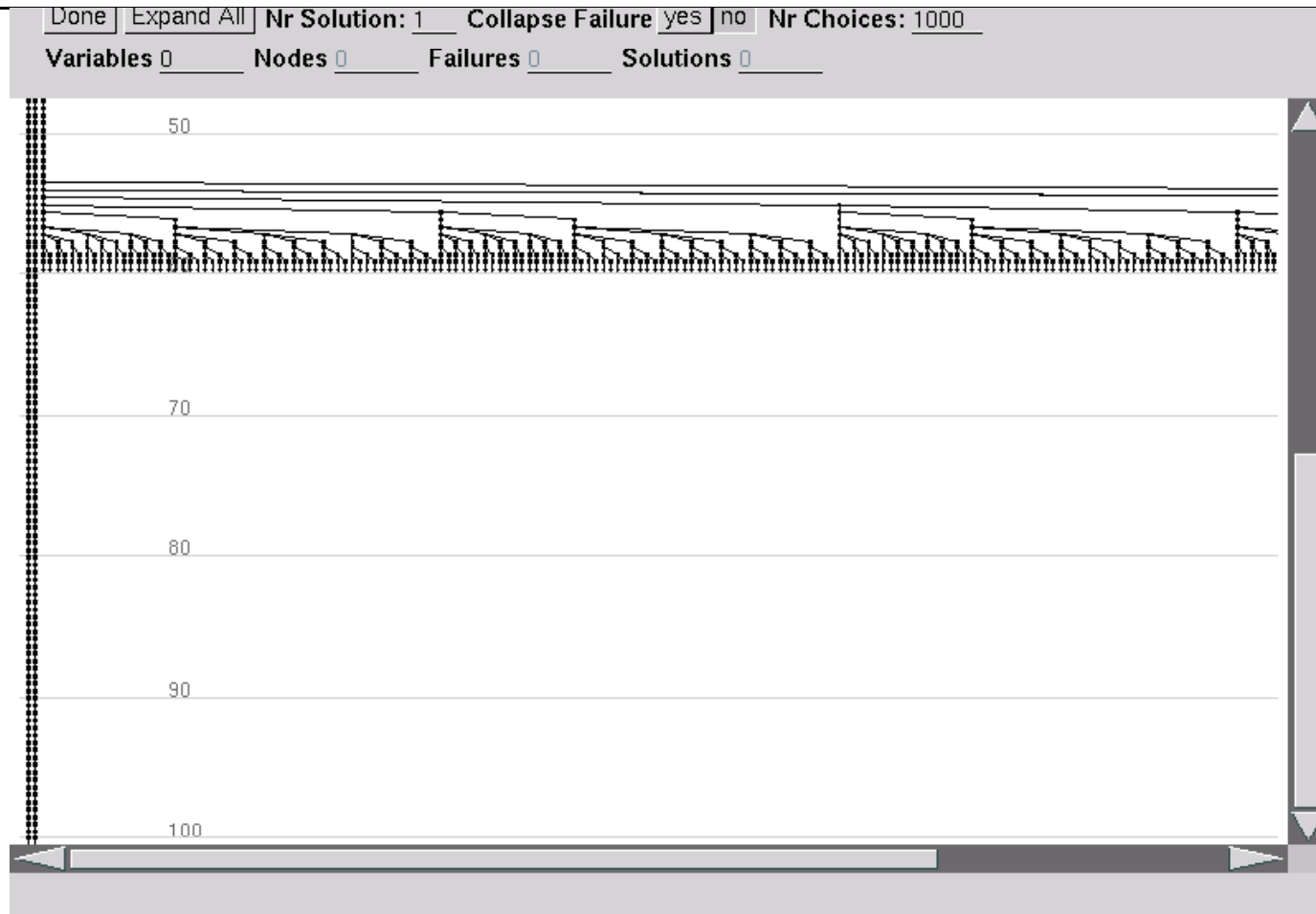


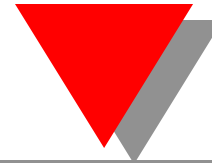
Search strategy C

- ◆ **Incomplete search**
 - for each flight, try all possible preference values
 - but only try one possible stand per preference
- ◆ **Increases number of alternatives considered**
- ◆ **Removes many symmetries**
- ◆ **But may also remove valid alternative solutions**
- ◆ **Code:**

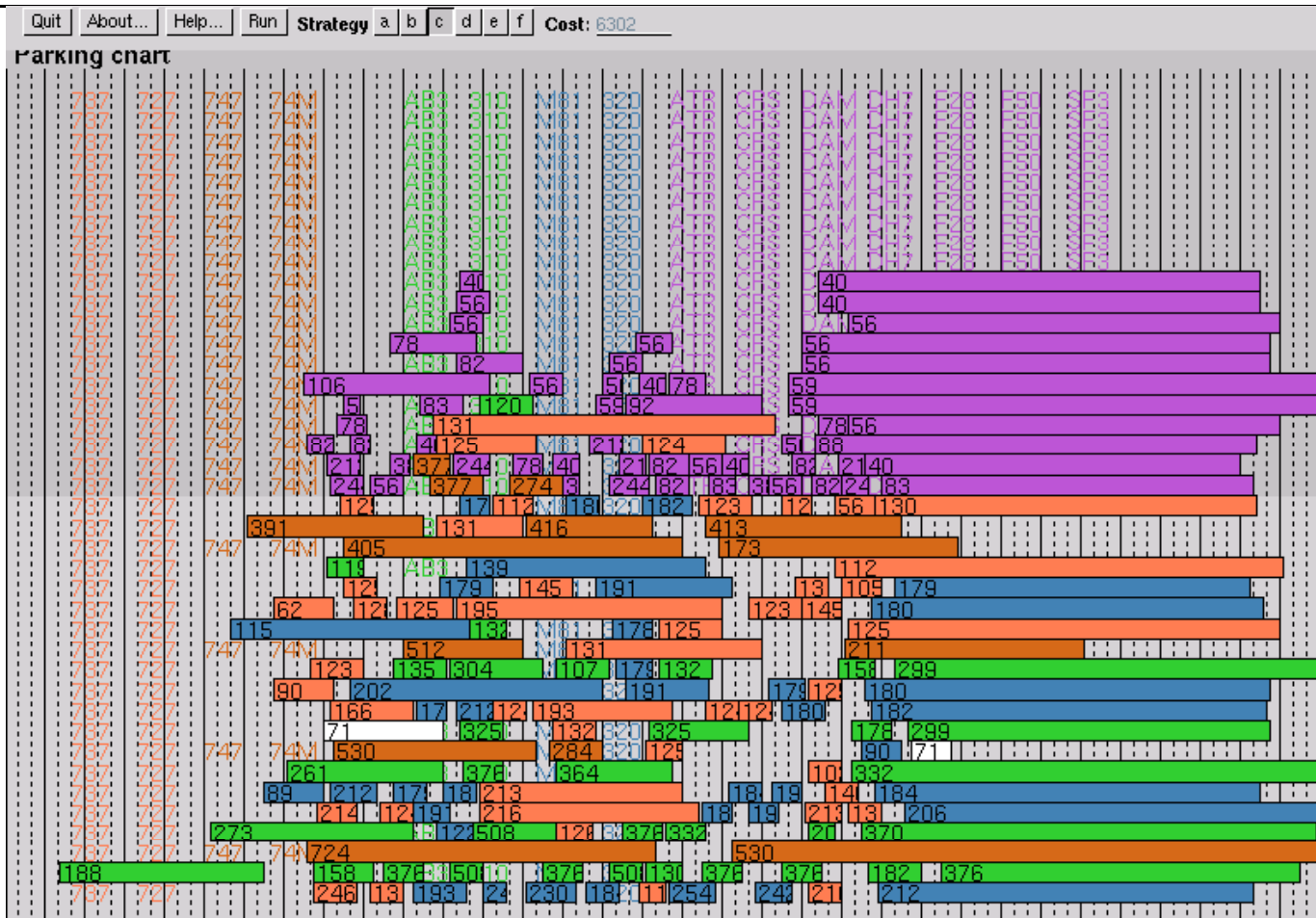
```
label([]).  
label([Pref-Stand|Rest]):-  
    indomain(Pref),  
    once(indomain(Stand)),  
    label(Rest).
```

Search tree display (Strategy C)



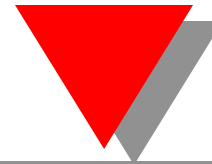


Best solution strategy C

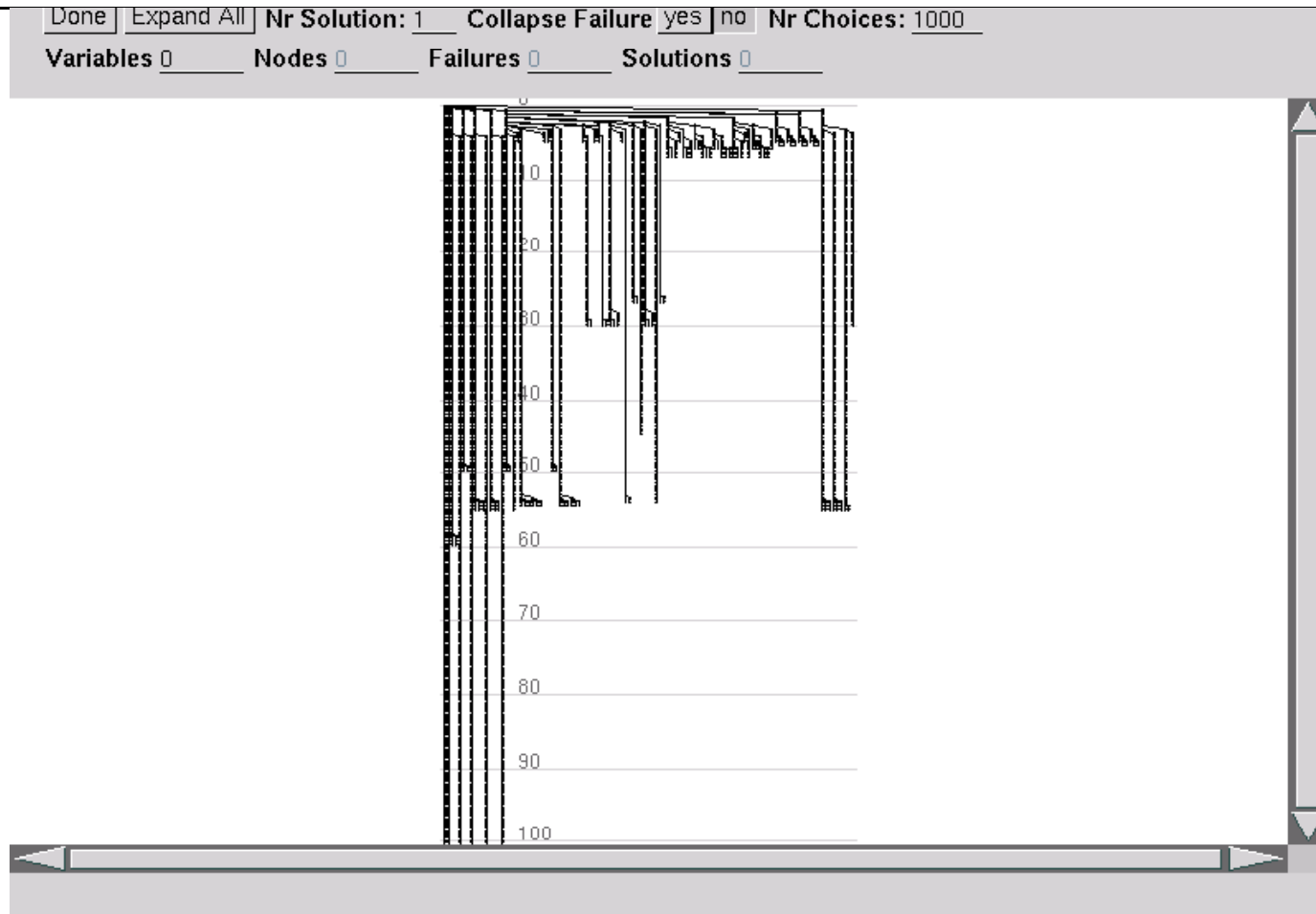


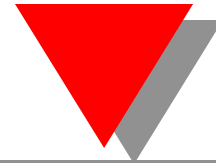
Introducing partial search (D-F)

- ◆ Instead of chronological backtracking, try partial search together with different strategies
- ◆ Try to explore more of search tree
- ◆ Possible to loose optimal solution
- ◆ Parameters
 - Credit = N
 - Extra choices = 5

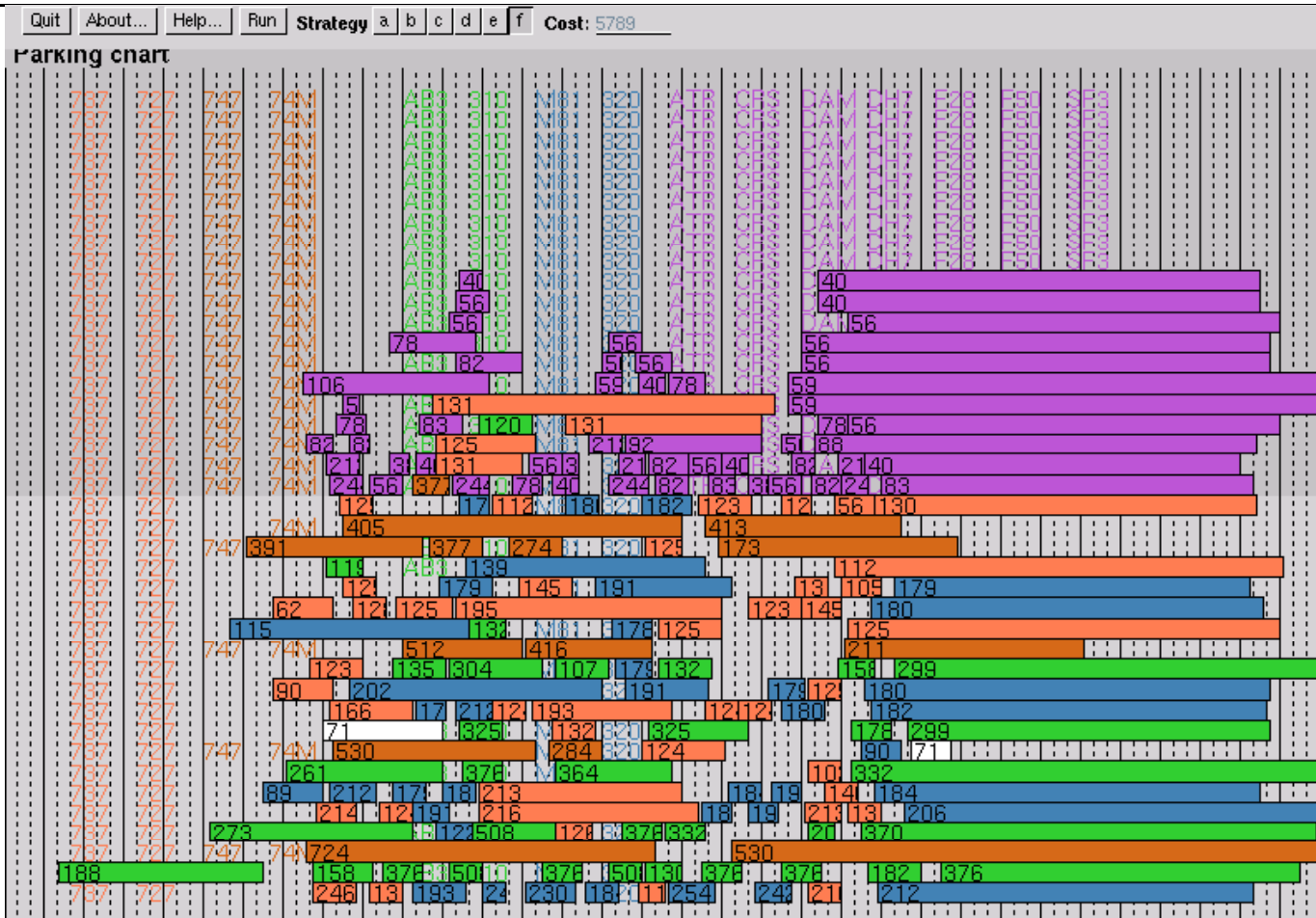


Search tree display (Strategy F)





