# Standard Models 2
# for
# Finite Domain Constraint Solving

**Helmut Simonis**
**COSYTEC SA**
4, 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
  - 3 case studies
  - Model in CHIP
  - Experimental results
- Evaluation
  - Limits and possibilities
  - Other models

# Caveats

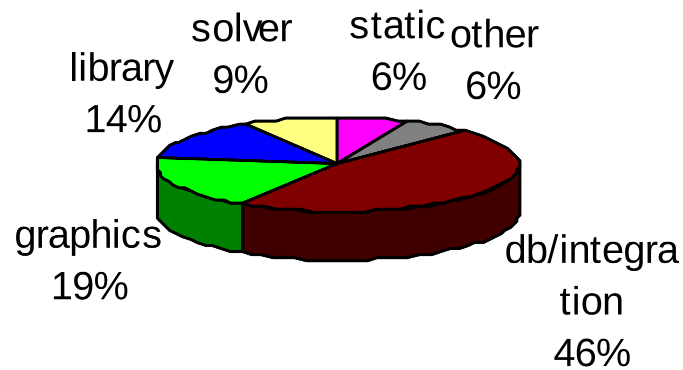
- Work in progress
  - models evolve in time
  - new constraints, new strategies appear
- Each problem is different
  - additional constraints
  - core of model remains the same
- There are no general solutions
  - need to adapt constraints, strategies
  - verify solutions against real data
- Models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

# Limited Liability

- Examples are industrial prototypes, not a PhD thesis
  - developed in a few days
  - based on a number of operational systems
- Better results possible with more time to study problem
  - examples:
    - Stand allocation
    - Nurse scheduling
- The model may not work for you
  - design decisions
  - bottleneck detection in problem
  - data dependent

# 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

- Another PACT 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
- Personnel assignment
  - Train, airlines

# Last year’s models

- Assignment
  - stand allocation for airport
  - classical problem for CLP
- Scheduling
  - resource restricted scheduling
  - benchmark problem in OR/scheduling
- Personnel time tabling
  - nurse scheduling
  - nice model, good results
- Transport
  - airline fleet rotation
  - solved to optimality

# Update

- **Stand allocation**
  - 0/1 integer programming model
  - proof of optimality
  - T. Kasper, MPI, Saarbruecken
- **Nurse scheduling**
  - 0/1 integer programming model
  - A. Bockmayr, MPI, Saarbruecken
  - 3 instances: solution missing (as in CHIP)
- **Nurse scheduling**
  - Gymnaste: Product by UJF, Praxim, COSYTEC
  - presentation at conference

# This year's models

- Scheduling
  - Production scheduling
  - Real-life complications
  - Multi-criteria search
- Transport
  - Lorry Bulk Transport
  - Linked to scheduling
- Production Sequencing
  - Batch based production
  - Linked to scheduling

Apologies:
no crew scheduling this year

# Some Background on CLP

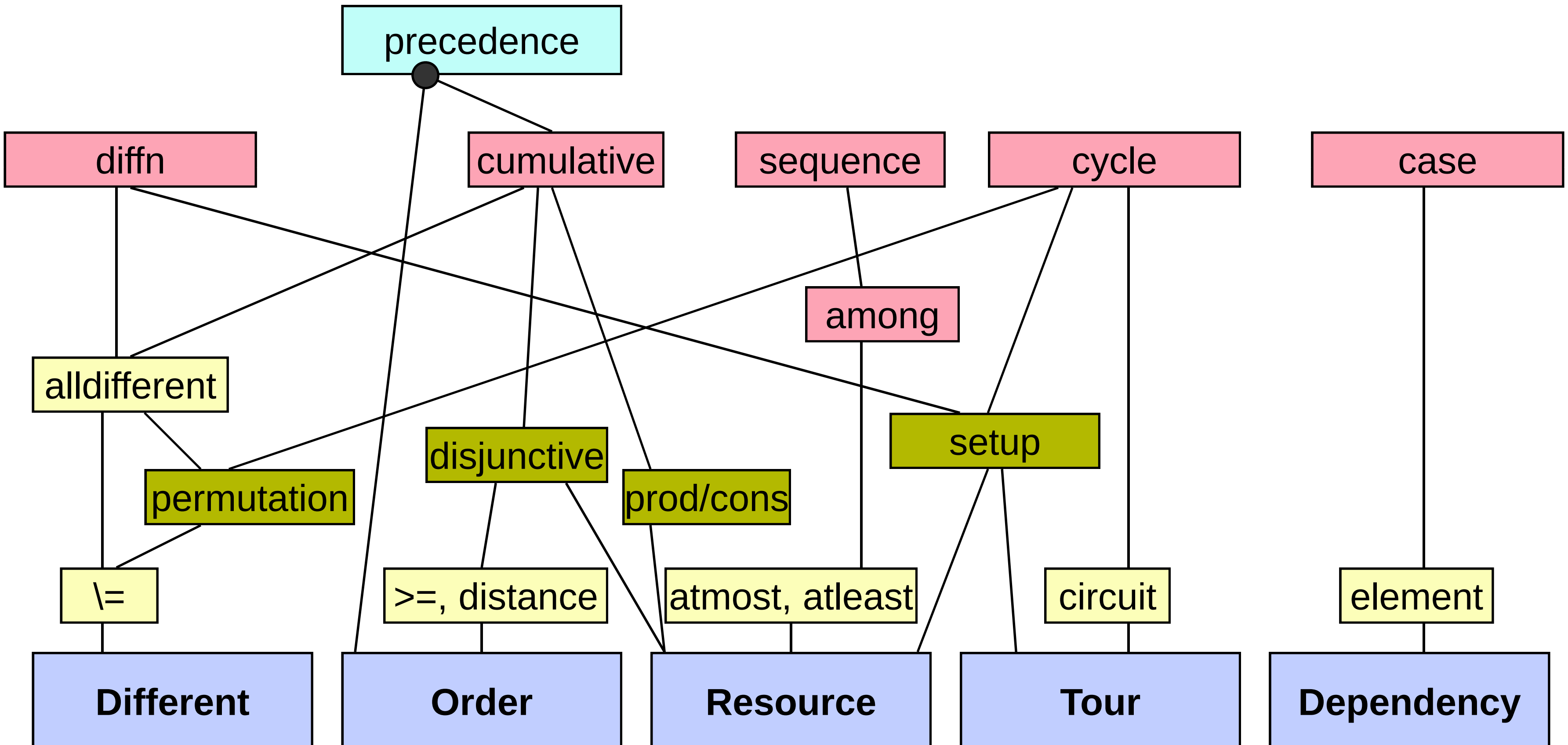
# Incomplete finite domain solver

- 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

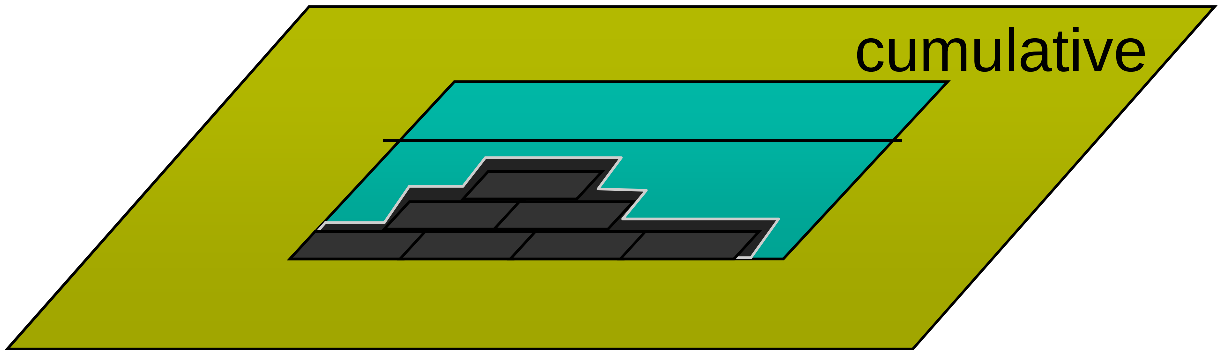
# Global constraints

- 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)
  - proper level of abstraction

# Constraint morphology

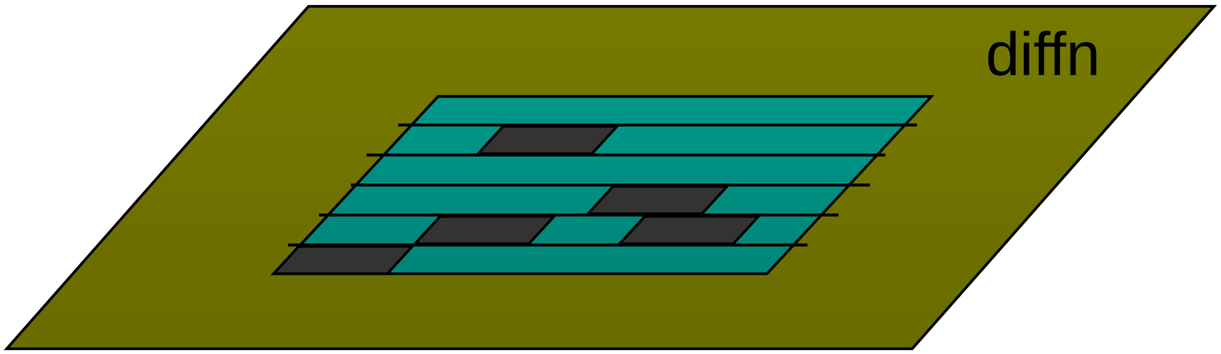


# The Cumulative global constraint



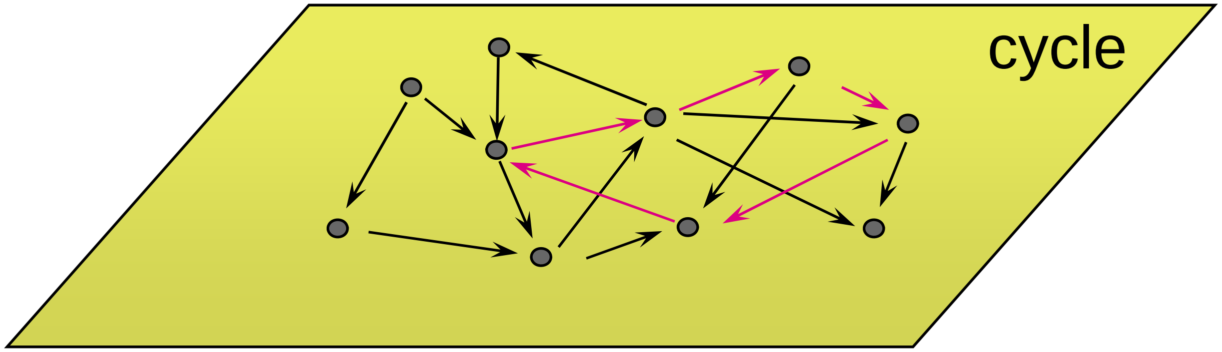

- **Cumulative constraint**
  - Resource limits over periods of time
  - Upper/lower limits
  - Soft/hard limits
  - Gradual constraint relaxation
- **Application**
  - Resource restrictive scheduling, producer consumer constraints, disjunctive schedule, manpower constraints, overtime

# The Diffn global constraint




- **Diffn constraint**
  – non overlapping areas on n-dimensional rectangles
  – distances between rectangles
  – limit use of areas
  – relaxation
- **Application**
  – layout, packing, resource assignment, setup, distribution planning, time-tabling

# The Cycle global constraint




◆ **Cycle constraint**

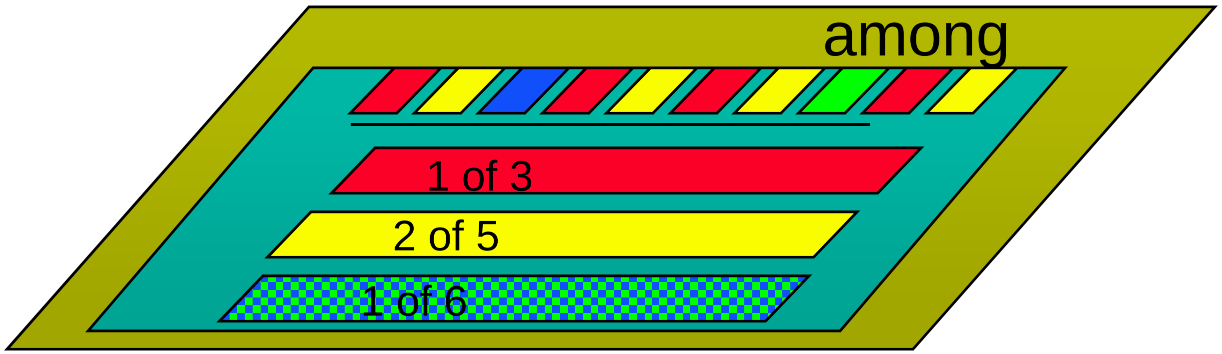
– Finds cycles in directed graphs with minimal cost