Best solution strategy F



Results (1000 backtrack)

Strategy	First sol	2 nd sol					Best sol
A	6664	6539					6539
B	6417	6302					6302
C	6417	6302					6302
D	6664	6656	6542	6539			6539
E	6417	6302	6028	5913	5904	5789	5789
F	6417	6302	6028	5013	5904	5789	5789

Lower bounds

- ◆ **Current lower bound trivial**
 - all flights which must be parked on apron
- ◆ **Enumeration for optimal solution impossible**
- ◆ **Individual lower bound based on cliques**
 - all flights which compete for the same parking at the same time
 - easy to find with cumulative constraints
 - if clique bigger than number of stands, some flights must be on apron
- ◆ **Difficult to combine lower bounds**
 - common flights in cliques
- ◆ **Requires custom programming**

Rescheduling

- ◆ Requirement to resolve solution when some data/constraint changes
- ◆ Work in progress, part of solution is already fixed
- ◆ Two extremes
 - find best solution for new problem, ignore previous solution
 - save as much as possible of previous solution, even if cost increases
- ◆ Rescheduling time measured in seconds

Example 2: Resource restricted scheduling

- ◆ **Scheduling a project under resource constraints**
- ◆ **Problem consists of**
 - **Set of tasks(fixed duration)**
 - **Set of precedence constraints between tasks**
 - ◆ task must finish before another can start
 - **Set of resource constraints**
 - ◆ cumulative resources
 - ◆ each task may use multiple resources in different quantities
- ◆ **Real-life situation:**
 - **construction projects**
 - ◆ resources are types of personnel, demand are groups of people required

Objective

- ◆ Objective find solution which minimizes C_{max} , overall end date

Model

◆ Variables

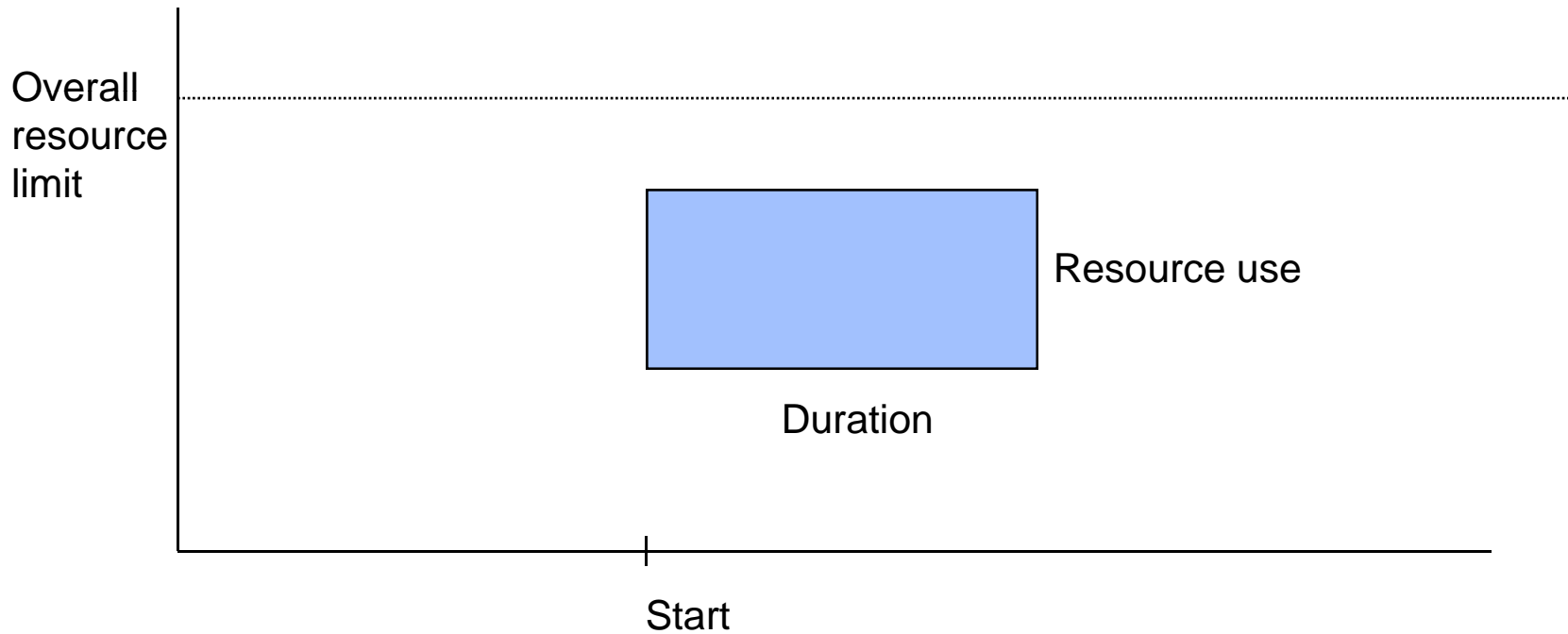
- Each task is represented by one domain variable, denoting the **Start**
- Duration, resource uses are fixed (integers)

Constraints

- ◆ **Version A**
 - precedence constraints handled by inequality
 - resource constraints expressed with cumulative
- ◆ **Version B**
 - as A
 - redundant precedence global constraint to exploit interaction between constraints
- ◆ **Version C**
 - as B
 - redundant disjunctive cumulative constraints on cliques of disjunctive tasks
 - cliques found by precedence

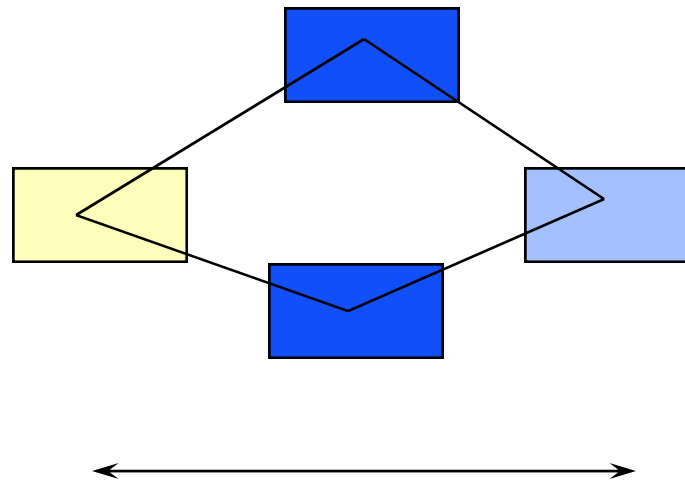
Cumulative resource constraint

For each machine type:



Redundant precedence constraint

- ◆ Interaction of inequality and resource limits
- ◆ Interaction of multiple resource constraints
- ◆ Detection of disjunctive cliques

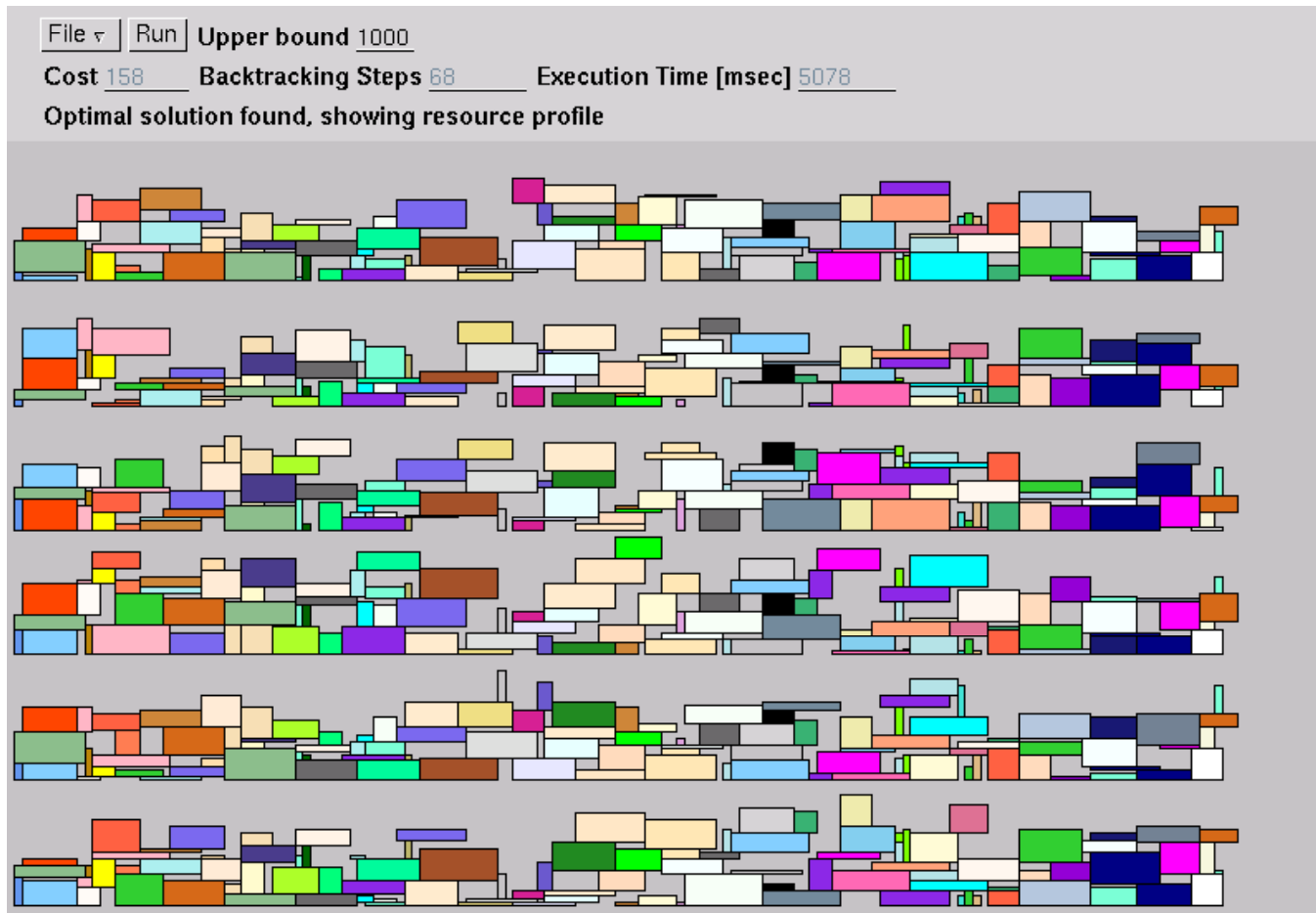


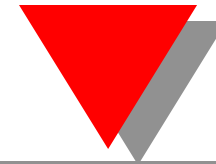
Data

- ◆ **Alvarez data set (version 2)**
 - only looking at large test cases
- ◆ **48 problems of 103 tasks each**
 - including start and end tasks
- ◆ **Generated by program**
 - not real-life data
 - some rather simple
 - some extremely hard
- ◆ **Not all optimal solutions known**

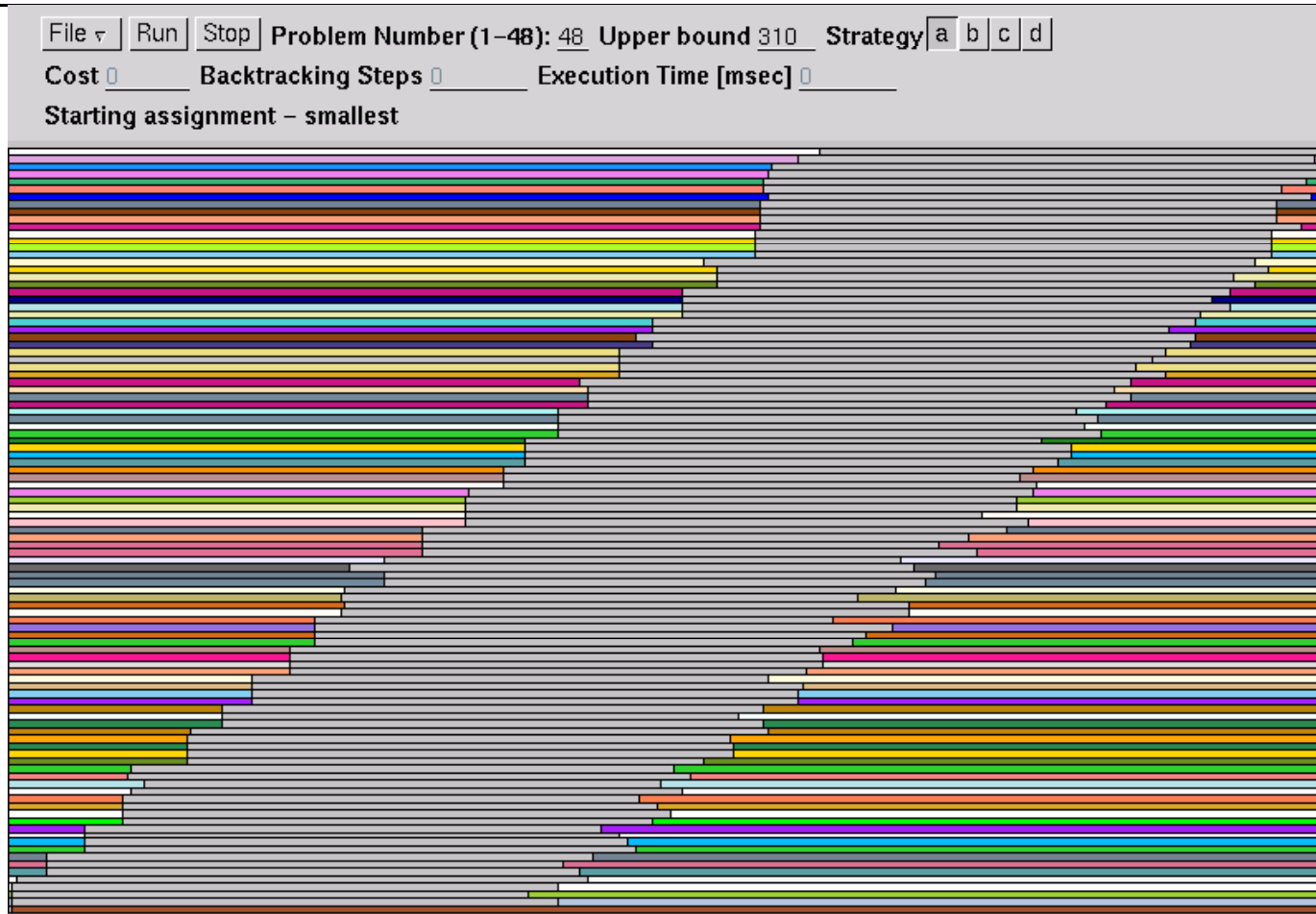


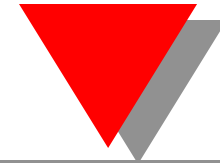
Solution (problem 1)



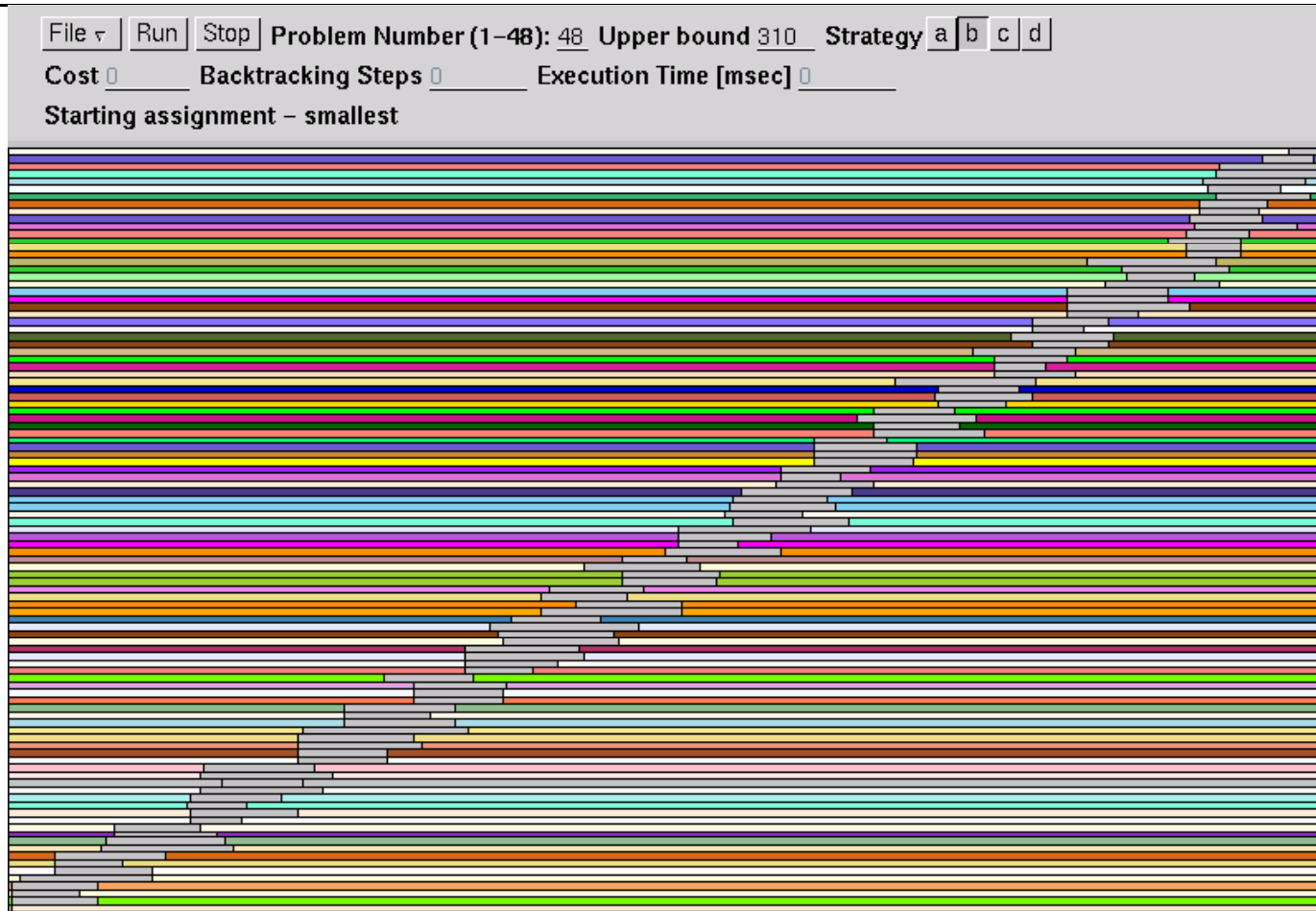


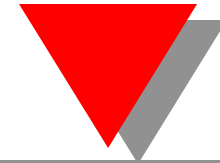
Initial propagation (Model A, problem 48)



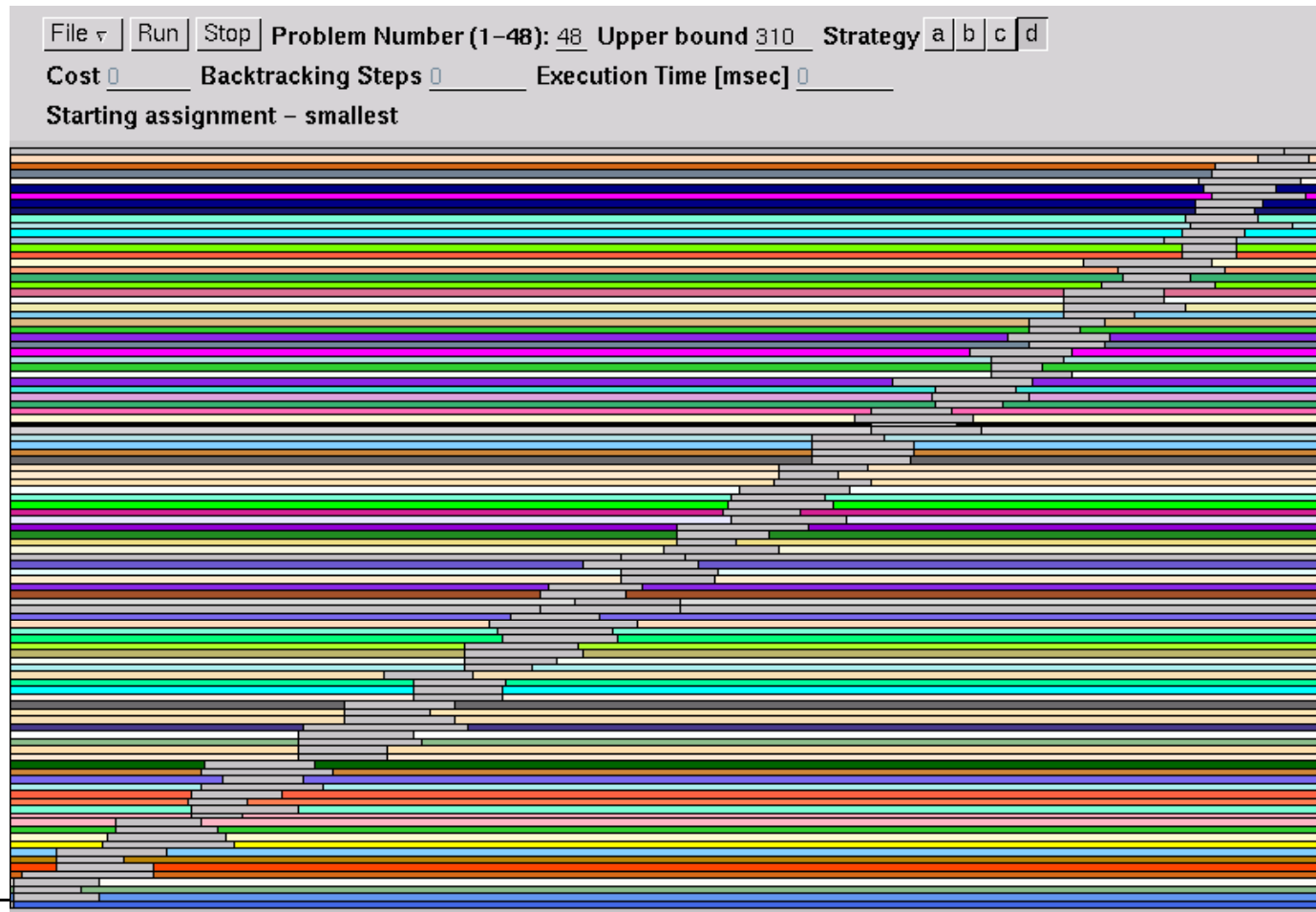


Initial propagation (Model B, problem 48)





Initial propagation (Model C, problem 48)



Search strategy “Smallest”

- ◆ **Variable selection**
 - smallest value in domain
- ◆ **Value selection**
 - smallest value in domain
- ◆ **Method finds solutions in every instance**
 - assign tasks left to right
- ◆ **Initial solution quite good**

Search strategy “Most constrained”

- ◆ **Variable selection**
 - smallest domain, occurs in most constraints
- ◆ **Value selection**
 - smallest value in domain
- ◆ **Method does not find solutions in all cases**
 - too eager to fix small value in middle of schedule
 - constraints not strong enough to detect failure
- ◆ **Good second stage search procedure**
 - start with upper bound from smallest procedure

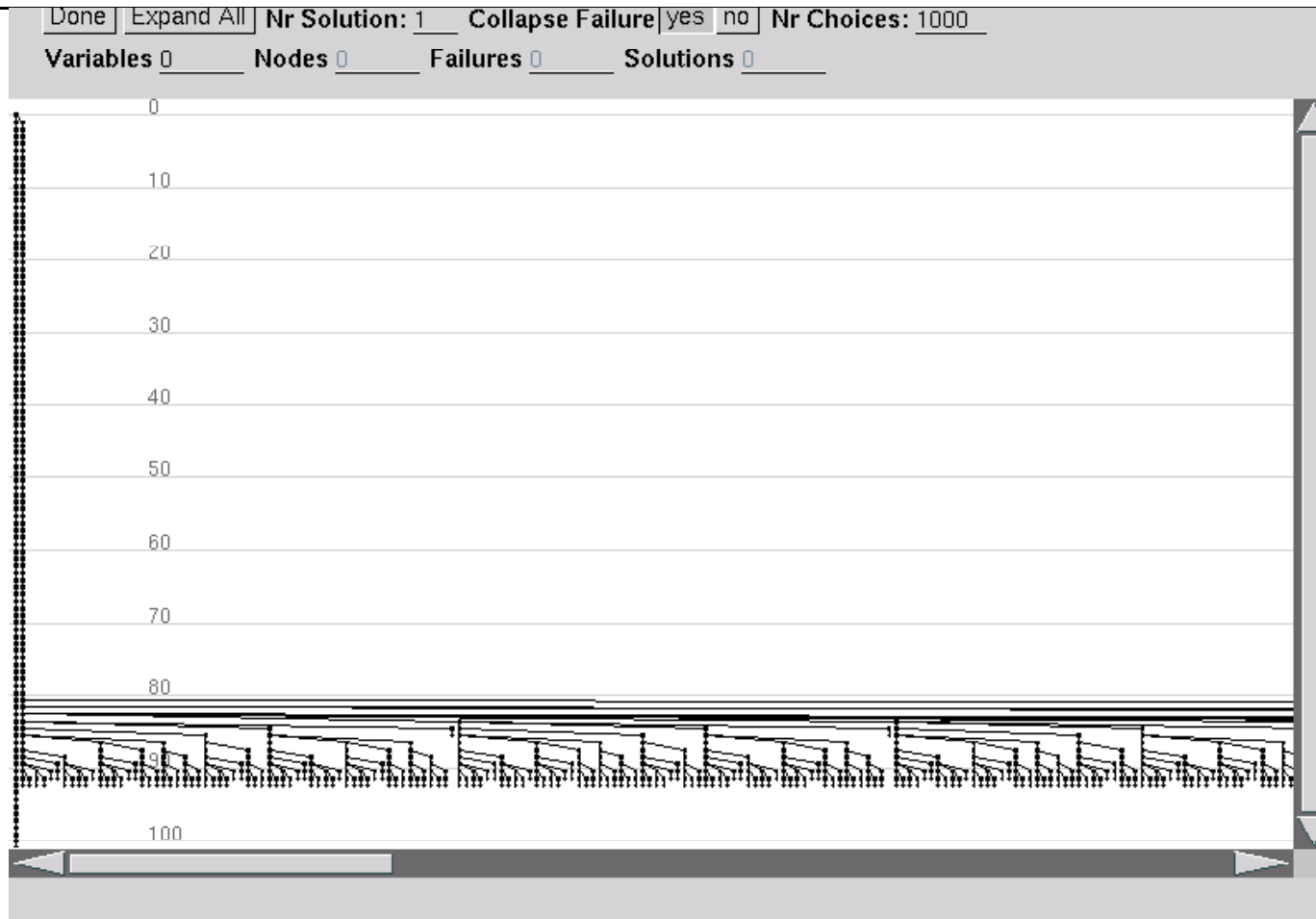
Partial search

- ◆ Combined with smallest or most constrained strategy
- ◆ Improves search behavior in some cases
- ◆ Gives up too easily for other examples

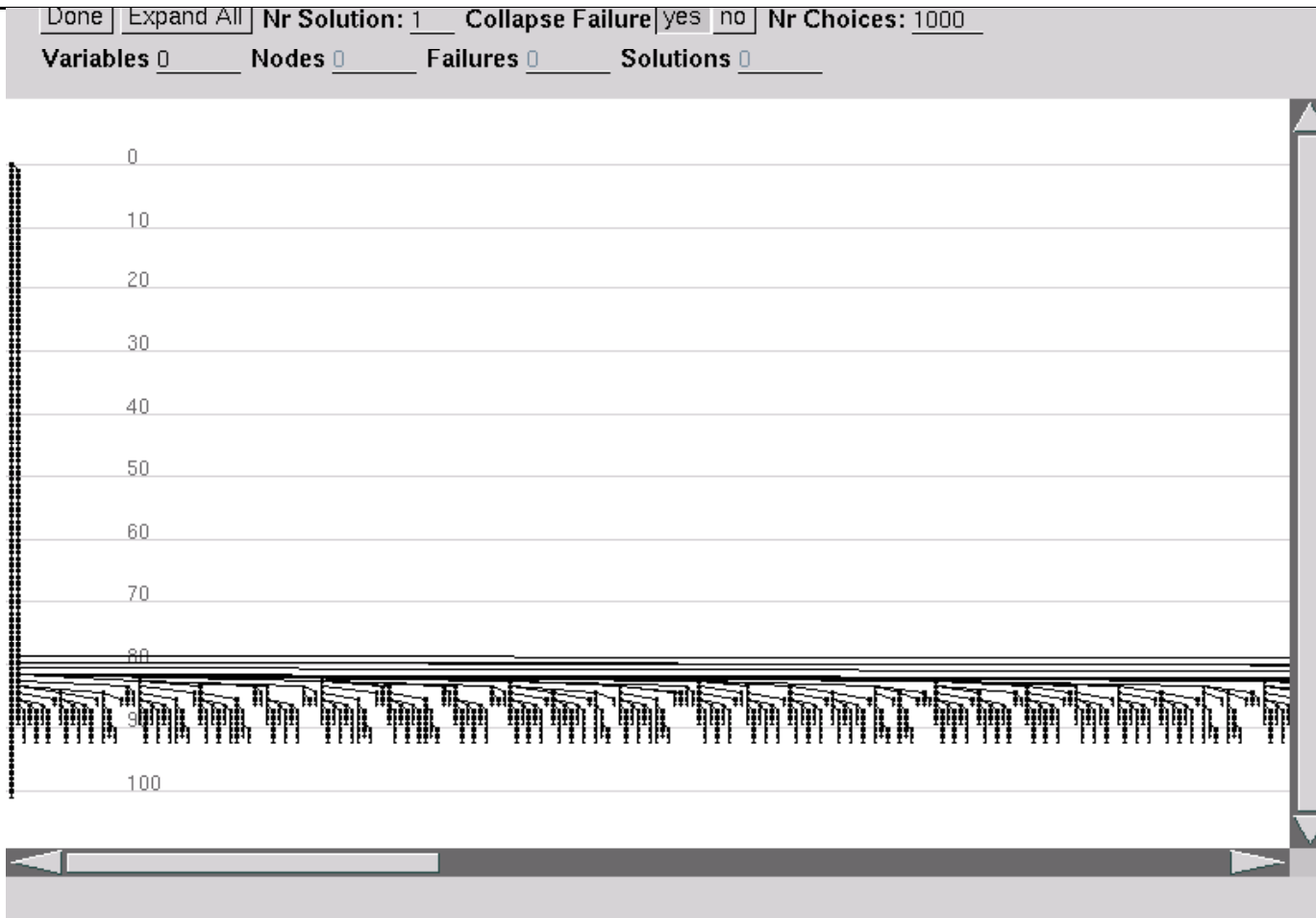
Results

- ◆ For time out 30s or 100s
- ◆ For each of 48 problems
- ◆ For each of the models A,B,C and C with partial search
- ◆ lower bound, solution with smallest, solution with second stage most constrained
- ◆ Also best result obtained with CHIP