– Assign resources, find compatible start dates

◆ **Applications**

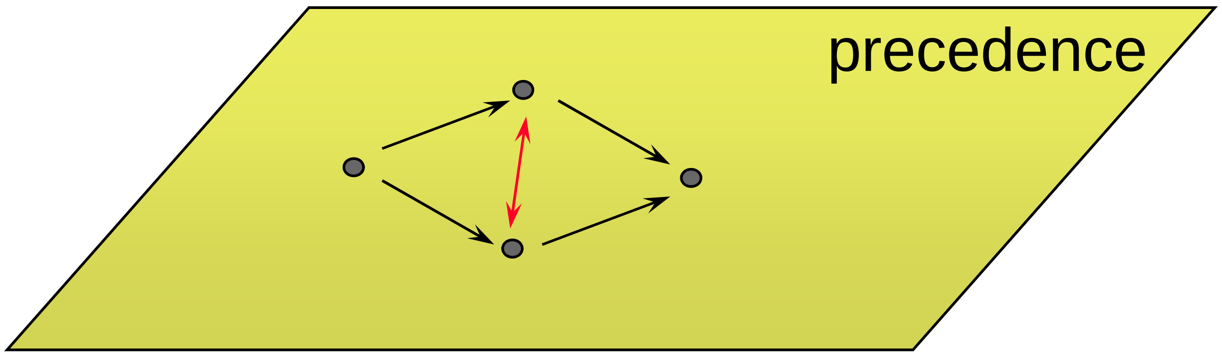
– Tour planning, personnel rotation, distribution problems, production sequencing

# The Among global constraint



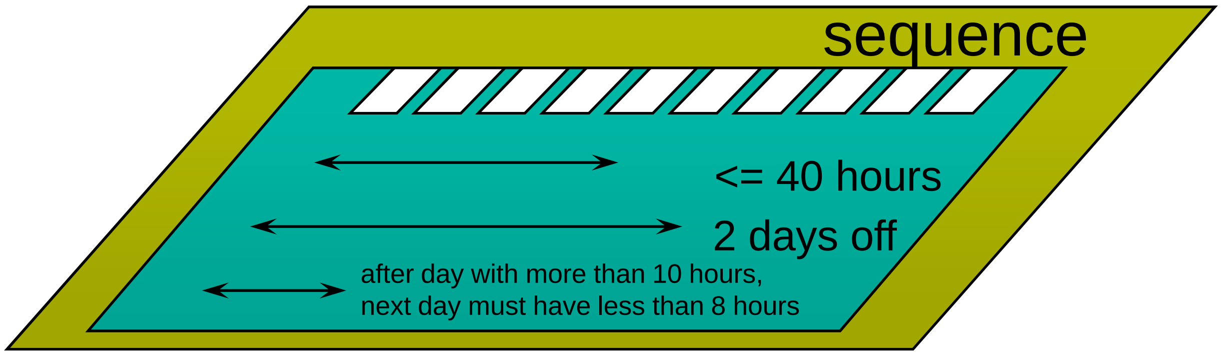

- **Among constraint**
  - **How often do values occur in (sub)sequences**
  - **based on counting arguments**
  - **interaction between sequences**
- **Applications**
  - **production sequencing, time tabling, coloring problems, set covering**

# The Precedence global constraint



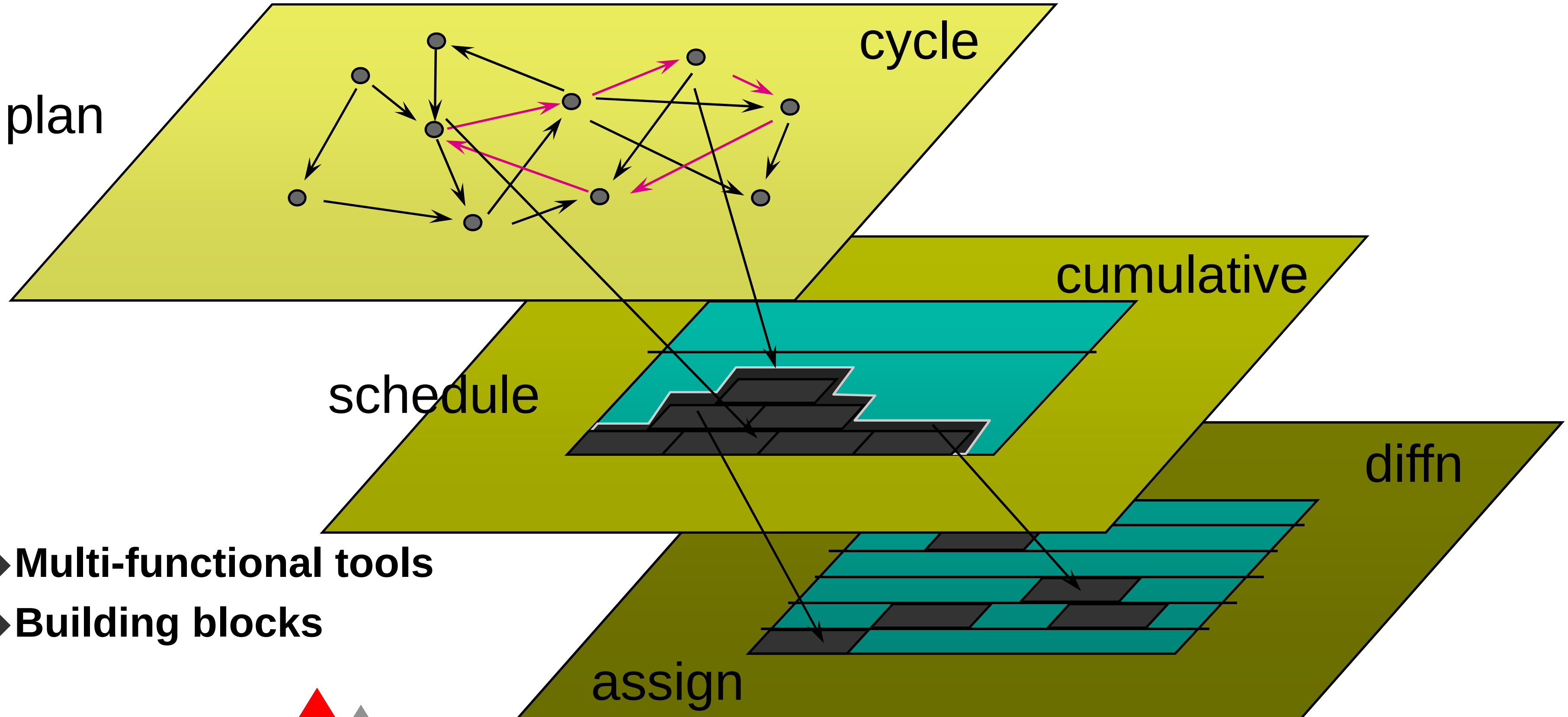

- **Precedence constraint**
  - **Combine resource constraints and precedence networks**
  - **Reasoning on latency (position in network)**
  - **Co-operation between multiple resources**
- **Applications**
  - **resource restricted scheduling, channel routing, frequency allocation**

# 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



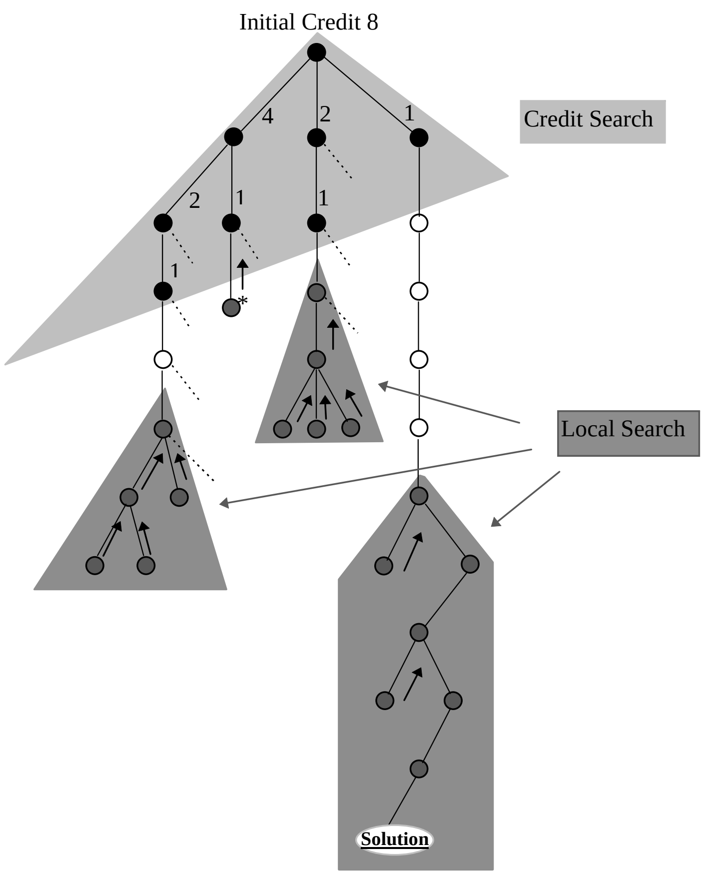

- Multi-functional tools
- Building blocks

# 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

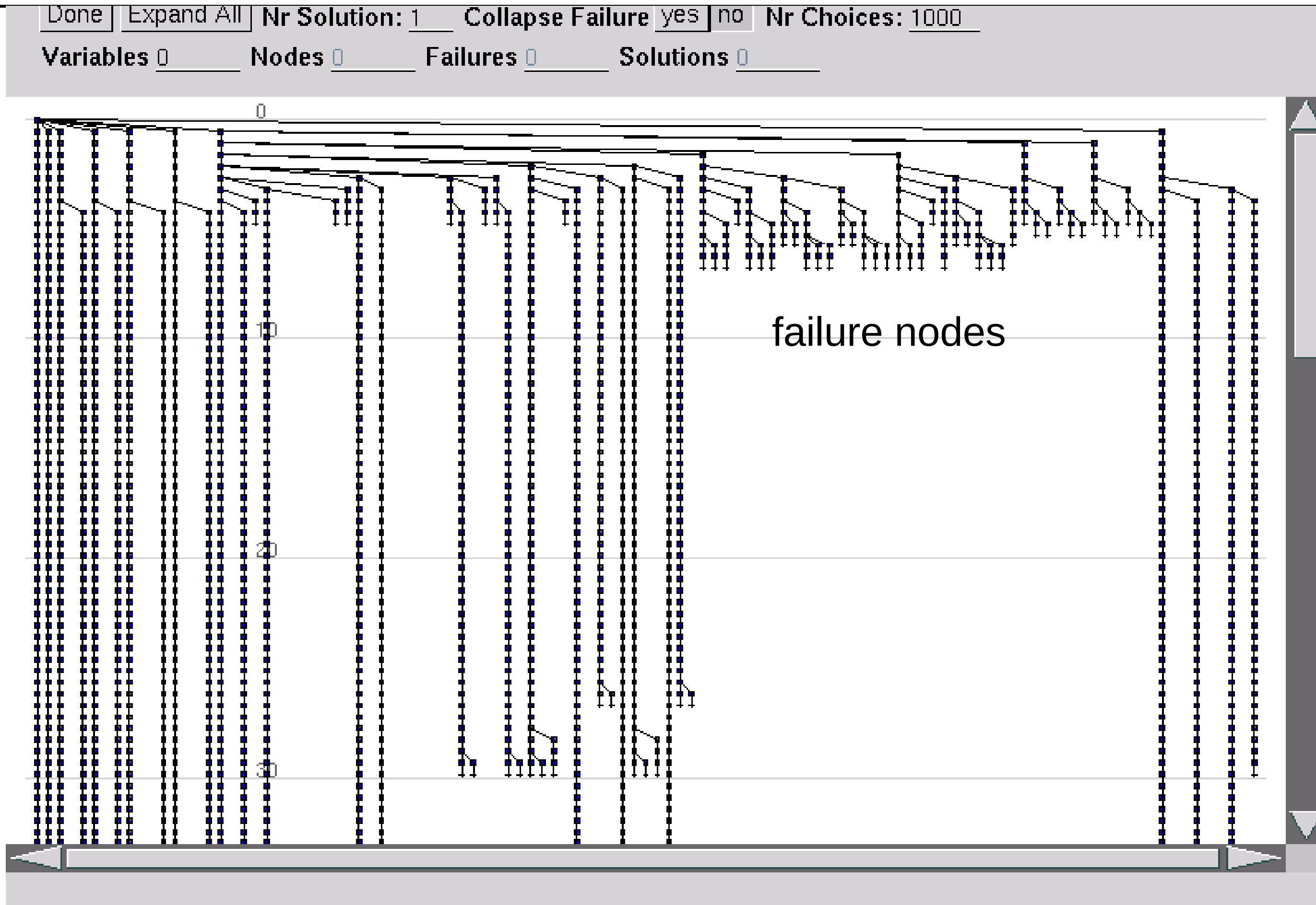
```
partial_dfs([X1..X10],
            8,
            10,
            my_delete,
            my_indomain,
            4,
            part(1,2)),
```