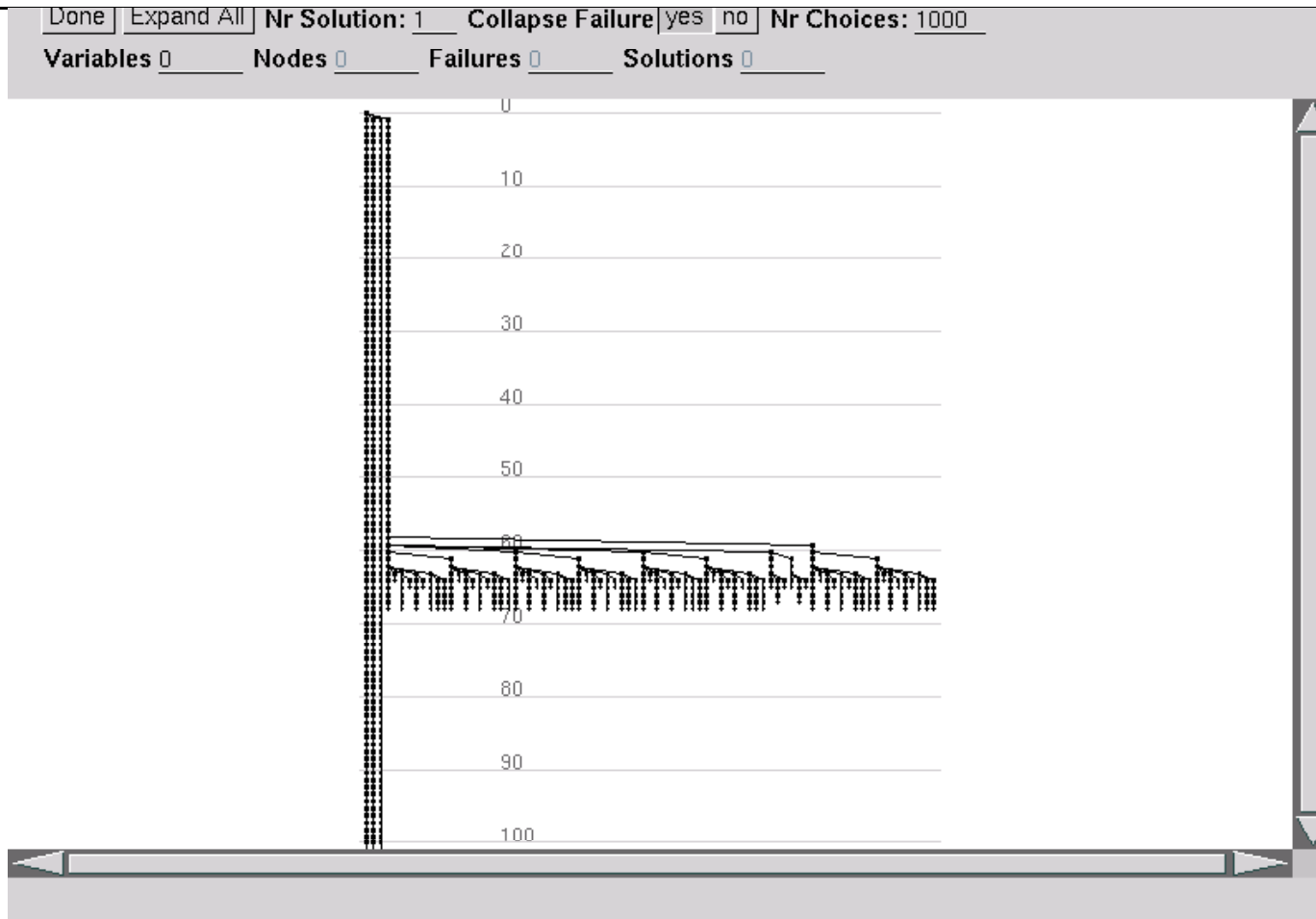
Search tree Model A, problem 48



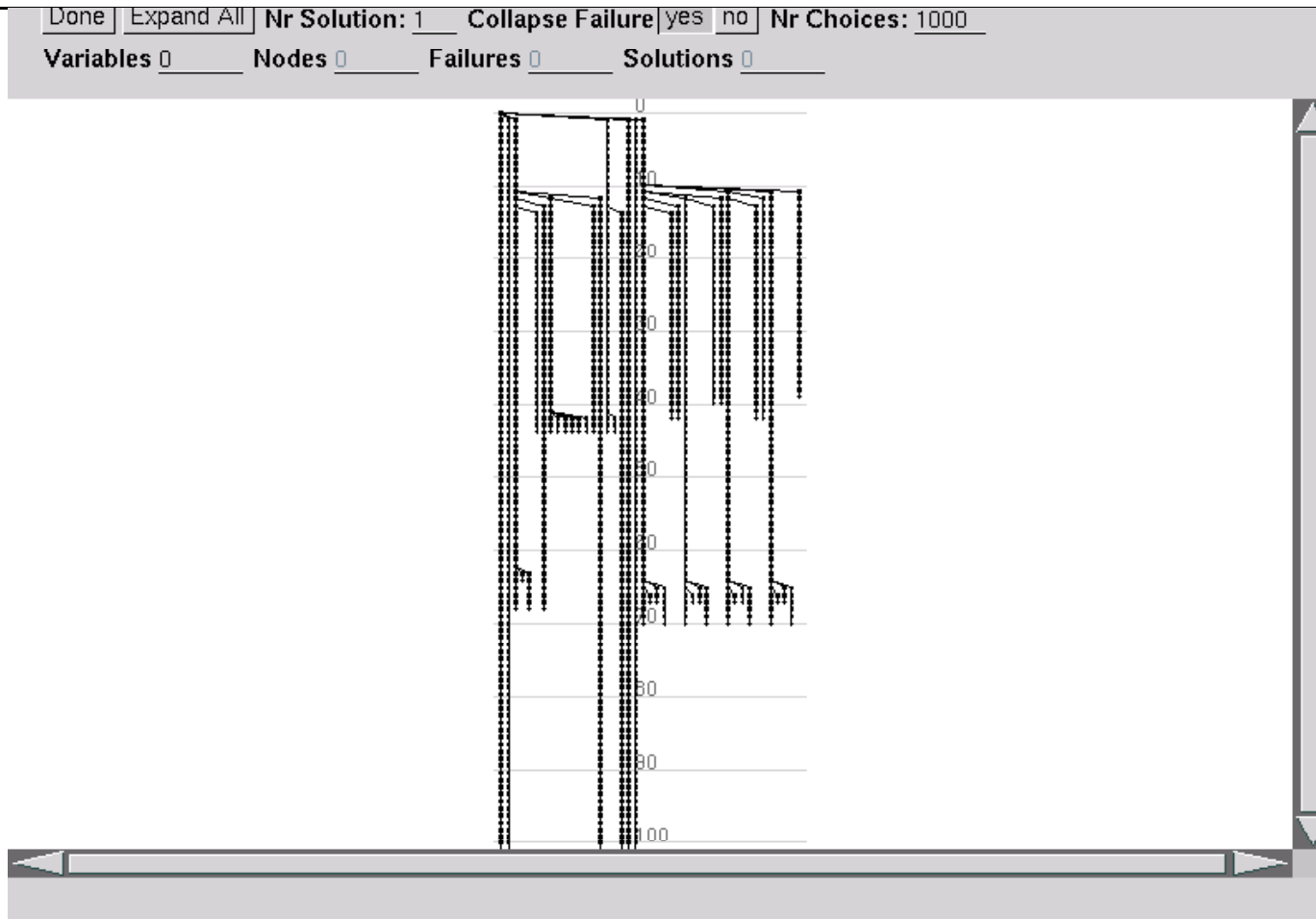
Search tree Model B, problem 48



Search tree Model C, problem 48



Search tree Model C + partial search, problem 48



Alvarez results 30s timeout, part 1

Nr	cumulative + ineq			precedence			precedence + cliques			+ partial search			Best
	Low	small	most	low	small	most	low	small	most	low	small	most	
1	158	166	158	158	168	158	158	168	158	158	162	158	158
2	167	208	-	167	208	-	167	208	-	167	213	193	187
3	188	282	-	188	296	-	188	281	-	188	286	-	265
4	264	469	-	399	470	-	402	470	-	402	469	-	441
5	170	252	-	170	252	223	170	258	217	170	256	217	192
6	159	293	-	159	294	271	159	301	-	159	316	270	212
7	211	374	-	211	360	-	238	354	-	238	360	-	292
8	189	356	-	223	363	-	232	357	-	232	338	-	295
9	139	139	139	139	139	139	139	139	139	139	139	139	139
10	119	119	119	119	119	119	119	119	119	119	119	119	119
11	144	204	-	144	200	184	144	197	176	144	198	178	172
12	171	318	-	274	317	-	274	312	-	274	312	-	294
13	113	144	-	114	149	132	114	147	130	114	142	128	127
14	114	188	-	118	190	-	118	190	-	119	199	-	156
15	146	233	-	146	226	-	146	238	-	146	238	-	184
16	145	212	-	153	219	-	153	210	-	153	211	-	181
17	208	209	209	208	209	209	208	209	209	208	209	209	209
18	204	250	237	207	244	-	207	244	-	207	245	-	232
19	207	349	-	247	316	-	247	326	-	247	330	-	306
20	270	480	-	464	498	-	464	495	-	464	495	-	478
21	226	290	-	247	294	-	247	290	-	247	290	-	276
22	223	314	-	278	319	-	278	319	-	278	319	-	296
23	202	391	-	341	391	-	341	377	-	341	378	-	372
24	251	466	-	427	474	-	427	469	-	427	475	-	452

Alvarez results 30s timeout, part 2

25	177	177	177	177	177	177	177	177	177	177	177	177	177
26	183	183	183	183	183	183	183	183	183	183	183	183	183
27	182	206	201	190	209	206	191	209	204	191	213	204	199
28	190	303	-	290	308	-	290	303	-	290	304	-	295
29	205	237	-	221	230	-	221	230	-	221	230	-	225
30	190	246	-	228	246	-	228	246	-	228	247	-	231
31	183	246	-	220	248	-	220	243	-	220	242	-	227
32	212	294	-	267	297	-	267	296	-	267	296	-	289
33	215	220	220	215	220	220	215	220	220	215	220	220	220
34	218	277	-	227	278	-	227	278	-	227	275	-	264
35	229	381	-	326	373	-	326	361	-	326	362	-	341
36	332	575	-	575	576	-	575	576	575	575	575	575	575
37	222	328	-	321	327	327	321	328	327	321	328	327	327
38	228	421	-	370	418	-	370	418	-	370	418	-	376
39	225	427	-	382	429	-	382	409	-	382	408	-	389
40	249	473	-	448	473	-	448	473	-	448	473	-	451
41	191	191	191	191	191	191	191	191	191	191	191	191	191
42	186	190	187	186	189	187	186	189	187	186	187	187	187
43	209	277	260	259	268	266	259	268	266	259	275	271	260
44	227	380	-	375	380	-	375	380	-	375	380	-	375
45	189	221	-	216	222	216	216	224	216	216	222	216	216
46	202	263	253	246	264	258	246	263	263	246	264	258	251
47	193	265	-	261	267	-	261	267	-	261	267	-	262
48	193	307	-	300	302	-	300	300	300	300	301	300	300

Alvarez results 100s timeout, part 1

Nr	cumulative + ineq			precedence			precedence + cliques			+ partial search			Best
	low	small	most	low	small	most	low	small	most	low	small	most	
1	158	166	158	158	168	158	158	168	158	158	160	158	158
2	167	208	-	167	208	-	167	208	-	167	213	193	187
3	188	282	-	188	296	-	188	281	-	188	286	-	265
4	264	469	-	399	470	-	402	470	-	402	469	-	441
5	170	252	-	170	252	223	170	258	217	170	254	217	192
6	159	293	-	159	294	271	159	296	-	159	304	270	212
7	211	374	-	211	360	-	238	354	-	238	353	-	292
8	189	356	-	223	361	-	232	355	-	232	336	-	295
9	139	139	139	139	139	139	139	139	139	139	139	139	139
10	119	119	119	119	119	119	119	119	119	119	119	119	119
11	144	204	-	144	198	187	144	197	176	144	195	176	172
12	171	318	-	274	317	-	274	312	-	274	312	-	294
13	113	144	-	114	149	131	114	147	130	114	142	128	127
14	114	188	-	118	190	-	118	190	-	118	194	-	156
15	146	233	-	146	226	-	146	238	-	146	233	-	184
16	145	212	-	153	218	-	153	209	-	153	211	-	181
17	208	209	209	208	209	209	208	209	209	208	209	209	209
18	204	247	237	207	244	-	207	244	-	207	245	-	232
19	207	347	-	247	316	-	247	326	-	247	330	-	306
20	270	480	-	464	498	-	464	495	-	464	495	-	478
21	226	290	-	247	294	292	247	290	288	247	290	288	276
22	223	314	-	278	319	-	278	319	-	278	319	-	296
23	202	391	-	341	391	-	341	377	-	341	378	-	372
24	251	458	-	427	466	-	427	462	-	427	474	-	452

Alvarez results 100s timeout, part 2

25	177	177	177	177	177	177	177	177	177	177	177	177	177
26	183	183	183	183	183	183	183	183	183	183	183	183	183
27	182	206	201	190	209	206	191	208	204	191	213	204	199
28	190	303	-	290	308	-	290	303	-	290	304	-	295
29	205	237	-	221	230	-	221	230	-	221	230	-	225
30	190	246	-	228	246	-	228	246	-	228	247	-	231
31	183	246	-	220	244	-	220	239	-	220	239	238	227
32	212	294	-	267	297	-	267	296	-	267	296	-	289
33	215	220	220	215	220	220	215	220	220	215	220	220	220
34	218	277	275	227	273	-	227	273	-	227	275	-	264
35	229	381	-	326	370	-	326	361	-	326	362	-	341
36	332	575	-	575	576	-	575	576	575	575	575	575	575
37	222	328	-	321	327	327	321	328	327	321	328	327	327
38	228	421	-	370	418	-	370	418	-	370	418	-	376
39	225	427	-	382	429	-	382	409	-	382	408	-	389
40	249	473	-	448	473	-	448	473	-	448	473	-	451
41	191	191	191	191	191	191	191	191	191	191	191	191	191
42	186	190	187	186	189	187	186	189	187	186	189	187	187
43	209	277	260	259	267	-	259	268	266	259	275	271	260
44	227	380	-	375	380	-	375	380	-	375	380	-	375
45	189	221	-	216	222	216	216	224	216	216	217	216	216
46	202	263	253	246	264	258	246	263	263	246	264	258	251
47	193	265	-	261	267	-	261	266	-	261	267	-	262
48	193	307	-	300	302	-	300	300	300	300	300	300	300

Evaluation

- ◆ **Improvements in constraints pay off**
 - most of the time
- ◆ **Better lower bounds by interaction of precedence and resource constraints**
 - sometimes dramatic improvement
- ◆ **Not a single best strategy**
 - combination of methods quite powerful
- ◆ **Partial search helpful to find good solutions**
- ◆ **Proof of optimality only by reaching lower bound**

Improvements (to obtain best results)

- ◆ **Better lower bounds**
 - custom programming
 - more detailed analysis
- ◆ **More and more complex strategies**
 - interval labeling
- ◆ **More time to search**
- ◆ **CHIP competitive with best custom programs**

Example 3: Nurse scheduling

- ◆ In hospitals, nurses work in three shift system
- ◆ Each nurse works at most one shift per day
 - M (morning), A (afternoon) and N (night) shift
- ◆ R (rest) periods must be allocated equally to each nurse
- ◆ The demand on each day for each shift is variable, but known
 - depends on number of patients, procedures
- ◆ The work assignment must follow a set of rules
 - many rules are strict, others are preferences
- ◆ A solution consists in an assignment for each nurse on each day to a value M,A,N,R which satisfies all rules
- ◆ Rule sets are changing over time

Objective

- ◆ Find a feasible solution
- ◆ Often, as well:
 - Balance work load of nurses
 - Do not balance work load of nurses
 - Satisfy many preferences if possible

Rule set of example

- ◆ In the scheduling period, there must be 4 days rest for each person
- ◆ In any sequence of 6 days, there must be atleast one rest day
- ◆ In any sequence of 5 days, there may be atmost 2 rest days
- ◆ In the scheduling period, a person is not allowed to work more than 4 night shifts
- ◆ It is not allowed to work 4 night shifts in a row
- ◆ A day working night shift may not be followed by a day working day shift
- ◆ It is not allowed to work night shifts, switch one day to a day shift and then work nightshift again
- ◆ There can be no bridge days, one day of work in between two days off
- ◆ The demand of service required must be satisfied for each day

Model

◆ Variables

- Variables describe the work of each person on each day
- Four possible values
 - ◆ 0 Rest
 - ◆ 1 Morning
 - ◆ 2 Afternoon
 - ◆ 3 Night
- Variables arranged in array of lines and columns
 - ◆ Lines = work for one person
 - ◆ Columns = work on one day

Model

◆ Constraints on day

- Restrict the number of times a value is chosen on a day
- $\text{atmost}(N, [X1, X2..Xn], [0, 0, ..0], [V], \text{all})$
 - ◆ The value V is taken N times among the variables $X1..Xn$
 - ◆ N fixed in this case by demand profile
 - ◆ Propagation event all: propagate whenever possible
- Redundant among constraints
 - ◆ combinations of work pattern
 - ◆ rest days required (for perfect problems)

◆ Constraints on persons

- $\text{among}(1, 6, 6, 4, 4, [X1, X2..Xk], [0, 0..0], [0], \text{all}),$
 - ◆ in any sequence of 6 days, atleast 1 and atmost 6 days off
 - ◆ atleast and atmost four days off altogether
- $\text{among}(0, 2, 5, 4, 4, [X1, X2..Xk], [0, 0..0], [0], \text{all}),$
 - ◆ in any sequence of 5 days, atleast 0 and atmost 2 days off

Model

◆ Constraints on person (cont'd)

- among(0,3,4,0,4,[X1,X2..k],[0,0..0],[3],all)
 - ◆ in any sequence of four days, atmost 3 night shifts
 - ◆ altogether atmost 4 night shifts
- sequence([0,0,2],[X1,X2..k],[0,0..0],[[sum,1,#=[3]],[sum,1,#=[1]]],all)
 - ◆ in any sequence of two consecutive variables, there must atleast 0 and atmost 0 (never) be a pattern which starts with one variable having value 3 (= Night) followed by one variable having value 1
- sequence([0,0,3],[X1,X2..k],[0,0..0],[[sum,1,#=[3]],[sum,1,#=[1,2]],[sum,1,#=[3]]],[[sum,1,#=[0]],[sum,1,#=[1,2,3]],[sum,1,#=[0]]],all)

in any sequence of three variables there can not be a pattern of either the form

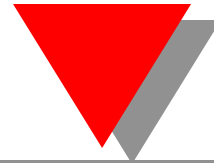
N[MA]N

nor of the form

R[MAN]R

Data

- ◆ 3 data sets of 100 randomly generated demand profiles
- ◆ 10 persons/14 days
- ◆ perfect problems, no spare capacity
- ◆ Set X: demand per day 7..8 persons
 - demand per shift 1..4
 - one problem unsolvable
- ◆ Set Y: demand per day 6..9 persons
 - demand per shift 1..4
 - one problem unsolvable
- ◆ Set Z: demand per day 5..10 persons
 - demand per shift 1..4
 - atleast 4 problems unsolvable



Typical solution (problem 16, data set X)

Problem 16 Cost 0 Backtracking Steps 298 Execution Time [msec] 3225

N	R	N	A	R	N	N	R	M	M	M	R	M	A
N	R	N	A	R	N	A	M	N	R	R	M	M	M
A	R	A	N	N	R	N	A	R	A	M	A	N	R
M	M	A	N	R	R	M	A	N	R	A	A	A	R
M	N	R	N	A	R	M	A	R	N	A	R	A	A
M	N	R	M	N	N	N	R	R	M	M	M	A	R
M	A	R	M	N	N	N	R	N	A	M	R	R	M
R	M	M	R	M	A	R	N	A	M	N	A	R	A
R	M	A	R	M	A	R	N	A	A	R	A	N	A
R	M	A	R	M	M	R	N	A	A	R	N	N	N

4	4	1	2	3	1	2	1	1	3	4	2	2	2
1	1	4	2	1	2	1	3	3	4	2	4	3	4
2	2	2	3	3	4	4	3	3	1	1	1	3	1

Search strategies

- ◆ **Combination of several concepts**
 - variable selection
 - value selection
 - search method
- ◆ **Systematic analysis of combinations**
- ◆ **No custom strategy used**
 - depending on demand
 - depending on already assigned values

Variable selection

- ◆ **static by line**
 - top to bottom, left to right
- ◆ **static by column**
 - left to right, top to bottom
- ◆ **most constrained**
 - variables sorted by column
- ◆ **static order by demand and most constrained**
 - sort variable columns by decreasing demand
 - use most constrained on this list of variables
 - labeling starts in “difficult” part

Value selection

- ◆ **Only 4 possible values**
- ◆ **Perfect problems**
- ◆ **Static order**
 - indomain
- ◆ **Randomized order**
 - random starting point for indomain
 - values not independent
- ◆ **Domain splitting**
 - enforce a rest day (value 0) or a working day (values 1,2,3)
 - use of disequality inside labeling
 - default in some CLP systems

Search method

- ◆ **Full search**
 - limited by number of backtrack allowed (5000)
- ◆ **Partial search**
 - quadratic credit
 - 5 backtrack in local search
 - also limited by number of backtrack (5000)

Strategies

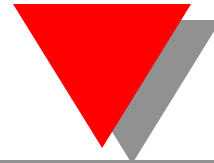
- ◆ **Static variable order, full search**
 - A: by line
 - B: by line, randomized
 - C: by column
 - D: by column, randomized
- ◆ **Dynamic variable order, full search**
 - E: most constrained
 - F: most constrained, randomized
 - G: most constrained on demand order
 - H: most constrained on demand order, randomized

Strategies 2

- ◆ **Dynamic variable order, partial search**
 - I: most constrained
 - J: most constrained, randomized
 - K: most constrained on demand order
 - L: most constrained on demand order, randomized
- ◆ **Static variable order, partial search**
 - M: by line
 - N: by line, randomized
 - O: by column
 - P: by column, randomized
- ◆ **Dynamic order, partial search, domain splitting**
 - Q: most constrained
 - R: most constrained on demand order

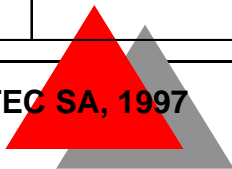
Evaluation

- ◆ Count the number of problems solved in each data set
- ◆ Randomized solver only run once
- ◆ All tests use the same number of choices (5000)



Success rate

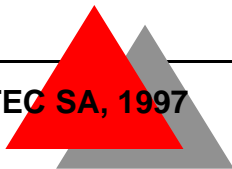
	Strategy	1000 backtracks			5000 backtracks		
		Data set 1 demand 7..8 per day	Data set 2 demand 6..9 per day	Data set 3 demand 5..10 per day	Data set 1 demand 7..8 per day	Data set 2 demand 6..9 per day	Data set 3 demand 5..10 per day
A	by line	0	0	0	0	0	0
B	by line, randomized	0	0	0	1	2	0
C	by column	9	16	11	9	18	13
D	by column, randomized	44	38	34	41	38	39
E	Most constrained	86	67	39	86	71	43
F	Most constrained, randomized	57	46	26	45	39	42
G	static sort, most constrained	57	46	39	63	53	41
H	static sort, most constrained, randomized	60	52	32	63	50	34
I	partial search, most constrained	98	90	69	99	97	83
J	partial search, most constrained, randomized	97	94	70	99	97	85
K	partial search, static sort, most constrained	98	90	63	99	96	77





Success rate, part 2

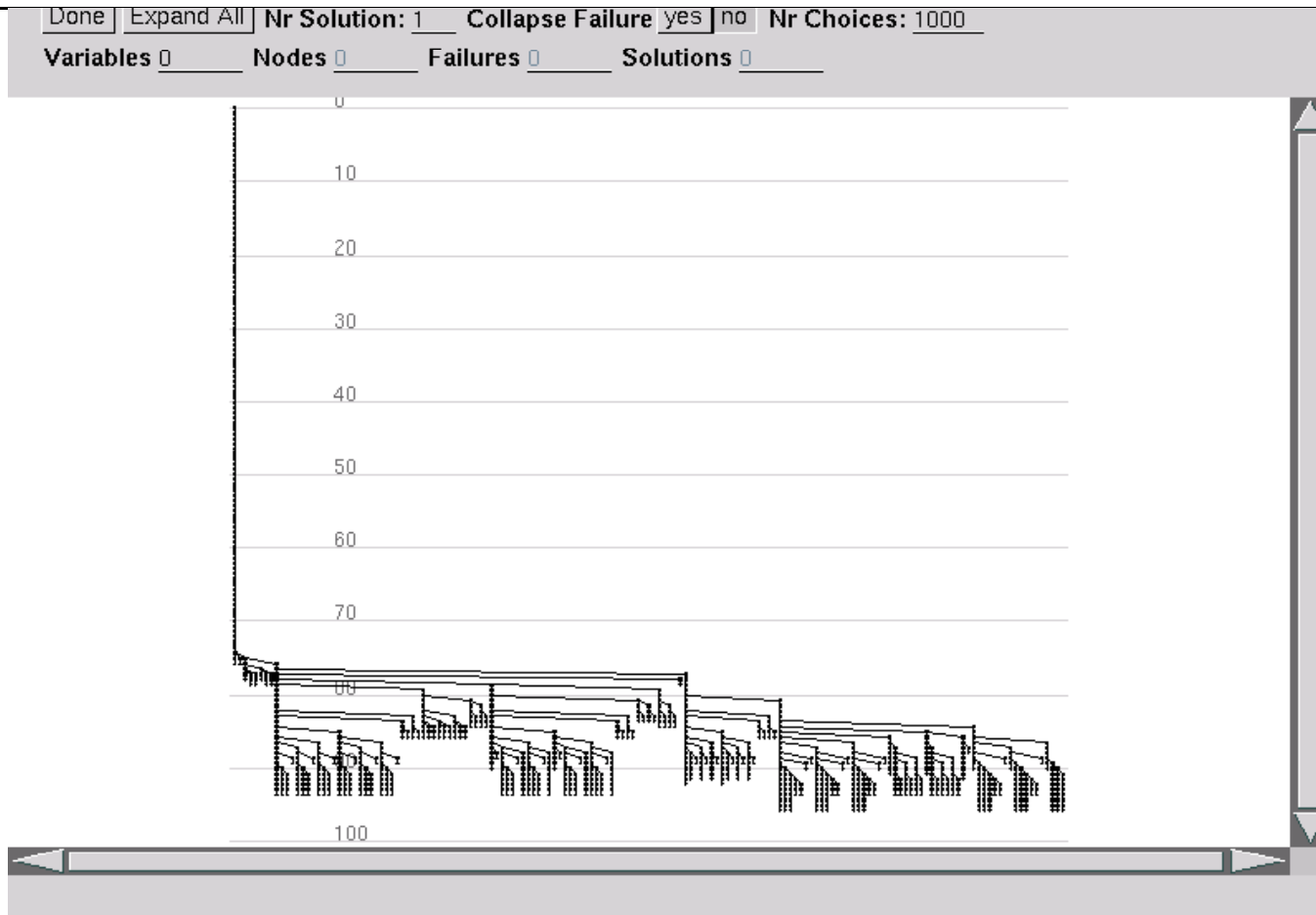
L	partial search, static sort, most constr, randomized	98	94	69	99	99	85
M	By line, partial search	0	0	0	0	0	0
N	By line, partial search, randomized	4	2	1	20	12	5
O	By column, partial search	13	13	12	21	18	13
P	By column, partial search, randomized	93	87	71	99	96	86
Q	Most constrained, partial search, R-MAN split	98	92	72	99	97	86
R	Most constrained, static order, partial search, R-MAN split	99	92	68	99	96	80



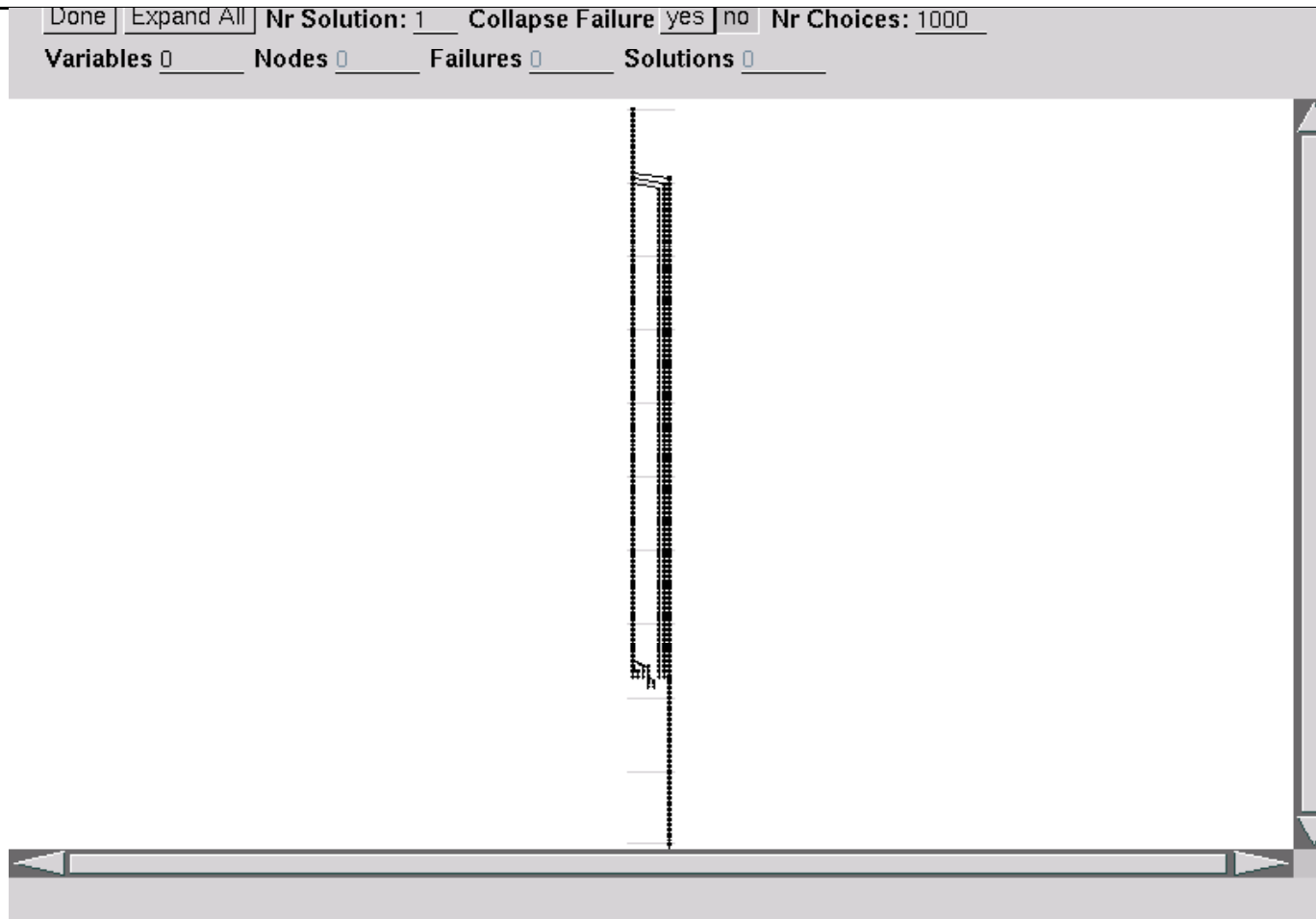
Search performance

- ◆ **Search tree analysis for full and partial search**
 - strategy E: most constrained
 - strategy I: most constrained and partial search
- ◆ **Plotting number of choices needed to find solutions**
 - x-axis: number of backtracks allowed
 - y-axis: number of problems solved
 - different plots for data sets X, Y and Z
 - shown for 1000 and 5000 backtrack horizon