# 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 of simple meta-call predicates
- Supported by work in DISCIPL Esprit project
  - Declarative debugging
  - Visualization of constraint programming results
- Understand behaviour of different strategies

# Typical search tree visual




top level (min_max)

percentage of variables assigned

success nodes

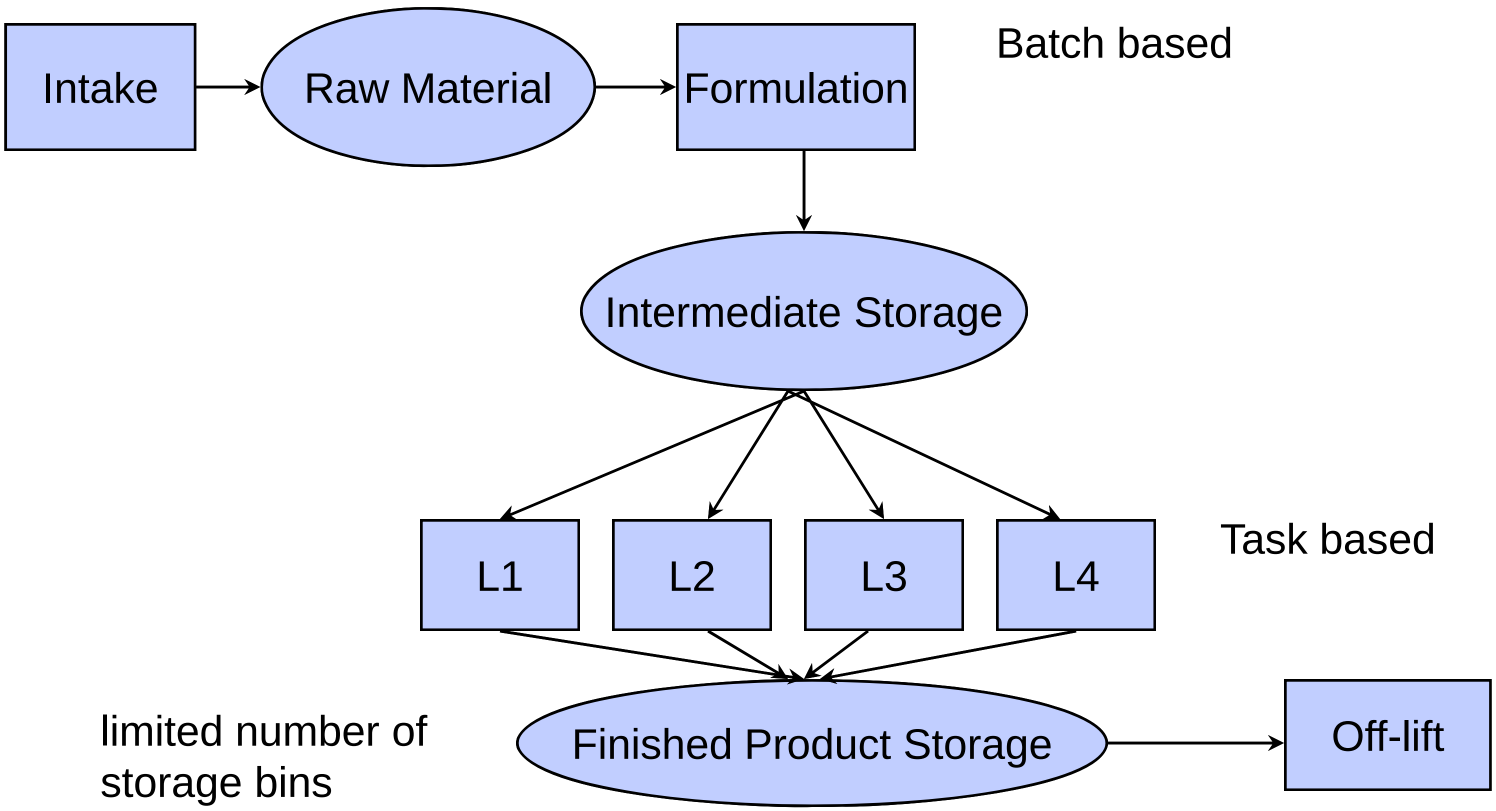
# Part 2

# Models

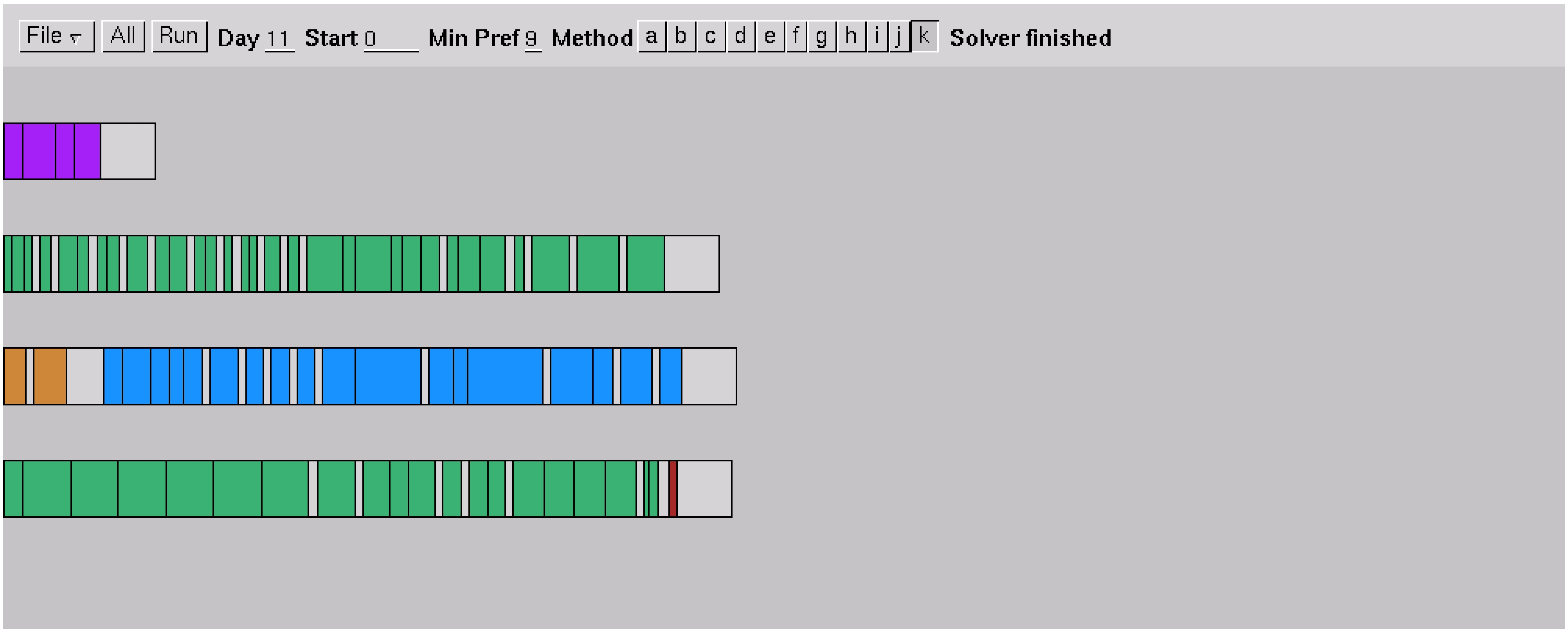
# Example 1: Production Scheduling

- Based on multiple real-life applications
  - Ignores some of the complexity
  - Simplified interaction between production days
- Typical semi-process scheduling problem
- Shows three different aspects of scheduling
  - task view
    - what is made when
  - successor view
    - what is followed by what
  - sequence view
    - how steps are arranged in sequence

# Process


Intake
Raw Material
Formulation
Batch based
Intermediate Storage
L1
L2
L3
L4
Task based
limited number of storage bins
Finished Product Storage
Off-lift

# Example solution



# Constraints

- Products come in different sizes
- Product/line specific duration,
- Line preference given by end-user
- Made to order production
- Orders multiple of batch sizes
- Not all products can be made on all lines
- Fastest line is not always best

# Throughput

- **Throughput**
  - based on historical data
  - updated periodically
- **Line preference**
  - quality criteria
  - utilization of plant with normal order book

| Line | Throughput | Preference |
|---|---|---|
| 1 | 9000 | 9 |
| 2 | 15000 | 3 |
| 3 | 6000 | 6 |
| 4 | - | - |

# Setup

- Product sizes 1-5
- Same size, same product: no setup
- Same size, different product: cleaning time
- Different size: machine set-up
- Not symmetrical

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 15 | 60 | - | 20 | - |
| 2 | 60 | 15 | 90 | 70 | - |
| 3 | - | 70 | 15 | - | - |
| 4 | 25 | 70 | - | 15 | - |
| 5 | - | - | - | - | 0 |

# Product contamination

- Contamination risk on production lines
- Can not make some products after other products
  - based on chemical composition
  - derivation from first principles rather complex
  - very high quality standards: no exceptions, ever
- Preference to make some product after given product
  - ignored here
- For each order
  - list of possible follow-on products
- Systematic cleaning operation at end of day

# Analysis results

- Main bottleneck
  - production lines
- Serious problem
  - finished product storage
- Also problem
  - formulation
- No problem
  - intake
  - raw material
  - intermediate storage
  - off- lift

# Design decision

- Scheduling of production lines
- Second solver for storage allocation
- Third solver for batch sequencing

# Objectives for line scheduling

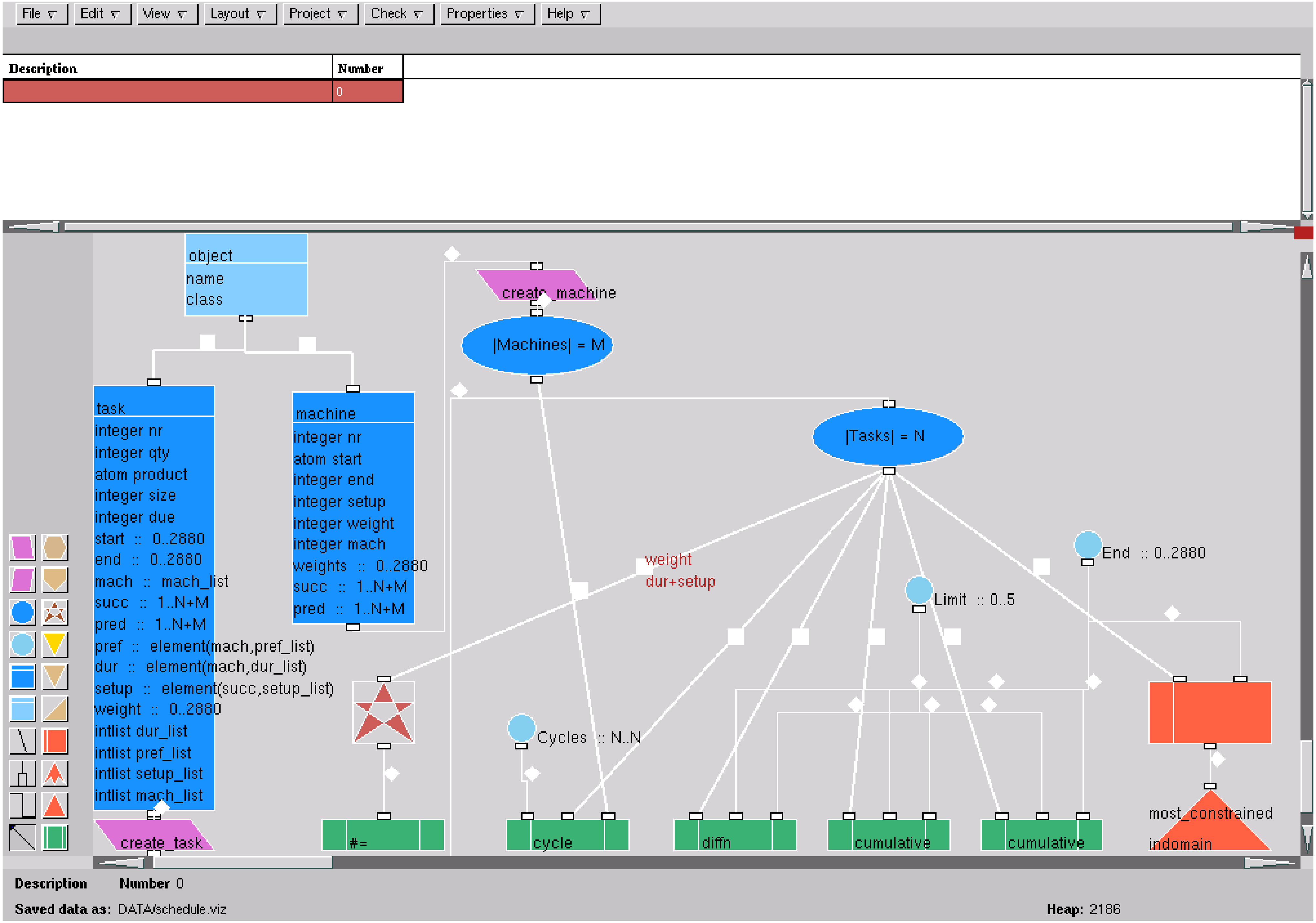
- 100% on-time
- No contamination
- Min setup
- Max line preference
- Max throughput
- Min stock cost
- Min bin space used
- Keep safety margin against delays
- Min energy cost
- Balance work load