Search tree strategy E, problem 16, data set X

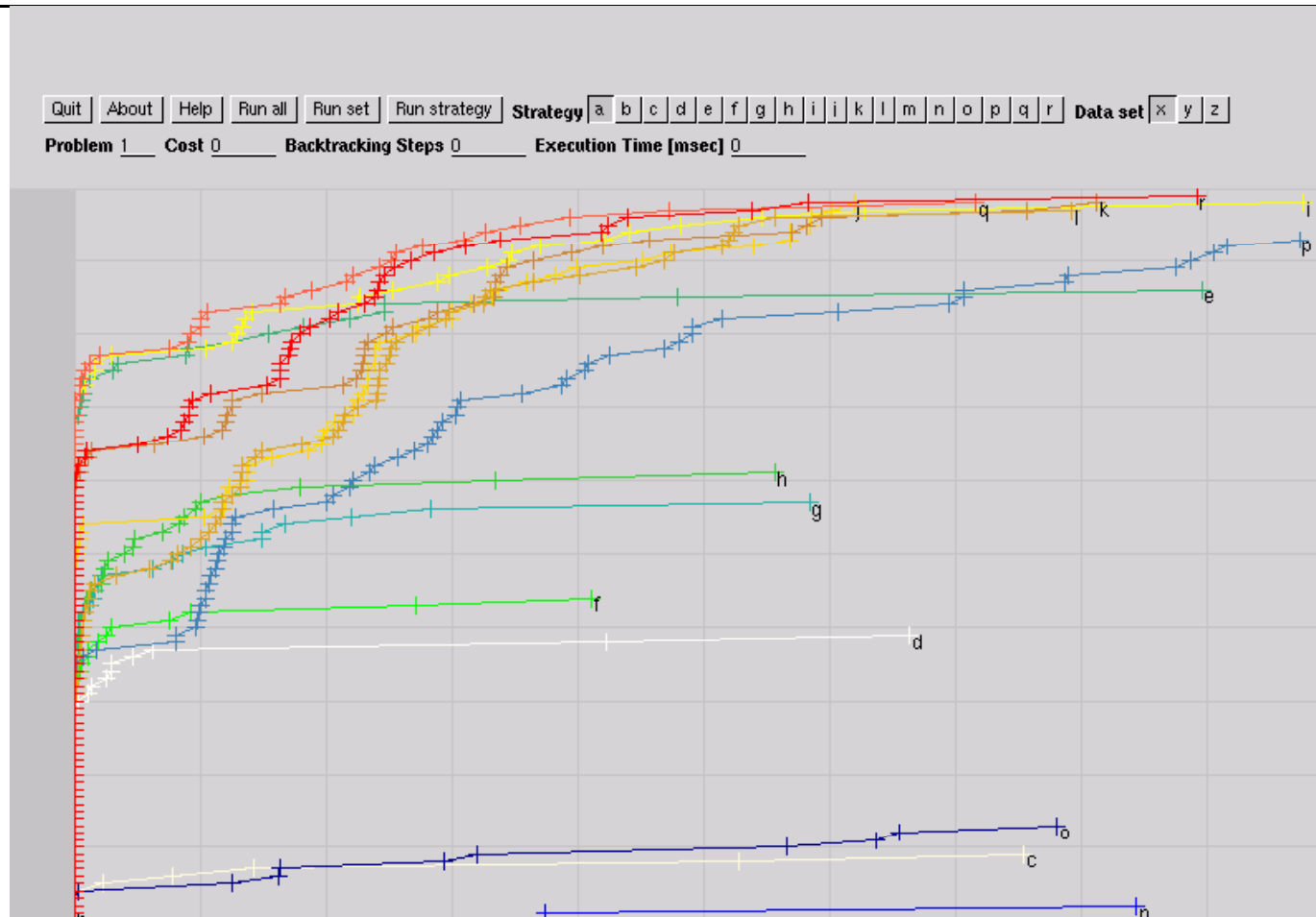


Search tree strategy I, problem 16, data set X



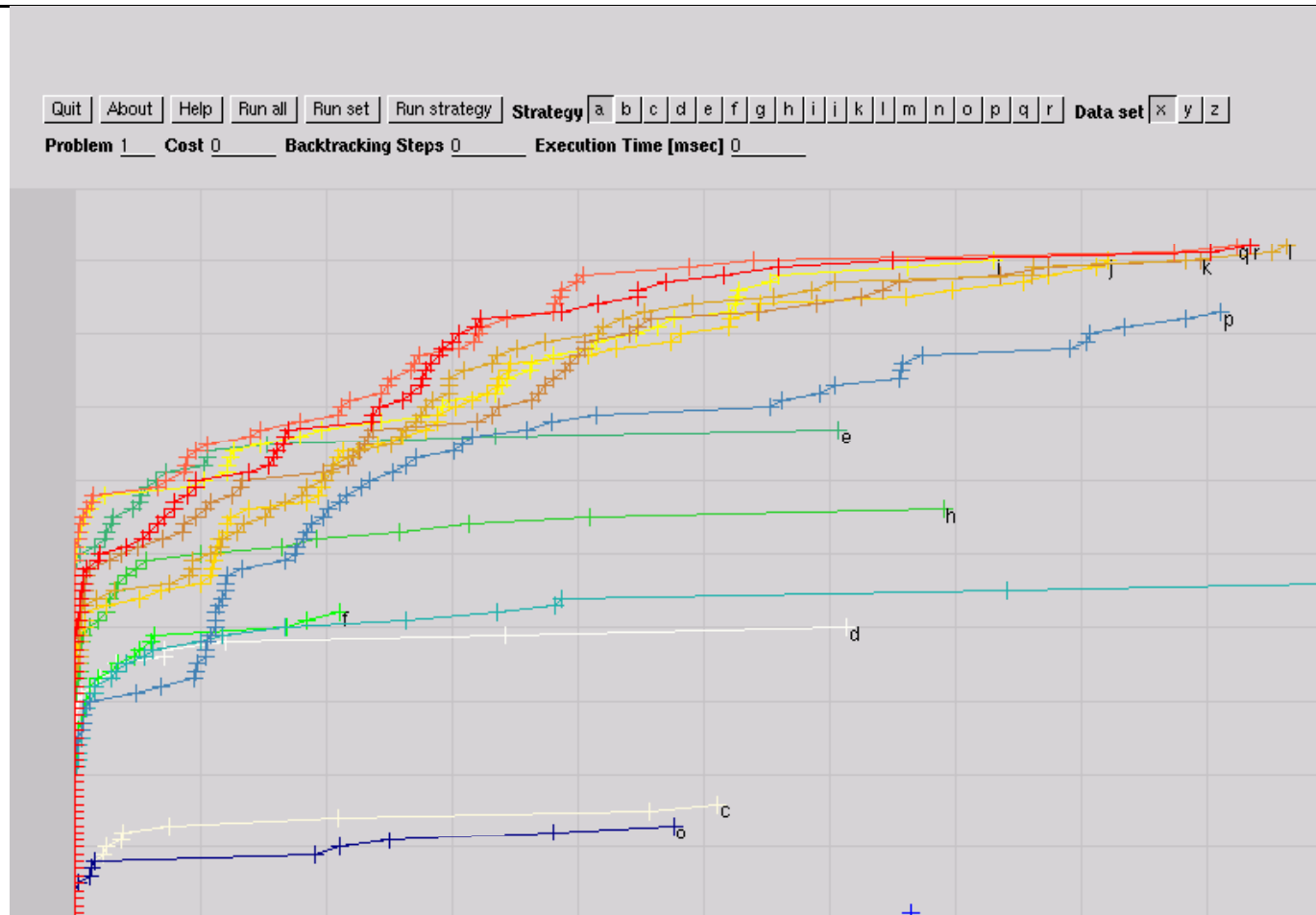


Data set X, backtrack plot strategies A-R, 1000 choices



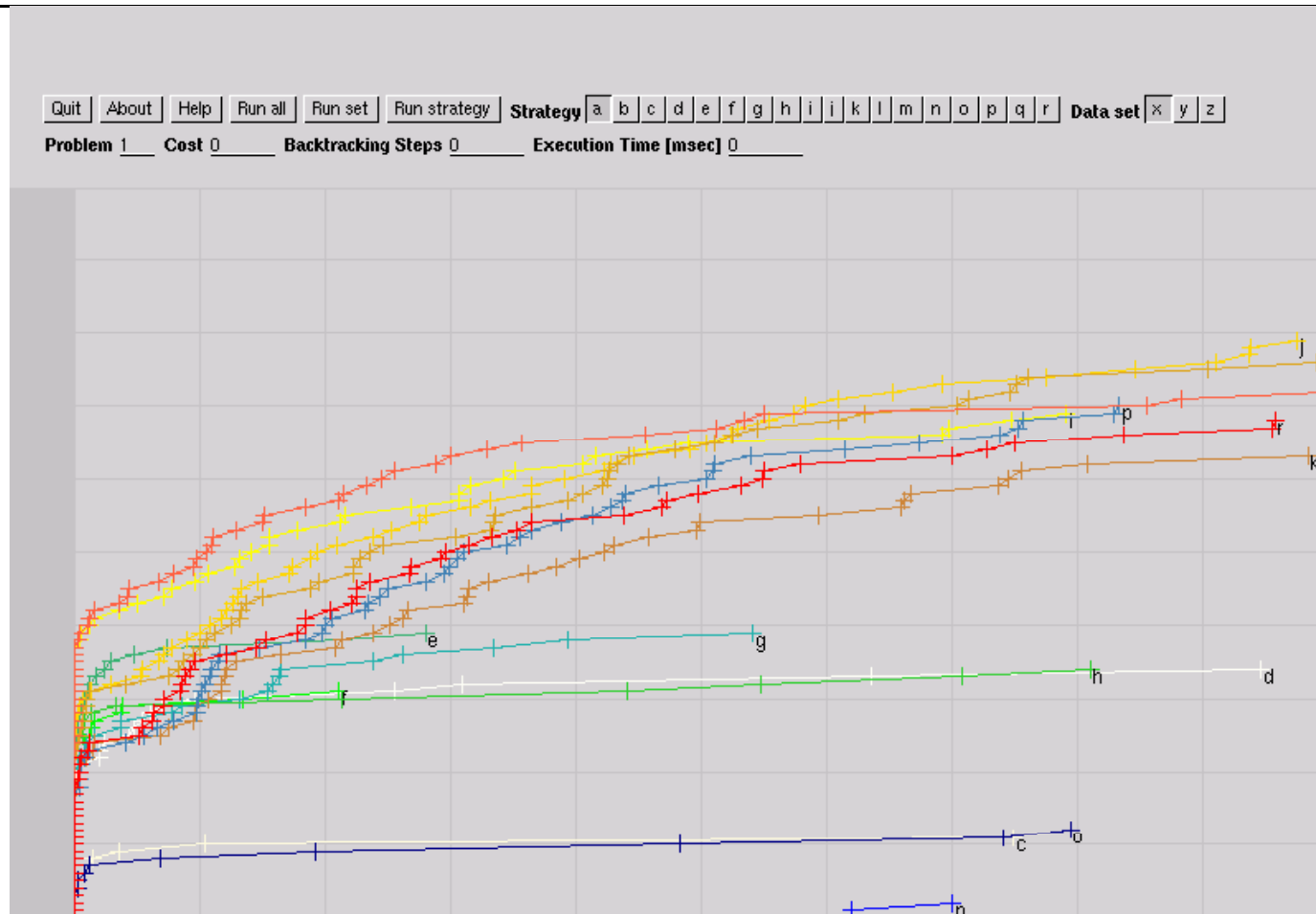


Data set Y, backtrack plot strategies A-R, 1000 choices



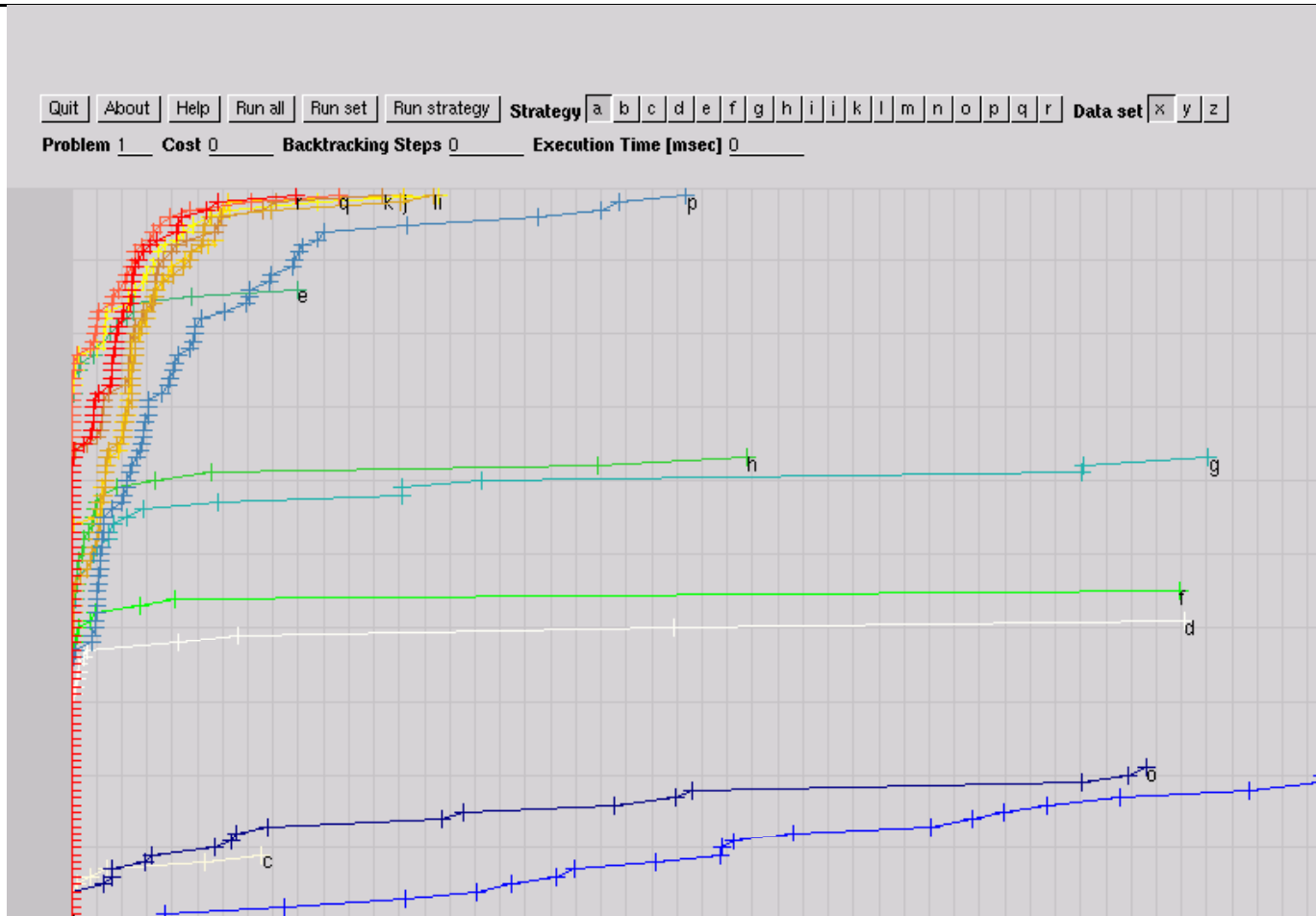


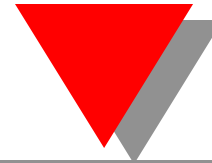
Data set Z, backtrack plot strategies A-R, 1000 choices



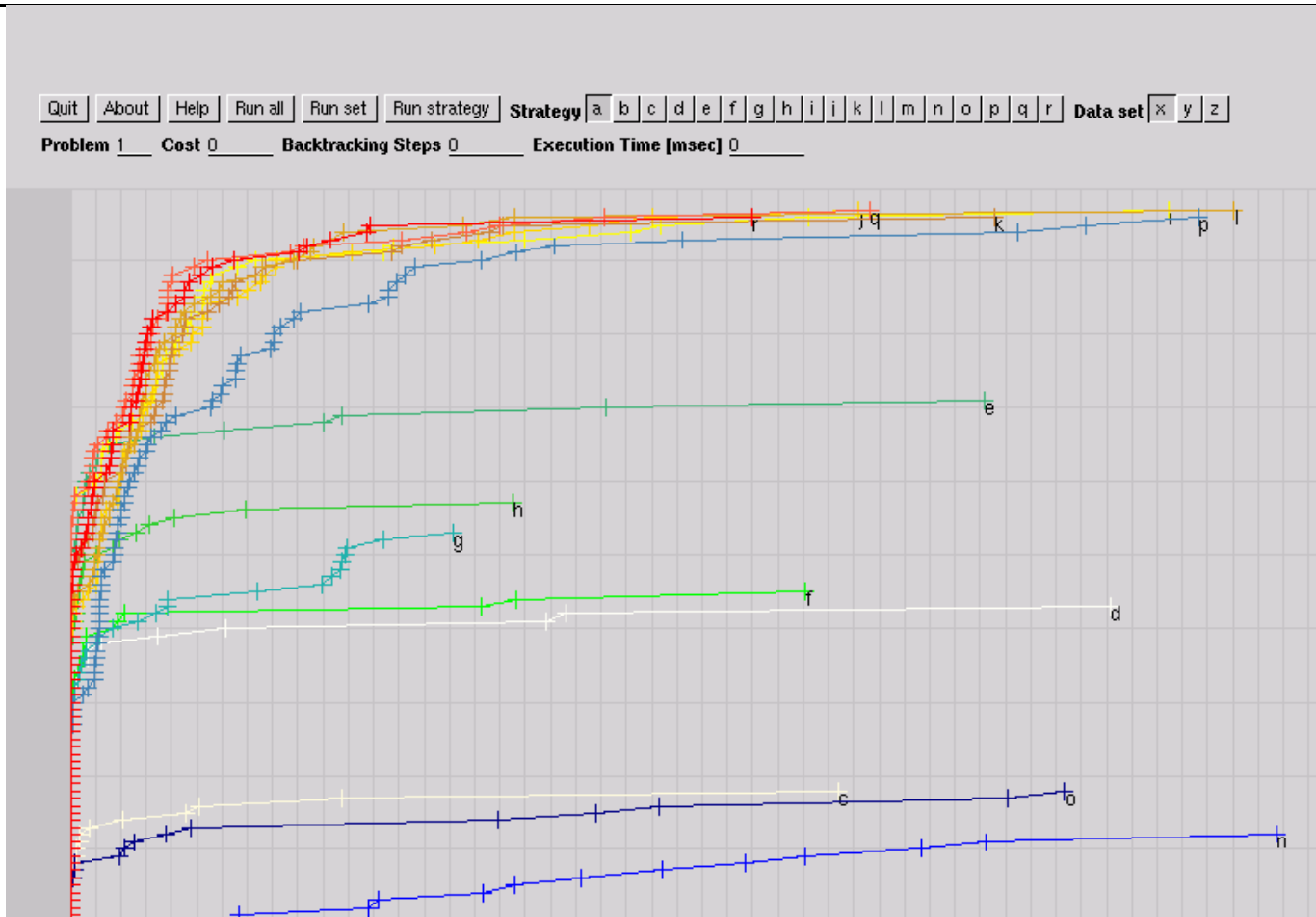


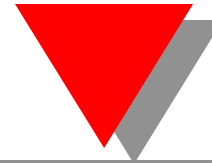
Data set X, backtrack plot strategies A-R, 5000 choices



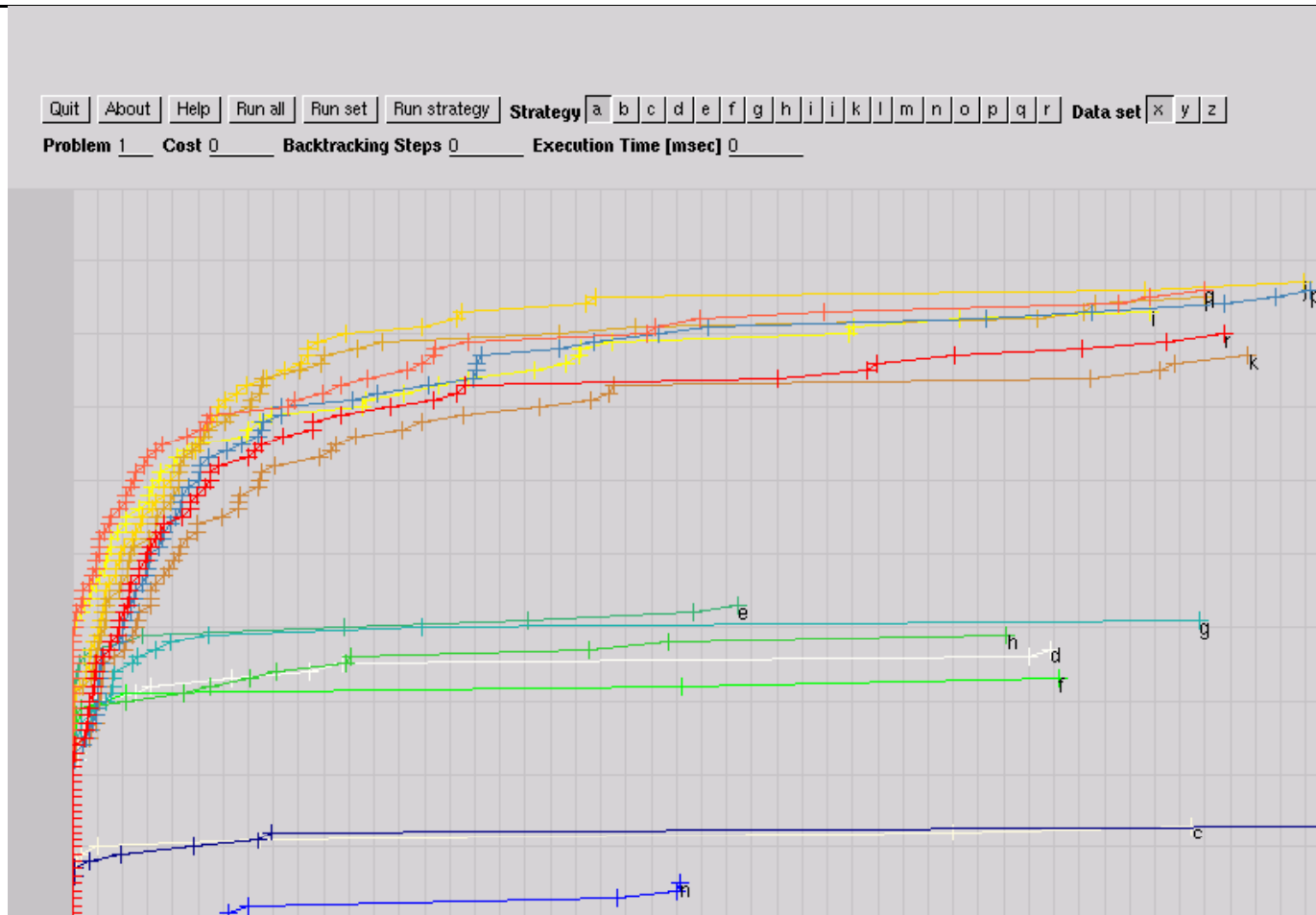


Data set Y, backtrack plot strategies A-R, 5000 choices





Data set Z, backtrack plot strategies A-R, 5000 choices



Evaluation

- ◆ **Rather complex constraints easily expressed**
- ◆ **Propagation limited**
 - does not detect inconsistency even in simple cases
 - constraints on lines and columns do not interact enough
 - needs redundant constraints for better results
- ◆ **Search strategy plays important role**
 - no single best method
 - combination of different methods achieves near 100% results
- ◆ **Partial search very significant improvement**
 - only if strategy is already working
- ◆ **Benefit of increased search visible for some strategies**
- ◆ **Effect of demand profile clearly visible**

Example 4: Fleet rotation

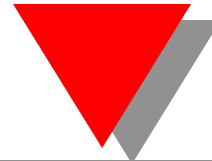
- ◆ **Given a set of flights**
 - Location A departure time A_{time} to
 - Location B arrival time B_{time}
- ◆ **Given minimal/maximal turn-around time**
 - 15 min - 4h
- ◆ **Find a set of aircraft rotations which covers all flights**
- ◆ **No empty flights allowed (cost)**

Objective

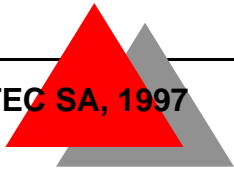
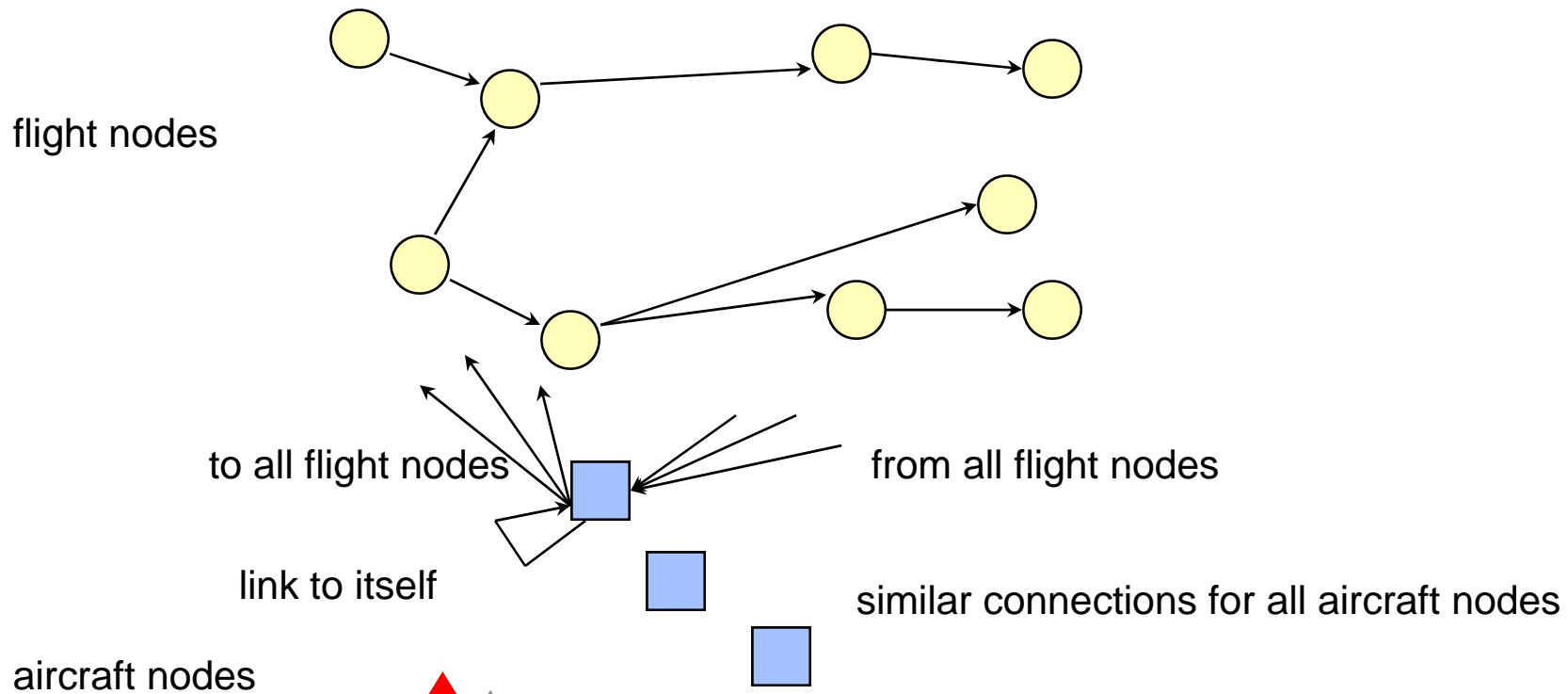
- ◆ **Here: minimize number of aircraft**
- ◆ **In reality, two alternatives:**
 - best use of given fleet (operational)
 - fleet requirement for program (strategic)

Idea of model

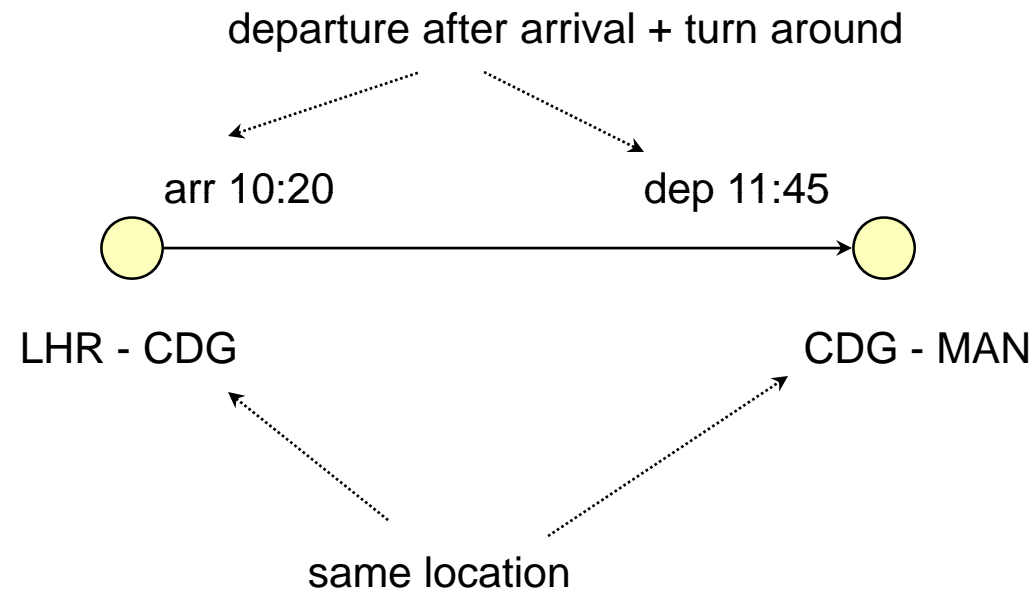
- ◆ Represent problem in form of a directed graph
- ◆ Find cycles in this graph
 - each cycle represents one aircraft
- ◆ Nodes are either
 - aircraft (special nodes)
 - flights (ordinary nodes)
- ◆ Links in graph
 - for each flight, decide what to do next
 - ◆ another flight, if it leaves from the arrival station after turn-around time, or
 - ◆ nothing, end work, represent as link to an aircraft node
 - for each aircraft, a link to a flight if work can start with this flight
 - for each aircraft, a link to itself



Graph



Successor flight



Cycles

- ◆ A cycle corresponds to the work done by one aircraft
- ◆ Each cycle contains one special node
- ◆ Special nodes may form cycle on their own
 - idle aircraft
- ◆ Successor value gives next activity
- ◆ Minimising number of aircraft equivalent to minimising number of cycles containing ordinary nodes

Model

◆ Variables

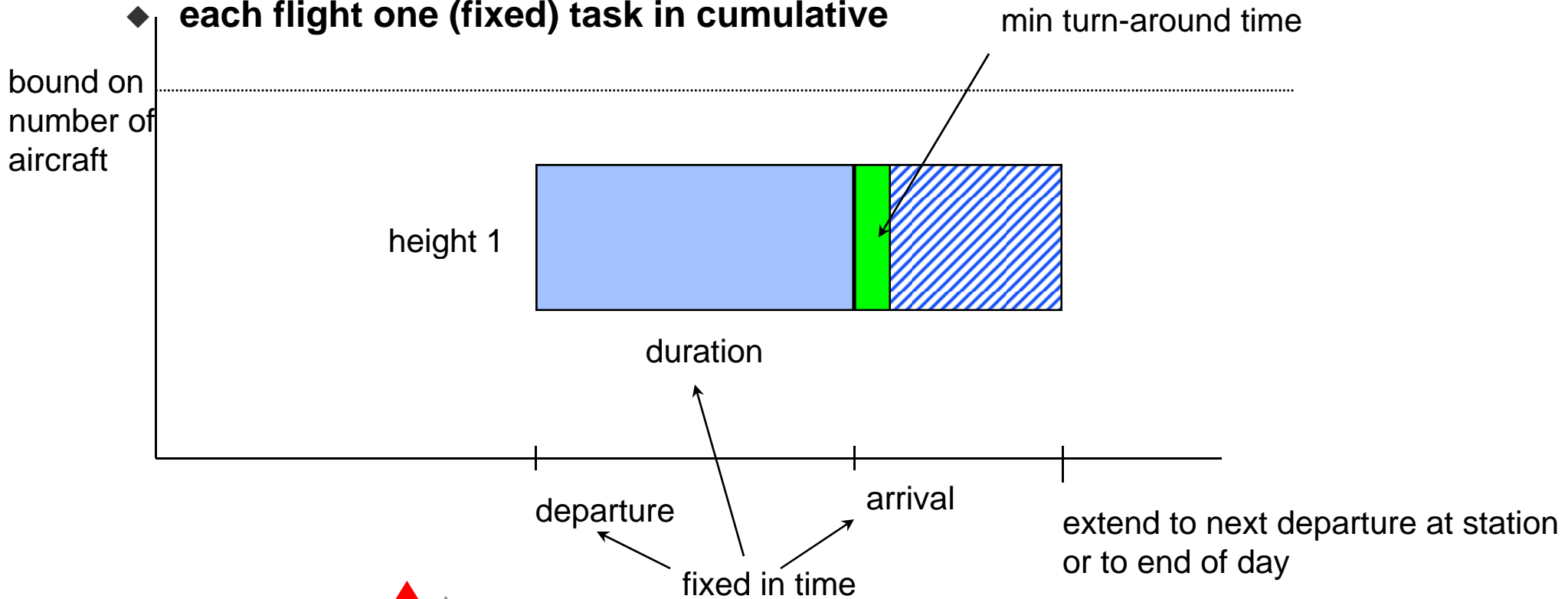
- One domain variable for each flight, one for each aircraft
- Domain of variable denotes possible successors
- Nodes numbered from 1 to $n+m$
 - ◆ 1.. n flight nodes
 - ◆ $n+1$.. $n+m$ aircraft special nodes
- Set of values forms permutation of 1.. $n+m$
- Number of cycles in permutation = number of aircraft