# Data

- Order table
  - Nr Product Qty Due Date Time required
- Throughput
  - Product Line Throughput Preference
- Setup
  - Product size Product size Duration
- Contamination
  - Product List of possible follow-on products

# Model

- Object model
- Variables
- Constraints
- Search Method

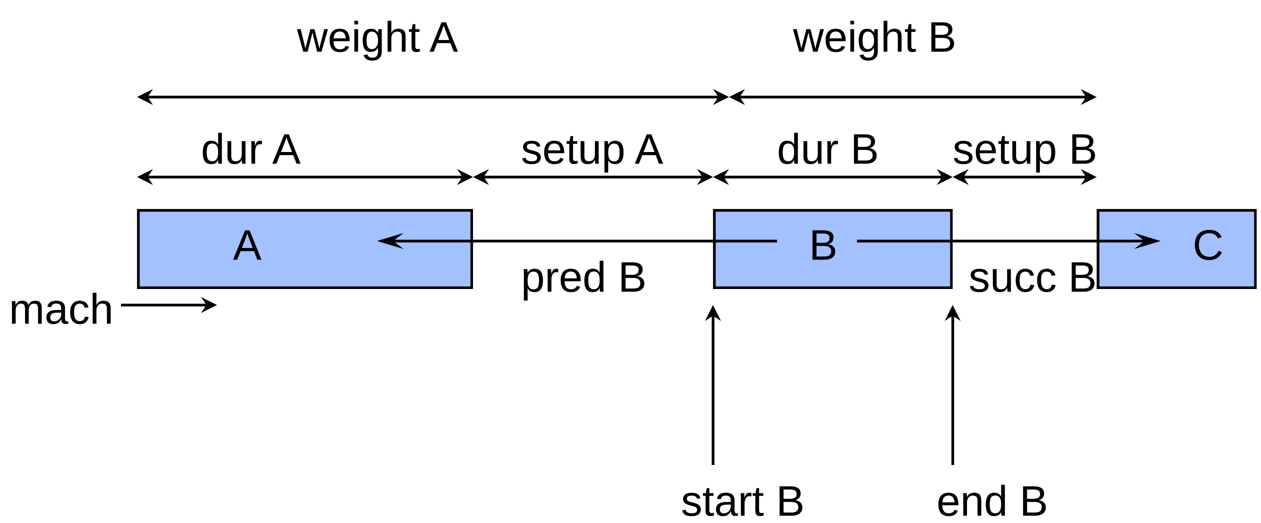

File
Edit
View
Layout
Project
Check
Properties
Help
Description
Number
0
object
name
class
create_machine
|Machines| = M
task
integer nr
integer qty
atom product
integer size
integer due
start :: 0..2880
end :: 0..2880
mach :: mach_list
succ :: 1..N+M
pred :: 1..N+M
pref :: element(mach,pref_list)
dur :: element(mach,dur_list)
setup :: element(succ,setup_list)
weight :: 0..2880
intlist dur_list
intlist pref_list
intlist setup_list
intlist mach_list
machine
integer nr
atom start
integer end
integer setup
integer weight
integer mach
weights :: 0..2880
succ :: 1..N+M
pred :: 1..N+M
|Tasks| = N
weight
dur+setup
End :: 0..2880
Limit :: 0..5
Cycles :: N..N
create_task
#=
cycle
diffn
cumulative
cumulative
most_constrained
indomain
Description Number 0
Saved data as: DATA/schedule.viz
Heap: 2186

# Object model

- 1-1 correspondence order <-> task
  - no need for combine orders for runs
  - different due dates
  - ease of extension
- Task class
  - static data
    - nr, qty, product, size, due date
- Machine class
  - static data
    - nr

# Variables

- **Task**
  - **start**
  - **end**
  - **mach**
  - **dur**
  - **setup**
  - **succ**
  - **pred**
  - **weight**
- **Machine**
  - **succ**
  - **pred**
  - **constant:**
    - **mach, setup, start, end**



# Constraints

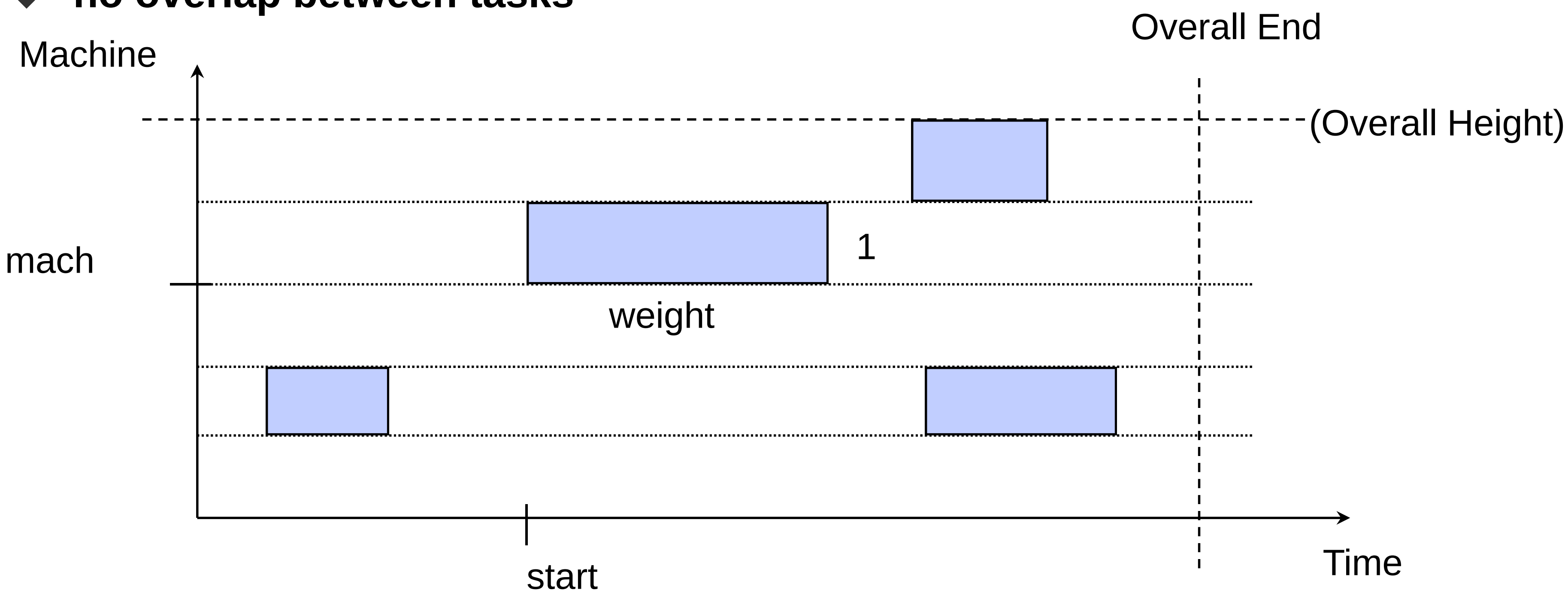
- linking variables
- diffn for machine assignment
- cycle for successor view
- redundant cumulative for machine use
- redundant cumulative for bin packing

# Linking variables

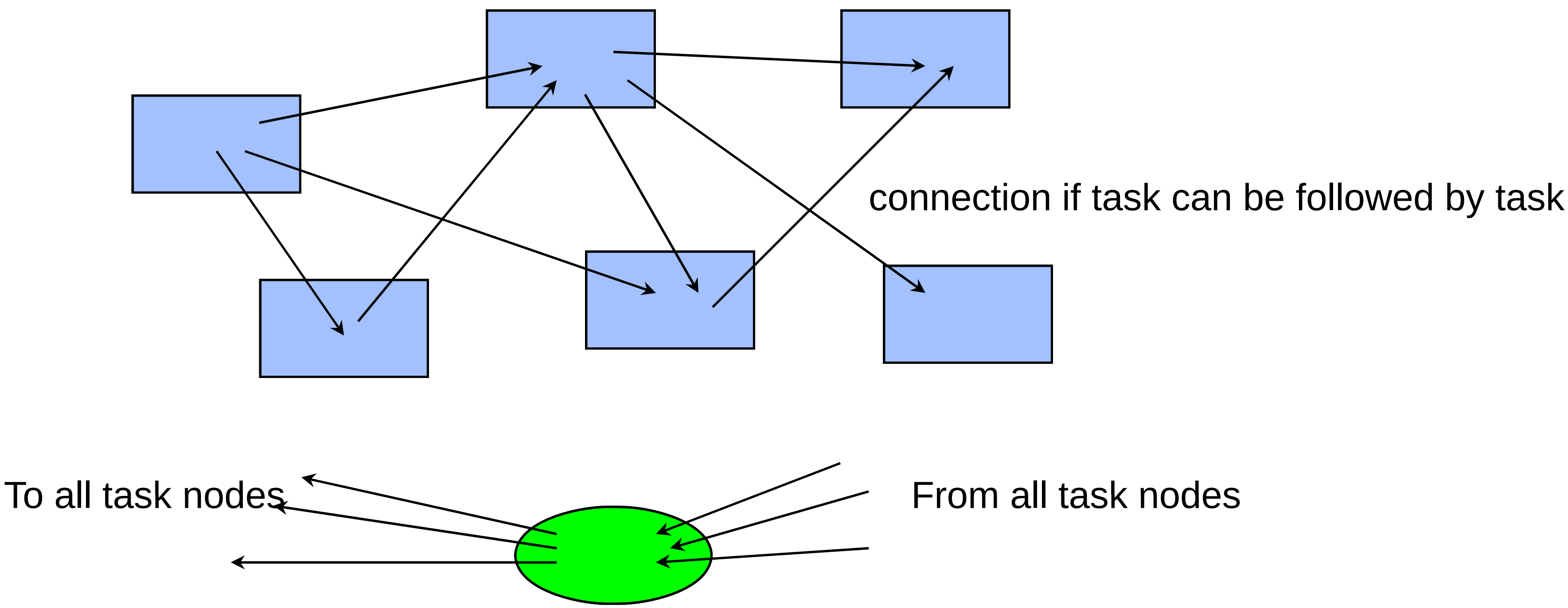
- Machine dependent duration
  - element(Mach, Duration table, Duration)
- Line Preference
  - element(Mach, Line preference table, Line Preference)
  - Line Preference #>= Min Preference
- Setup
  - element(Succ, Setup table, Setup)
- Weight
  - Weight #= Dur+Setup
- Start, End
  - End #= Start+Weight
  - End #<= Due

# Machine assignment

- **2D diffn constraint**
- **no overlap between tasks**

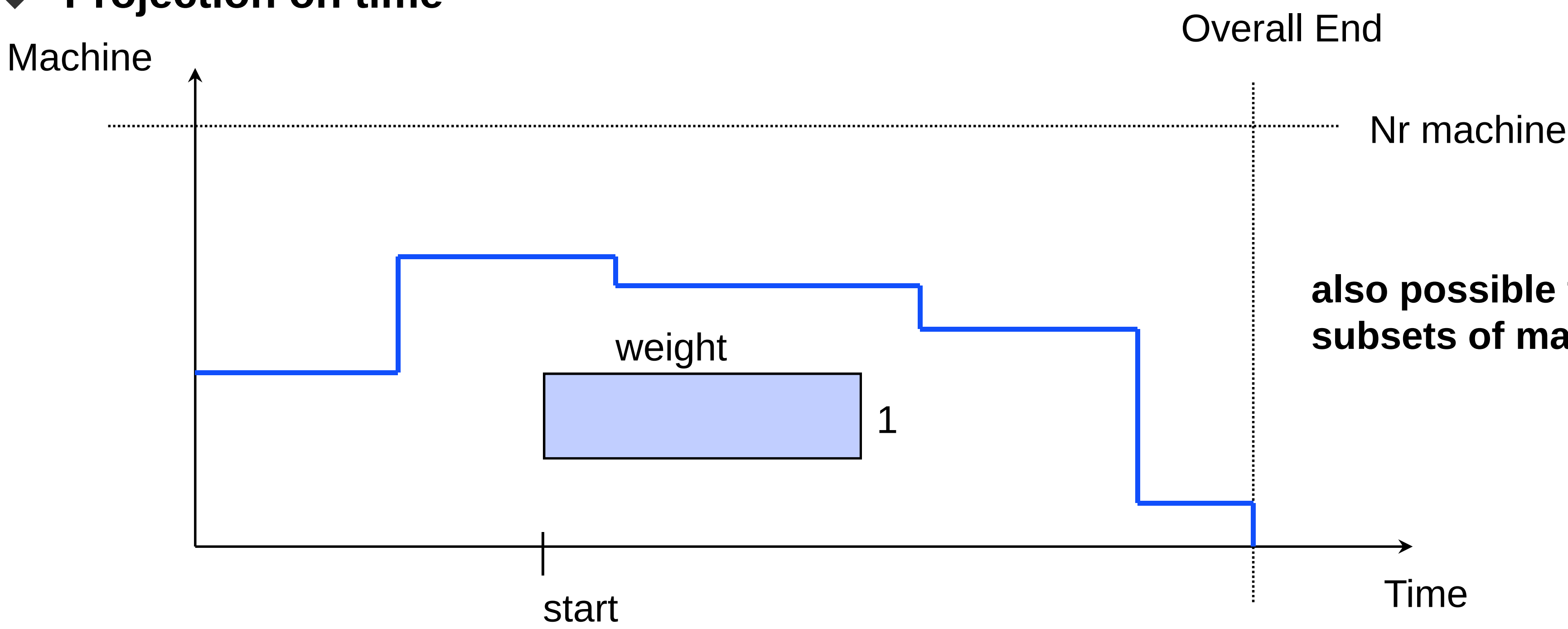


# Cycle graph


connection if task can be followed by task
To all task nodes
From all task nodes

# Machine use

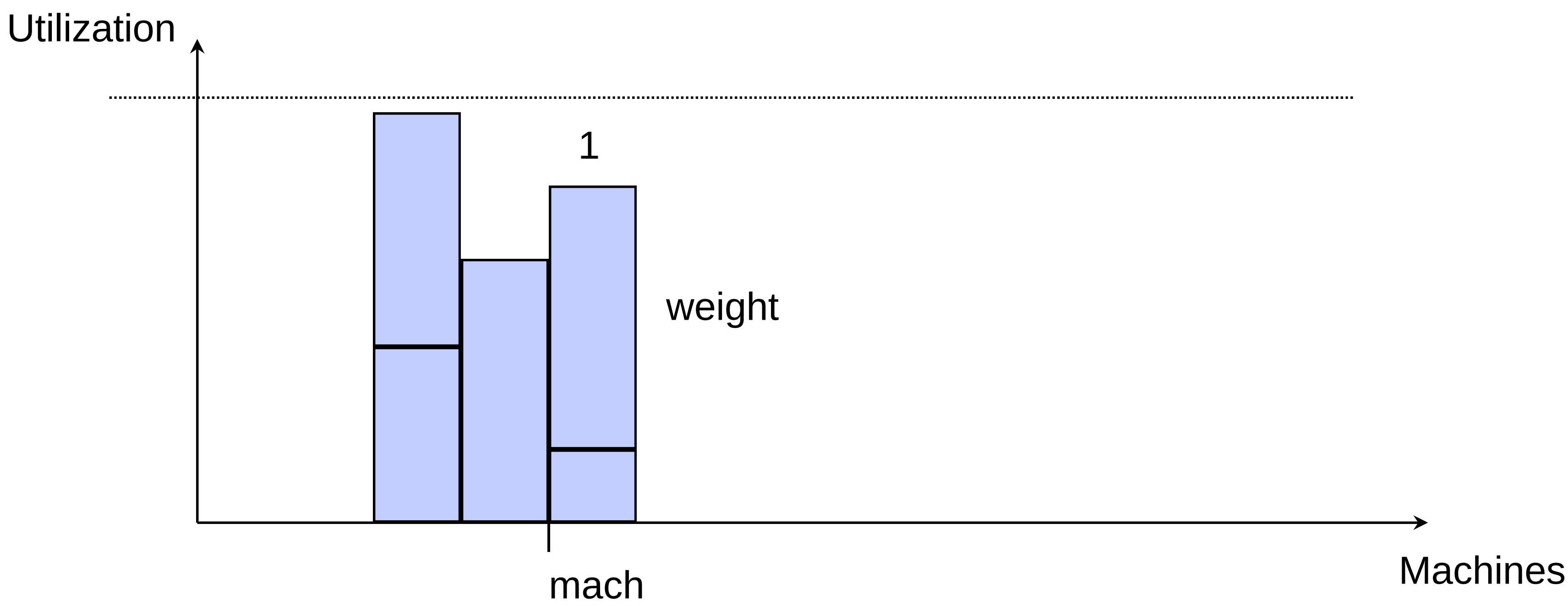
- **Redundant cumulative constraint**
- **Projection on time**




**also possible for subsets of machines**

# Bin packing view

- Redundant cumulative constraint
- Projection on machines



# Program skeleton

```
run(Day):-
        create_objects(Day,Tasks,Machines),

        length(Machines,M),length(Tasks,N),N1 is N+M,
        prepare_machine_variables(Machines,N1),
        prepare_variables(Tasks,N1,Tasks),
        extract_rect(Tasks,Rect),
        extract_3(Tasks,Start3,Dur3,Res3),
        extract_bin_packing(Tasks,Start,Dur,Res),
        End :: 0..2880,
        Height :: 1..5,
        Limit3 :: 0..3,

        set_cycle(Machines,Tasks,N1),
        diffn(Rect,unused,unused,[End,Height]),
        cumulative(Start3,Dur3,Res3,unused,unused,Limit3,End,unused),
        cumulative(Start,Dur,Res,unused,unused,End,unused,unused),

        toggle(method,current(Method)),
        choose_method(Method,Machines,Tasks,Rect,End).
```

# Variable set-up

```
prepare_variables([],_,_).
prepare_variables([A|A1],N,Tasks):-
        num(start,current(Low)), num(pref,current(Min_pref)),
        A@start :: Low..2880, A@end :: Low..2880,
        prepare_machine_choice(A,[1,2,3,4],Machine_list,Line_pref_list,Duration_list),
        A@mach :: Machine_list,
        element(A@mach,Line_pref_list,A@pref),
        A@pref #>= Min_pref,
        element(A@mach,Duration_list,A@dur),
        A@due  is 1440+A@time,
        A@due #> A@start+A@dur,
        A@end #= A@start+A@dur,
        A@succ :: 1..N, A@pred :: 1..N,
        prepare_setup(A,Tasks,Setup_list), element(A@succ,Setup_list,A@setup),
        A@weight :: 0..2880, A@weight #= A@dur + A@setup,
        prepare_variables(A1,N,Tasks).
```

# Cycle constraint

```
set_cycle(Mach,Tasks,Size):-
        extract_cycle_machine(Mach,Succ1,Weight1,Assign1,Start1,Pred1,Weights),
        extract_cycle_task(Tasks,Succ2,Weight2,Assign2,Start2,Pred2),
        length(Mach,N),
        append(Succ1,Succ2,Succ),
        append(Weight1,Weight2,Weight),
        append(Assign1,Assign2,Assign),
        append(Start1,Start2,Start),
        append(Pred1,Pred2,Pred),
        cycle(N,Succ,Weight,0,10000,Assign1,Weights,Assign,Start),
        inverse(Succ,Pred,all,all).
```

# Search Strategies

- **Conventional labeling**
  - assign start time and machine
- **Successor labeling**
  - assign follow-on production
- **Continuation labeling**
  - assign predecessor , one task at a time
- **Snake**
  - one machine at a time, assign successors
- **Cyclic snake**
  - all machines, define cyclic successors
- **Multi-snake**
  - all machines, define successors dynamically

# Conventional labeling

- Assign machine and start date
  - does not work with setup times
  - after assignment in time and on machine
    - not easy to understand the implication for sequencing
    - gaps wide enough for additional tasks ?
    - requires left to right assignment

# Example Code

```
choose_method(a,Mach,Tasks,Rect,End):-
        method_extract_a(Tasks,Vars),
        min_max((labeling(Vars,1,smallest,assign_a),
                draw(Rect,Tasks)),End,0,2880,0,60).

method_extract_a([],[]).
method_extract_a([A|A1],[t(S,M,W)|R]):-
        S = A@start,
        M = A@mach,
        W = A@weight,
        method_extract_a(A1,R).

assign_a(t(S,M,W)):-
        indomain(M),
        indomain(S),
        indomain(W).
```

# Successor labeling

- Choose successor for each task
- assign start times as second step
- creates unconnected lines

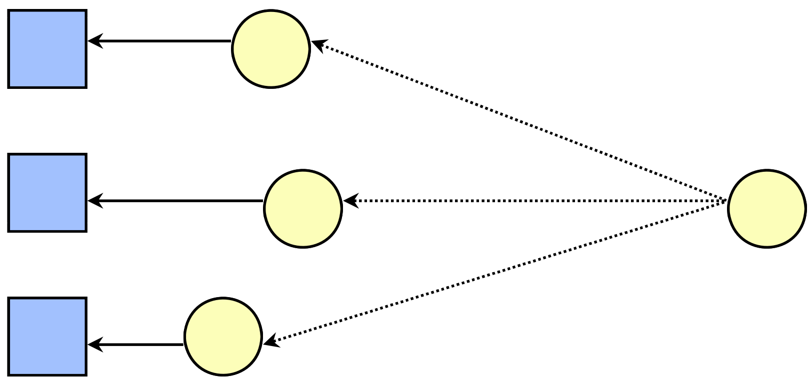
# Example Code

```
choose_method(b,Mach,Tasks,Rect,End):-
        method_extract_b(Tasks,Vars),
        method_extract_start(Tasks,Start),
        min_max(( labeling(Vars,1,most_constrained,assign_b),
                labeling(Start,0,smallest,indomain),
                 draw(Rect,Tasks)),End,0,2880,0,60).

method_extract_b([],[]).
method_extract_b([A|A1],[t(Succ,Setup)|R]):-
        Succ = A@succ,
        Setup = A@setup,
        method_extract_b(A1,R).

assign_b(t(Succ,Setup)):-
        fix_b(Succ,Setup).
```

```
method_extract_start([],[]).
method_extract_start([A|A1],[S|R]):-
        S = A@start,
        method_extract_start(A1,R).
```

```
fix_b(Succ,Setup):-
        Succ #> 4,
        indomain(Setup),
        indomain(Succ).
fix_b(Succ,Setup):-
        Succ #<= 4,
        indomain(Succ).
```

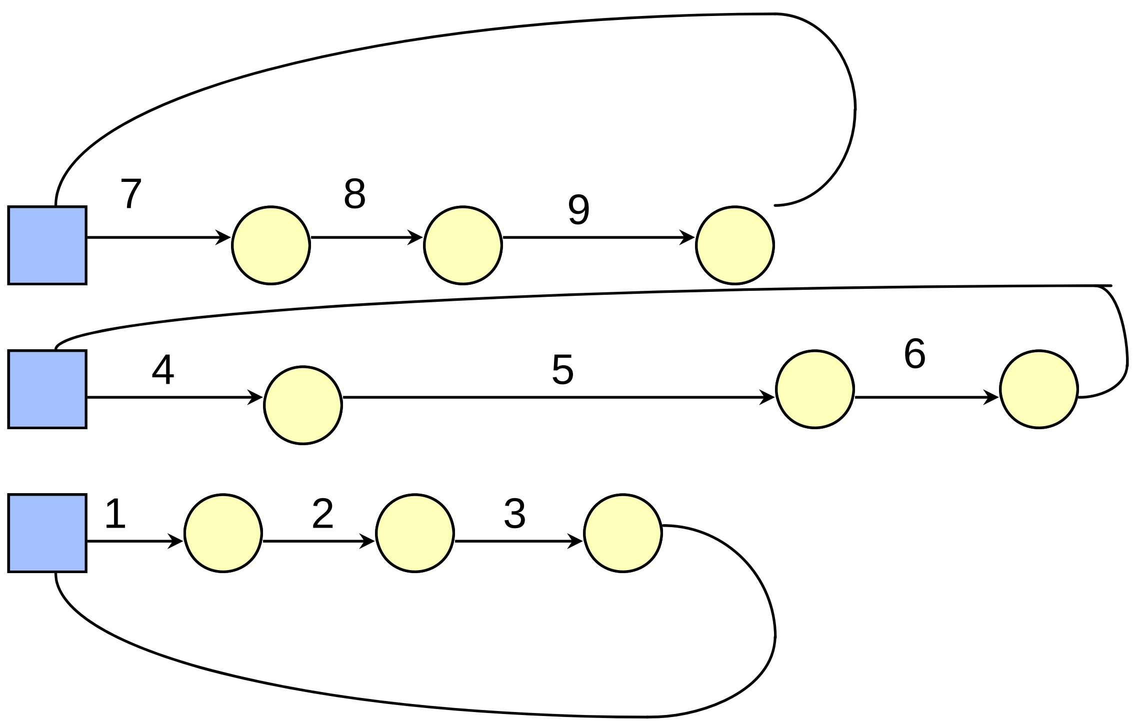
# Continuation labeling



# Example Code

```
choose_method(f,Mach,Tasks,Rect,End):-
        method_extract_f(Mach,Machine_nr),
        min_max((back_snake(Tasks,Machine_nr), draw(Rect,Tasks)),End,0,2880,0,60).

method_extract_f([],[]).
method_extract_f([A|A1],[S|R]):-
        S = A@nr,
        method_extract_f(A1,R).

back_snake([],_).
back_snake([H|T],Frontier):-
        back_snake_select(X,[H|T],R,Frontier),
        delete(X@pred,Frontier,Rest),
        inc_back,
        once(indomain(X@start)),
        back_snake(R,[X@nr|Rest]).

back_snake_select(A,B,C,Frontier):-
        delete(A,B,C),
        not not member(A@pred,Frontier),
        !.
```