◆ Constraint

- cycle constraint
 - ◆ $\text{cycle}(N, \text{Succ})$
 - ◆ N = number of cycles (fixed, number of special nodes)
 - ◆ Succ = list of Successor variables

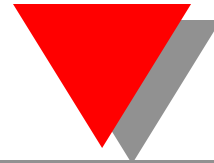
Lower bound calculation

- ◆ Cumulative profile of flights
- ◆ each flight one (fixed) task in cumulative

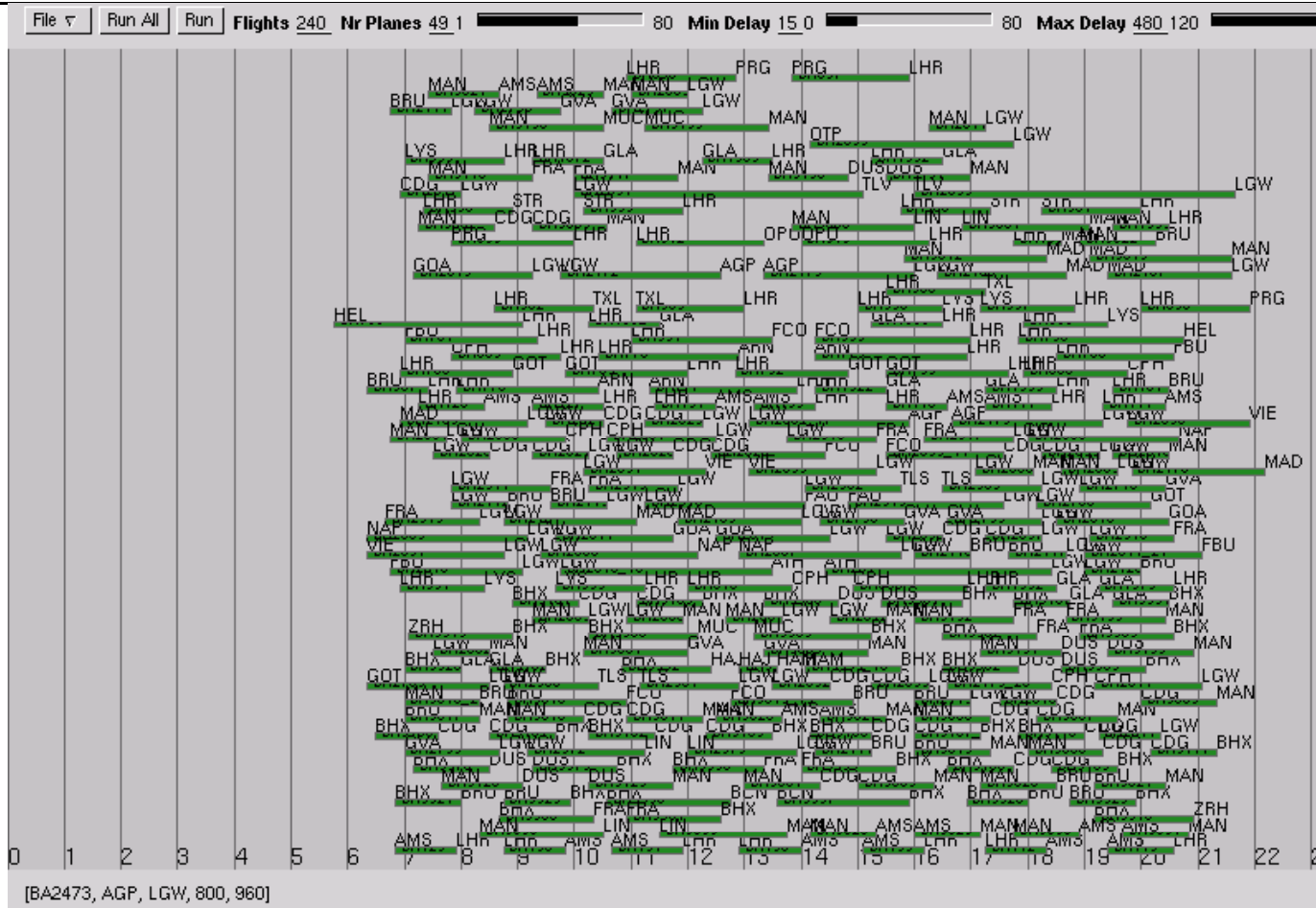


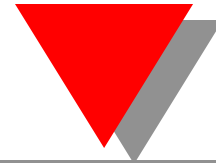
Data

- ◆ **Real data of 737 fleet of major European airline**
 - obtained from open data base
- ◆ **Rotation problem not realistic**
 - problem is already solved
- ◆ **240 flights on this day**
 - corresponds to well optimized rotations
- ◆ **Generated (artificial) sub-problems by removing flights randomly**
 - problems become harder, as no empty movement allowed
 - degenerated problem when many flights removed (no return flights)

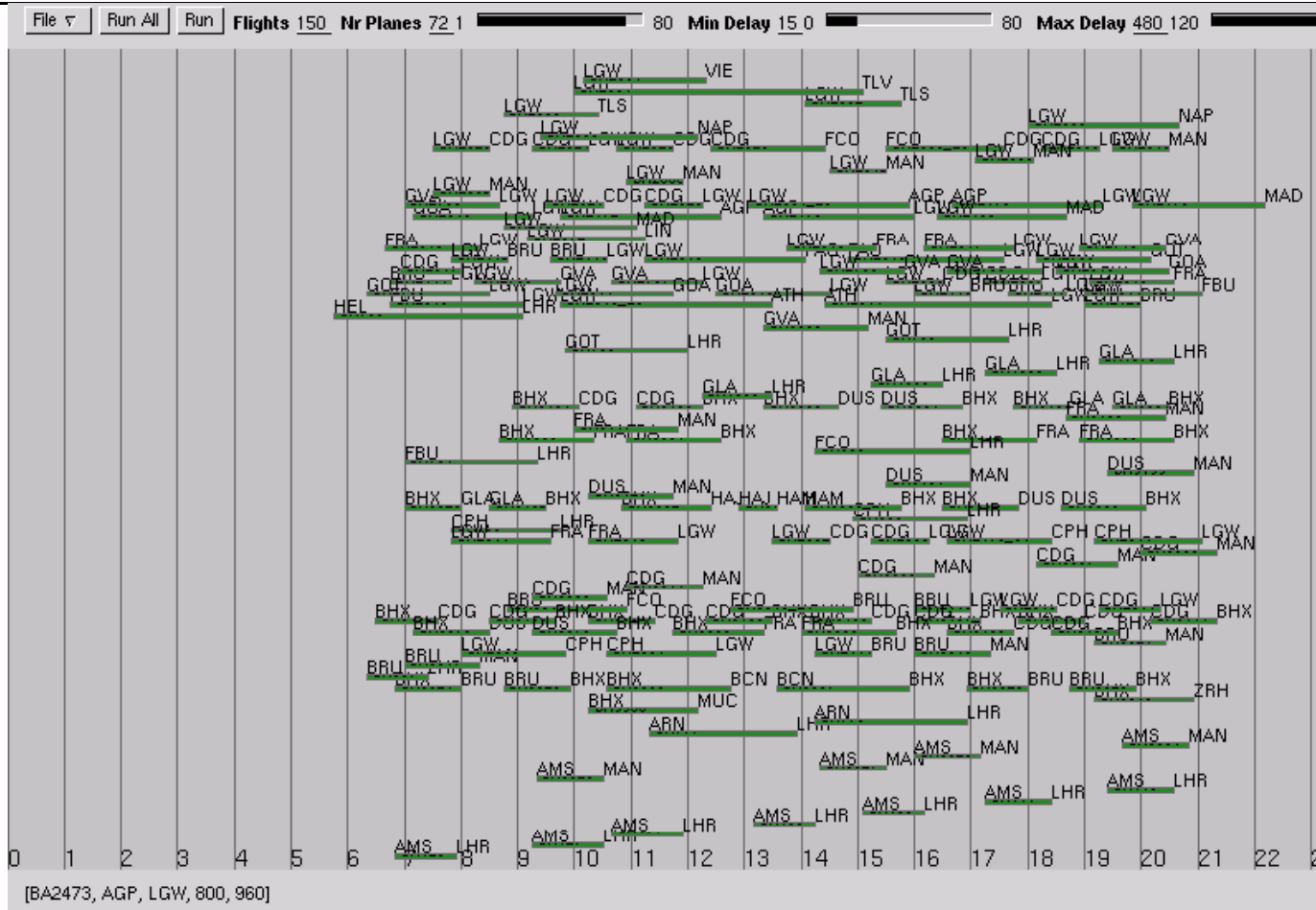


Solution (240 flights)

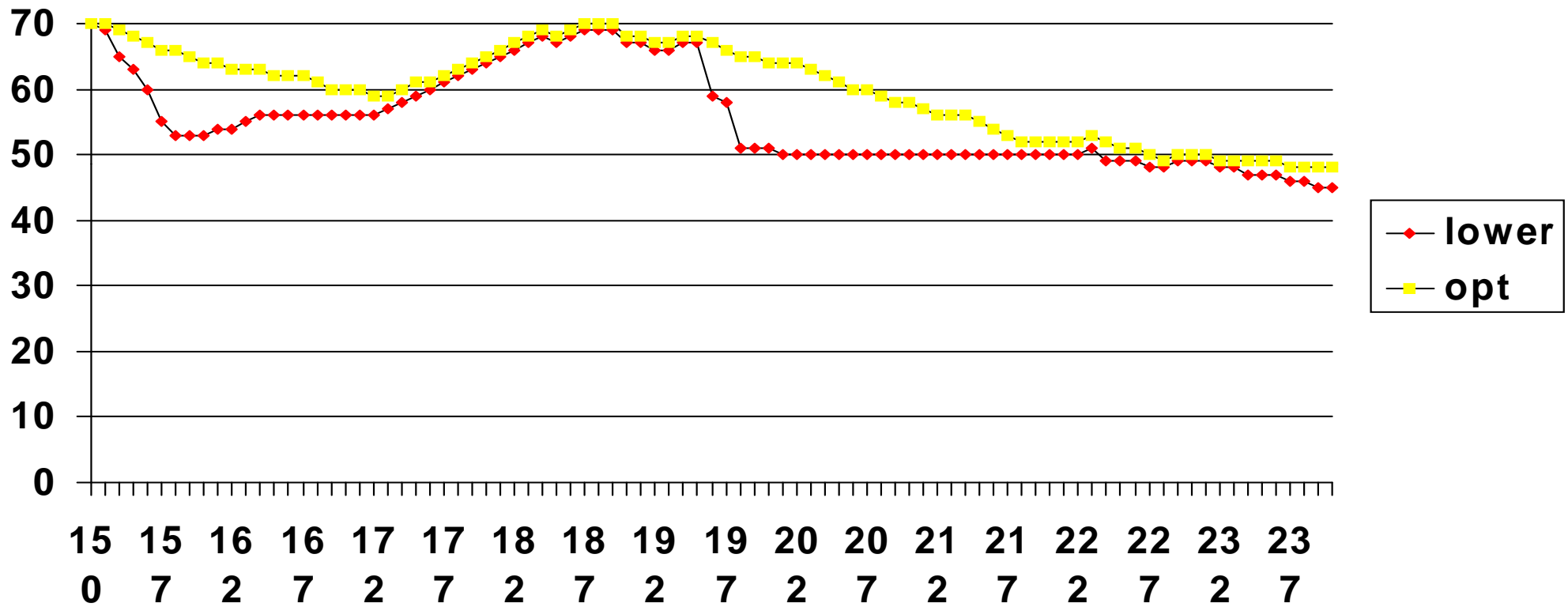




Solution (150 flights)



Lower bound/optimal solution



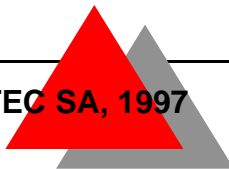
Evaluation

- ◆ **System finds and proves optimal solution in every instance without backtracking!**
- ◆ **Lower bound with cumulative quite good in non-degenerated case**
 - rather bad if many return flights are missing
- ◆ **Problem is too simple in its basic form**
 - additional constraints make optimality proof very hard
 - more complex problem require custom strategies
- ◆ **Model with cycle does not explicitly use time nor location**
 - all information incorporated in successor in directed graph
 - link with start time, resource assignment possible in cycle via parameters
- ◆ **Lower bound calculation uses completely different model**
 - relaxing some aspect of problem to obtain lower bound



Part 3

Summary



Evaluation

- ◆ Presented typical problems for finite domain constraints
- ◆ Each expressed easily by combination of global constraints
 - Examples for each of the global constraints
- ◆ Standard search methods provide good answers
 - often possible to improve by using custom programming
- ◆ Test cases show some stability of model for varying data
 - very important for practical applications

Stand allocation

- ◆ **Typical example of assignment problems**
- ◆ **Relatively simple instance**
 - resources not too tight
- ◆ **Model can be extended to handle other constraints**
 - binary interaction
 - more domain restrictions
- ◆ **Scheduling/Rescheduling quite fast**
- ◆ **Cost model not very powerful, optimization difficult**
- ◆ **Model useful for many practical problems**

Resource restricted scheduling

- ◆ Well known OR benchmark
- ◆ Problem very well structured
- ◆ Benchmark set
 - good for comparison
- ◆ Good lower bounds and first solutions
- ◆ Partial search useful to find improved solutions
- ◆ Project scheduling problems use this model
 - more types of constraints
 - work in progress
 - optimization may conflict with stability
- ◆ CHIP provides good results compared to specialized packages

Nurse scheduling

- ◆ **Simple model based on among and sequence constraints**
 - allows handling of complex rule sets without lots of primitive constraints
- ◆ **Problem size shown realistic**
- ◆ **Work in progress, exceptions create complications**
- ◆ **Results very good for perfect, balanced problems**
 - 100 % utilization
- ◆ **Problems get harder if demand varies a lot**
- ◆ **No single best strategy**
 - combination of heuristics more powerful than search
- ◆ **Partial search essential to avoid useless backtracking**

Fleet management

- ◆ **Basic problem in airline, train and lorry transport**
- ◆ **Only most basic form shown**
 - additional constraints
 - ◆ home bases
 - ◆ maintenance
 - ◆ empty stock movement
- ◆ **Good lower bound with cumulative**
 - ignores location continuity
- ◆ **Cycle constraint provides even better result**
 - systematic proof of optimality
- ◆ **Significant time to do initial propagation**
- ◆ **Full applications need custom strategies**

Next steps

- ◆ **Evaluate results**
 - try different CLP systems
 - ◆ data available
 - test other methods
 - more test data
- ◆ **Other standard models required**
 - production sequencing
 - production scheduling
 - crew scheduling
 - layout problems