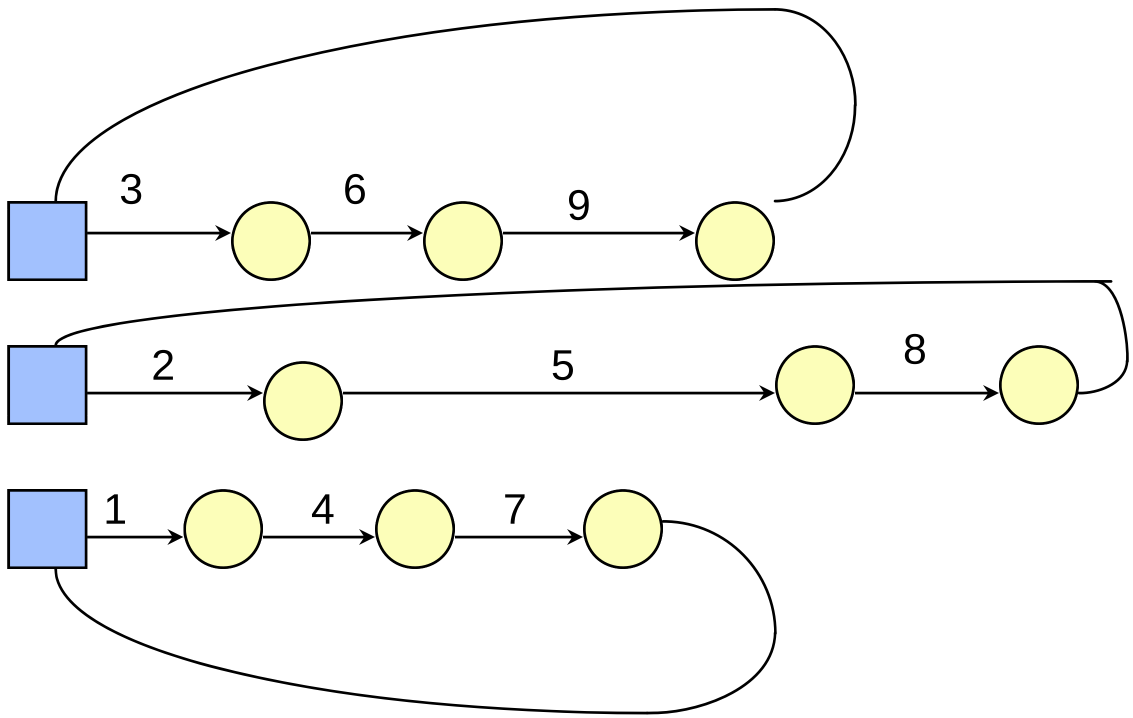
# Snake



# Example Code

```
choose_method(h,Mach,Tasks,Rect,End):-
        append(Mach,Tasks,Obj_list),
        Objs =..[f|Obj_list],
        min_max((snake(Objs,Mach),draw(Rect,Tasks)),End,0,2880,0,60).

snake(Objs,[]).
snake(Objs,[X|Frontier]):-
        !,
        fix_b(X@succ,X@setup),
        snake_cont(X,Objs,Frontier).

snake_cont(X,Objs,Rest):-
        X@succ > 4,!,
        arg(X@succ,Objs,Obj),
        once(indomain(Obj@start)),
        snake(Objs,[Obj|Rest]).
snake_cont(_,Objs,Rest):-
        snake(Objs,Rest).
```

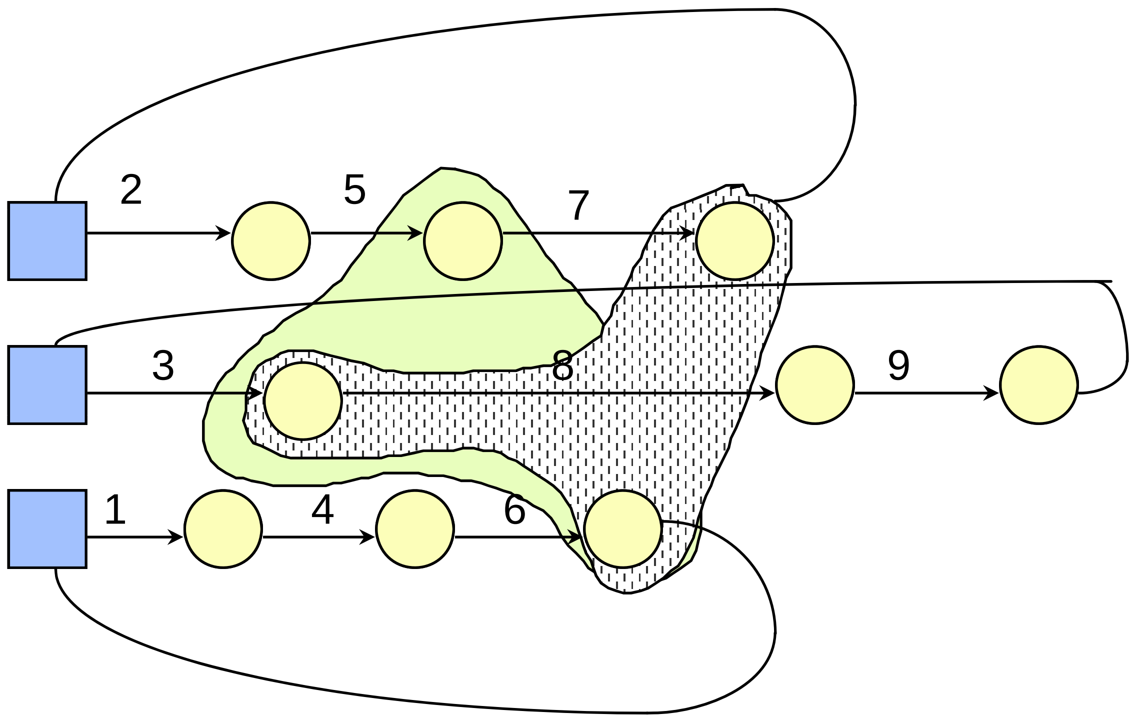
# Cyclic Snake



# Example Code

```
choose_method(i,Mach,Tasks,Rect,End):-
        append(Mach,Tasks,Obj_list),
        Objs =..[f|Obj_list],
        min_max((cyclic_snake(Objs,Mach),draw(Rect,Tasks)),End,0,2880,0,60).

cyclic_snake(Objs,[]).
cyclic_snake(Objs,[X|Frontier]):-
        fix_b(X@succ,X@setup),
        cyclic_snake_cont(X,Objs,Frontier).

cyclic_snake_cont(X,Objs,Rest):-
        X@succ > 4,!,
        arg(X@succ,Objs,Obj),
        once(indomain(Obj@start)),
        append(Rest,[Obj],Frontier),
        cyclic_snake(Objs,Frontier).
cyclic_snake_cont(_,Objs,Rest):-
        cyclic_snake(Objs,Rest).
```

# Multi-snake



# Example Code

```
choose_method(k,Mach,Tasks,Rect,End):-
        append(Mach,Tasks,Obj_list),
        Objs =..[f|Obj_list],
        min_max((multi_snake_end(Objs,Mach), draw(Rect,Tasks)),End,0,2880,0,60).

multi_snake_end(Objs,[]).
multi_snake_end(Objs,[F|Frontier]):-
        multi_snake_end_select(X,[F|Frontier],Rest),!,
        fix_b(X@succ,X@setup),
        multi_snake_end_cont(X,Objs,Rest).

multi_snake_end_cont(X,Objs,Rest):-
        X@succ > 4,!,
        arg(X@succ,Objs,Obj),
        once(indomain(Obj@start)),
        multi_snake_end(Objs,[Obj|Rest]).
multi_snake_end_cont(_,Objs,Rest):-
        multi_snake_end(Objs,Rest).

multi_snake_end_select(A,B,C):-
        list_attr_sort(B,[A|C],end,end).
```

# Results

- Strategies used
  - a: conventional labeling
  - b: successor, most constrained on succ, assign setup
  - c: successor, smallest on succ, assign setup
  - d: successor, most constrained on setup, assign setup
  - e: successor, smallest on setup, assign setup
  - f: continuation
  - g: continuation, selection on smallest start
  - h: snake
  - i: cyclic snake
  - j: multi snake on start
  - k: multi snake on end

# Parameters

- Optimization criteria: min End time
  - aims at balancing lines
- Optimization time limit 60s
- run on PC Pentium 233MHZ/64 Mb
- Windows NT 4.0
- CHIP V5.1 product version

# Last solution End date

| Day | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | - | 1251 | 1334 | 1251 | 1638 | 1486 | 1546 | 1324 | 1282 | 1324 | 1251 |
| 12 | - | 1473 | 1629 | 1764 | 1783 | 1553 | 1807 | 1473 | 1488 | 1473 | 1516 |
| 13 | - | 1305 | 1290 | 1290 | 1464 | 1402 | 1492 | 1726 | 1726 | 1726 | 1535 |
| 16 | - | 1657 | - | 1561 | 1702 | - | - | 1896 | 1934 | 1896 | 1561 |
| 17 | - | 1433 | 1523 | 1509 | 1640 | - | 1741 | 1430 | 1509 | 1430 | 1450 |
| 18 | - | 1850 | 1850 | 1850 | 1940 | - | - | 1850 | 1850 | 1850 | 1850 |
| 19 | - | 1468 | 1352 | 1481 | 1523 | 1403 | 1505 | 1434 | 1371 | 1434 | 1284 |
| 20 | - | 1353 | 1559 | 1080 | 1150 | 1356 | 1047 | 1456 | 960 | 1456 | 1032 |

# Last solution Production time

| Day | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | - | 3297 | 3326 | 3324 | 3342 | 3260 | 3291 | 3333 | 3328 | 3333 | 3324 |
| 12 | - | 4328 | 4295 | 4252 | 4311 | 4346 | 4292 | 4328 | 4328 | 4328 | 4325 |
| 13 | - | 3694 | 3712 | 3712 | 3719 | 3686 | 3731 | 3781 | 3781 | 3781 | 3751 |
| 16 | - | 4179 | - | 4098 | 4122 | - | - | 4217 | 4223 | 4217 | 4122 |
| 17 | - | 4118 | 4109 | 4135 | 4091 | - | 4066 | 4118 | 4135 | 4118 | 4116 |
| 18 | - | 3893 | 3855 | 3836 | 3836 | - | - | 3893 | 3817 | 3893 | 3801 |
| 19 | - | 3452 | 3374 | 3322 | 3360 | 3378 | 3361 | 3450 | 3432 | 3450 | 3413 |
| 20 | - | 2522 | 2552 | 2401 | 2453 | 2499 | 2449 | 2541 | 2425 | 2541 | 2441 |

# Last solution Setup

| Day | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | - | 1005 | 1080 | 970 | 1410 | 1350 | 1455 | 970 | 985 | 970 | 970 |
| 12 | - | 925 | 910 | 910 | 1255 | 1090 | 1150 | 925 | 940 | 925 | 955 |
| 13 | - | 905 | 905 | 900 | 1285 | 1180 | 1180 | 900 | 900 | 900 | 915 |
| 16 | - | 1000 | - | 955 | 1495 | - | - | 970 | 955 | 970 | 970 |
| 17 | - | 950 | 985 | 945 | 1165 | - | 1225 | 945 | 945 | 945 | 960 |
| 18 | - | 845 | 845 | 845 | 1100 | - | - | 845 | 845 | 845 | 875 |
| 19 | - | 965 | 955 | 945 | 1240 | 1225 | 1285 | 945 | 960 | 945 | 975 |
| 20 | - | 745 | 745 | 745 | 910 | 850 | 880 | 745 | 760 | 745 | 760 |

# Last solution Throughput

| Day | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | - | 199 | 197 | 197 | 196 | 201 | 199 | 197 | 197 | 197 | 197 |
| 12 | - | 201 | 203 | 205 | 202 | 200 | 203 | 201 | 201 | 201 | 201 |
| 13 | - | 196 | 195 | 195 | 195 | 196 | 194 | 191 | 191 | 191 | 193 |
| 16 | - | 196 | - | 200 | 199 | - | - | 194 | 194 | 194 | 199 |
| 17 | - | 191 | 191 | 190 | 192 | - | 193 | 191 | 190 | 191 | 191 |
| 18 | - | 174 | 176 | 177 | 177 | - | - | 174 | 178 | 174 | 179 |
| 19 | - | 189 | 194 | 197 | 195 | 194 | 195 | 190 | 190 | 190 | 192 |
| 20 | - | 212 | 209 | 222 | 218 | 214 | 218 | 210 | 220 | 210 | 219 |

# Extensions

- Model shown too simplistic
- Preference on sequence
- Effect of hard/soft due date
- Storage limitation
  - from formulation/ to finished product store
  - fixed per task
  - producer/consumer
- Dynamic throughput
- Calendars

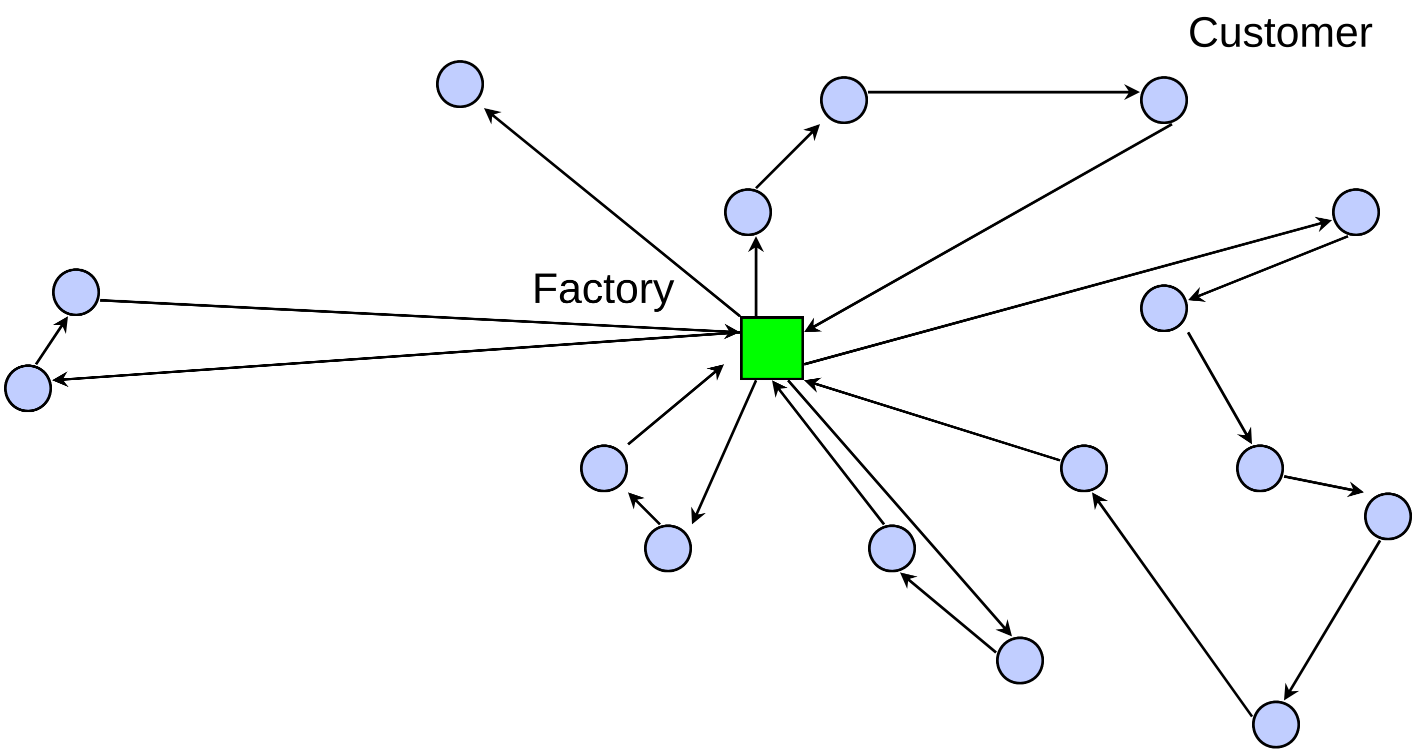
# Summary

- Different from classical scheduling
- Encountered in many environments
  - chemical industry
  - pharma
  - petro-chemical
  - food industry
  - agricultural
  - consumer products
- Not one best solution

# Example 2: Lorry Transport

- Typical bulk delivery
- Delivery trips from factory to customers
- Single source problem
- Related to Solomon problems
- Continuation of problem 1
- Surprising similarity in model to problem 1

# Process

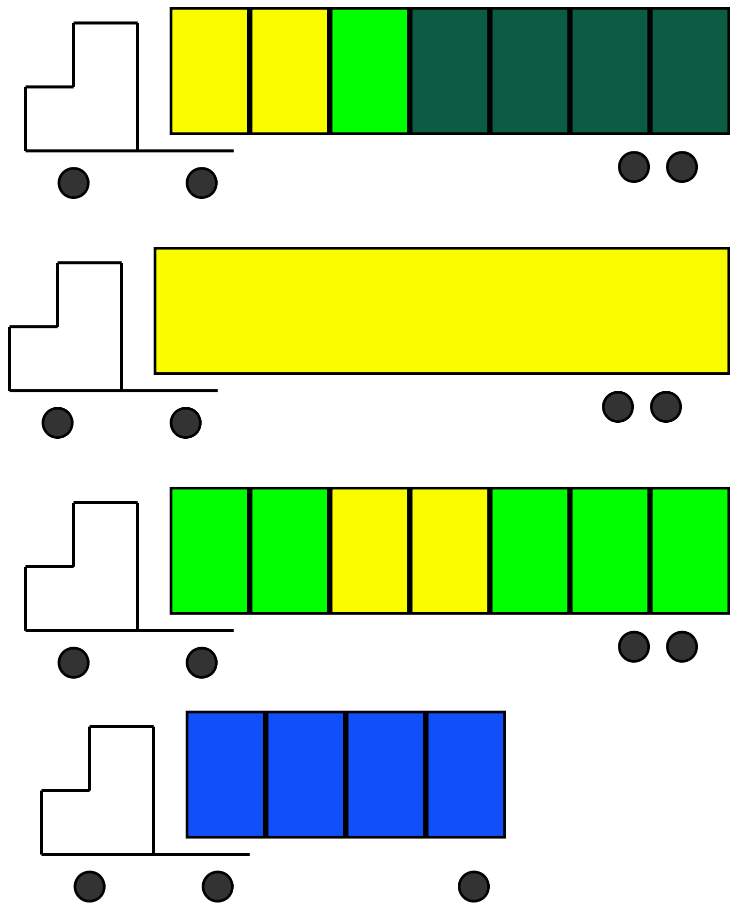


# Constraints

- lorry types (capacity, number of compartments)
- loading/ unloading times
- time windows for customer delivery
- Inter-location constraints
- accessibility of locations
- distance/ driving time
- one shift operation
- return to base
- unloading sequence (contamination)
- product size

# Lorry Types

- Non homogeneous fleet
- Compartments
  - multiple of 3 ton
  - up to 7 products
- Overall Capacity
  - 12
  - 15
  - 22



# Locations

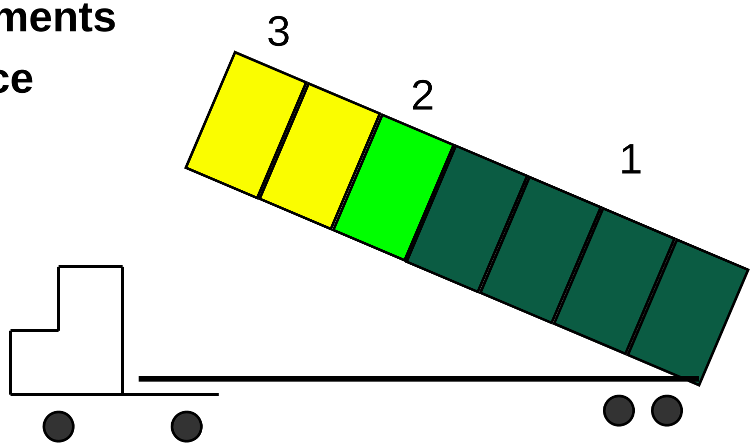
- Distributed around factory
  - different product families have different distribution
- Access restrictions
  - lorry type
  - width of road
  - turning circle
- Time windows
  - not during night
  - first/last delivery
  - rare: limited time window (more often for re-distribution problems)

# Inter-location constraints

- Distance matrix
  - based on road network
  - quite different from Euclidean distance
- Order of visit
  - multiple drops for same customer
- Partial order between locations
  - never go from this location to that
  - agricultural constraints
    - growth cycle of animals
    - disease control

# Contamination problem

- Depends on lorry type
  - sealed compartments
  - tanker
- Incompatible products
  - never transport together on same lorry
  - never put in neighboring compartments
  - contamination in delivery sequence
    - tipping sequence



# Objectives

- on-time, in full
- no contamination
- respect work rules
- min mileage
- min transport cost
- utilization of existing lorries
- cost of extra lorries hired-in
- MOT/downtime restrictions
- balance work load

# Data

- orders
- customers
- driving times
- vehicles
- factory
- products
- product size
- contamination

# Model

- Objects
- Variables
- Constraints
- Search Method

# Object model

- delivery
  - static data
    - product,
    - size,
    - qty,
    - compartments needed
- trip
  - static data
    - compartments
    - capacity
    - max mileage/driving time per day

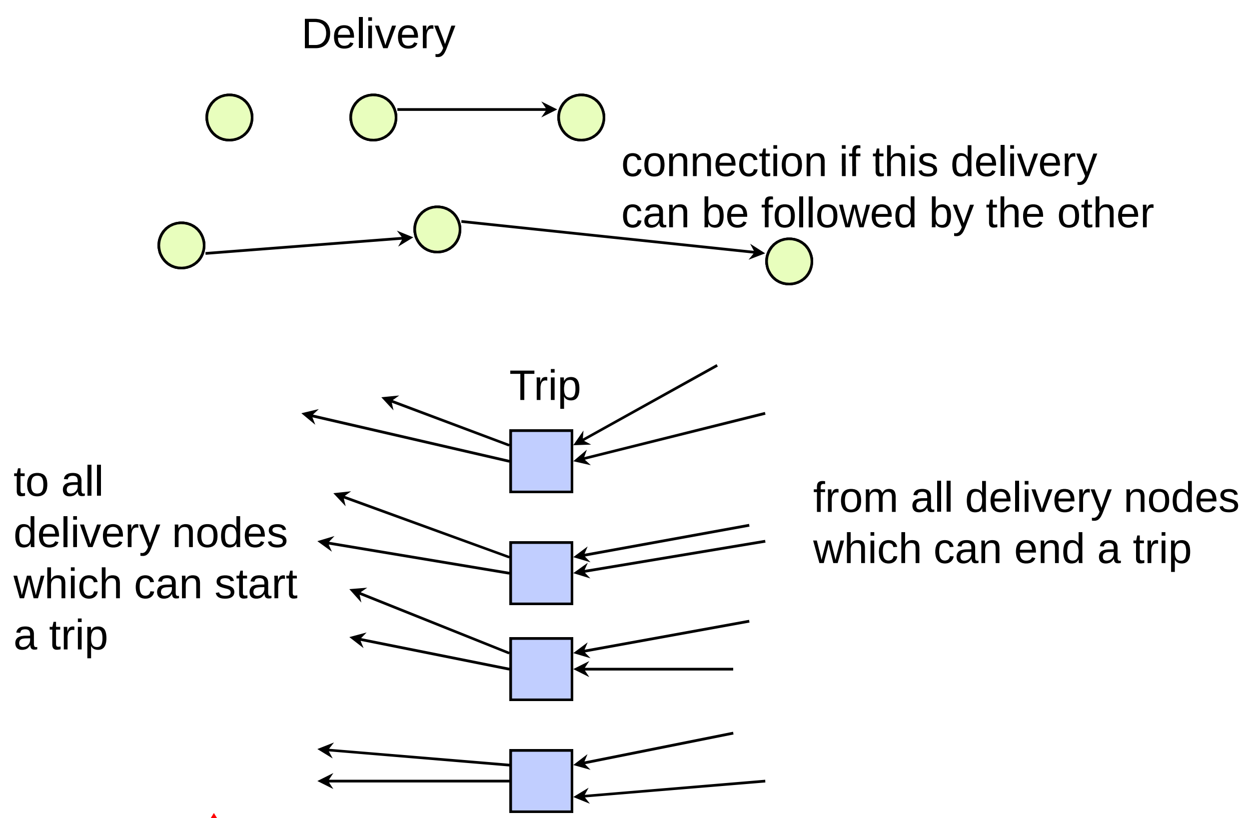
# Variables

- successors
  - contamination
  - inter-location
- weight
  - three types
    - distance to successor/driving time
    - 1 (use of one product, atleast one compartment)
    - qty (weight)
- assignment
- start times
  - corresponds to distance traveled/ arrival time at location
- work span of lorries
  - max driving time/ max distance

# Constraints

- given number of compartments per lorry
- given capacity per lorry
- max distance per lorry
- location constraints
- inter-location constraints
- contamination control

# Cycle graph


Delivery
connection if this delivery
can be followed by the other
Trip
to all
delivery nodes
which can start
a trip
from all delivery nodes
which can end a trip

# Cycle1 - Number of stops

- Weight of nodes
  - 1 (stronger: number of compartments needed)
- Capacity of special nodes
  - number of compartments in lorry
- Weight of special nodes
  - 0

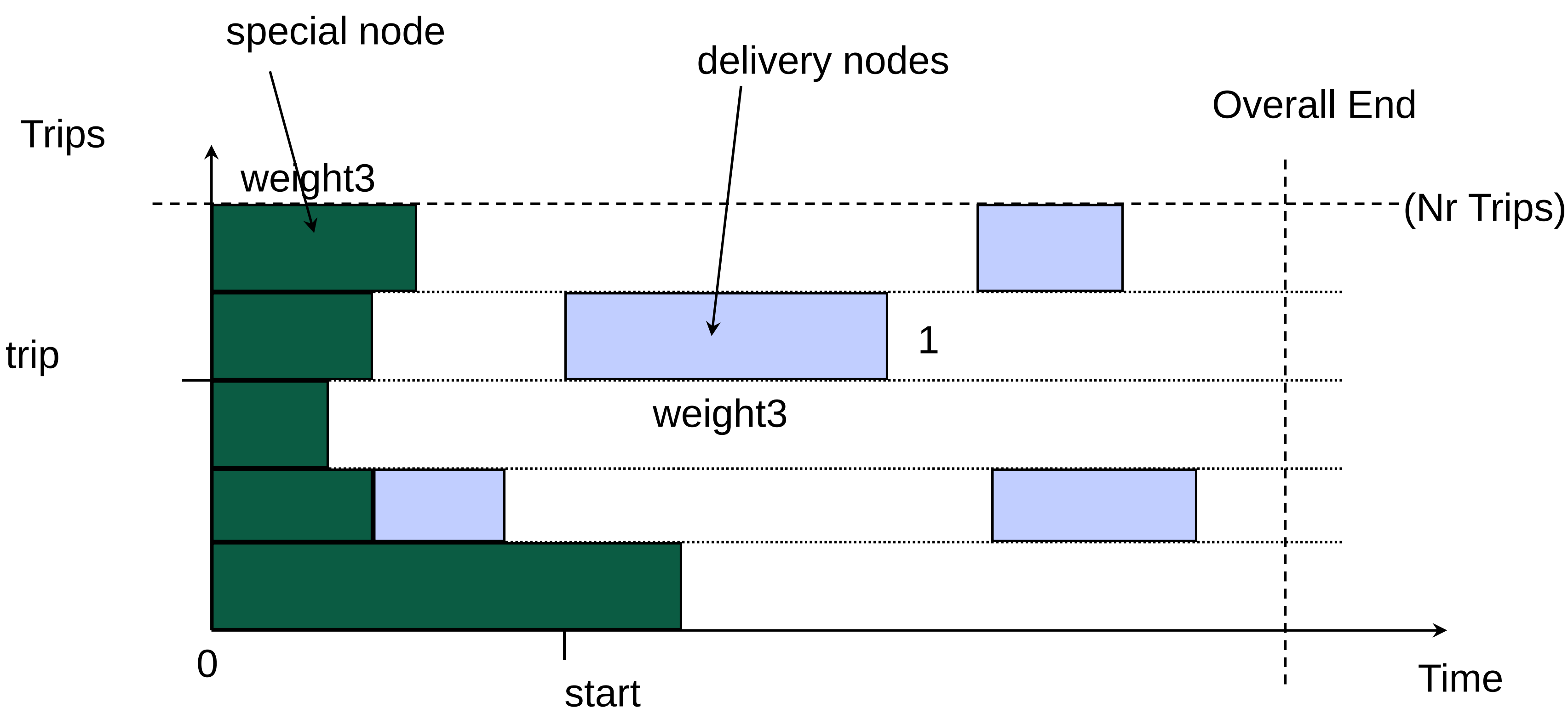
# Cycle2 - Capacity

- **Weight of nodes**
  - qty of product (stronger: rounded to nearest multiple of compartment size)
- **Capacity of special nodes**
  - capacity in tons of lorry
- **Weight of special nodes**
  - 0

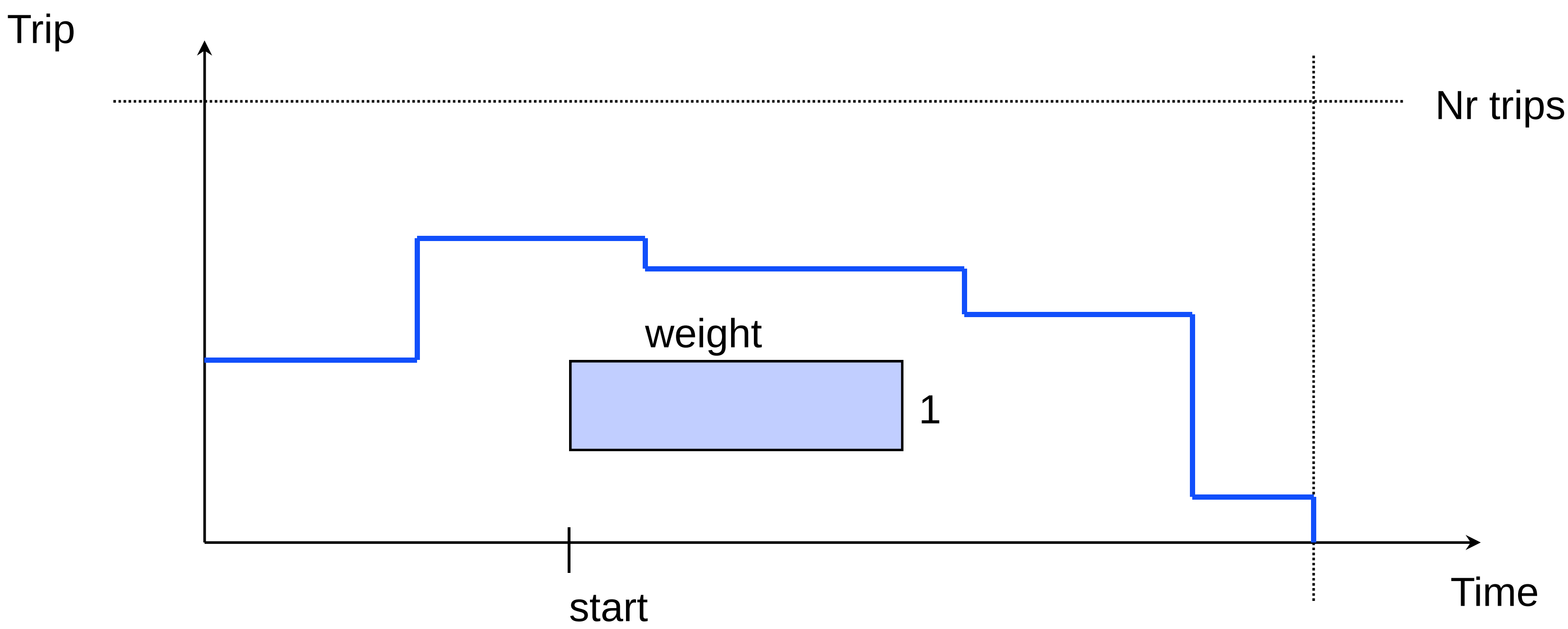
# Cycle3 - Distance

- Weight of nodes
  - defined by matrix
  - unloading time + distance to successor
- Capacity of special nodes
  - max working time
- Weight of special nodes
  - loading time + distance to successor

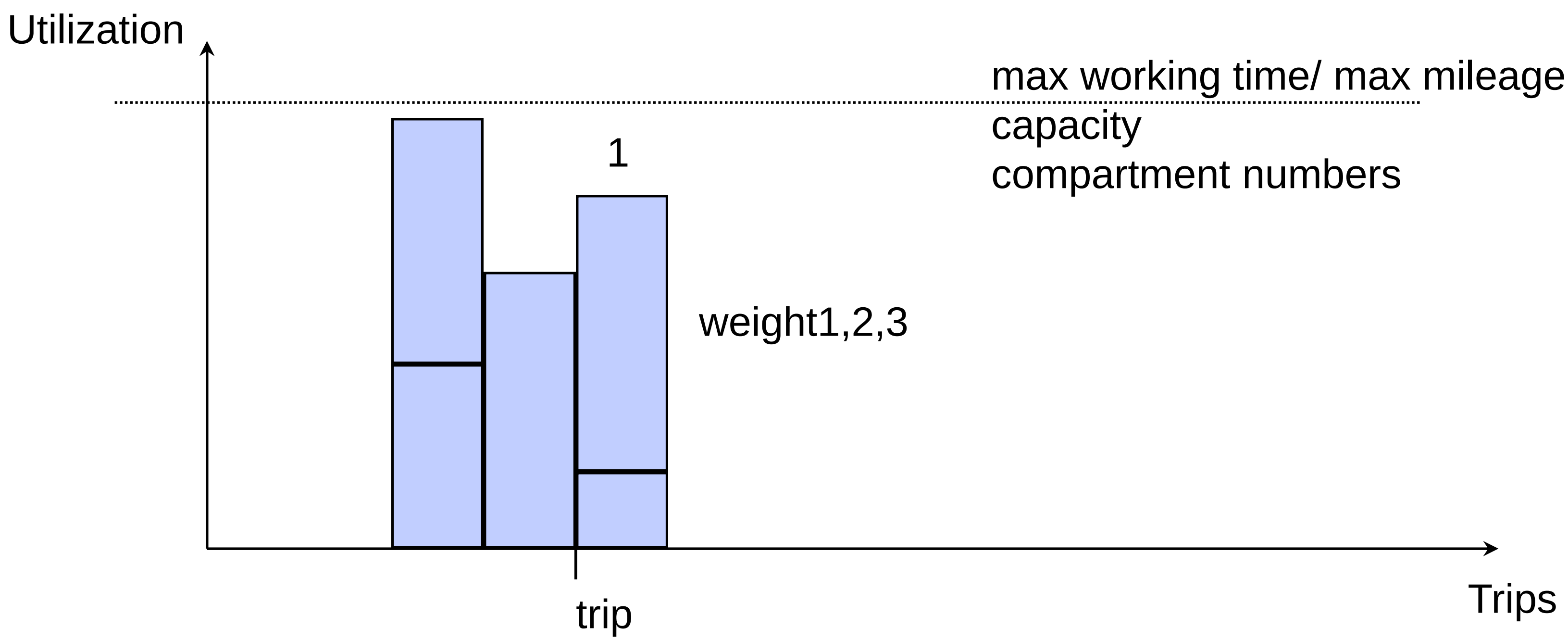
# Redundant diffn



# Redundant cumulative



# Bin packing view



# Program skeleton

```
run(Day):-
        create_objects(Day,Deliveries,Trips),
        length(Trips,M),
        length(Deliveries,N),
        N1 is N+M,
        prepare_trip_variables(Trips,N1),
        prepare_variables(Deliveries,N1,Deliveries),
        End :: 0..1000,
        Height :: 0..1000,

        set_cycle(Trips,Deliveries,N1,Ends,Lorries_used,Cost),
        project(Deliveries,[start,trip,weight3,1],Rect1),
        project(Trips,[0,trip,weight3,1],Rect2),
        append(Rect1,Rect2,Rect),
        diffn(Rect,unused,unused,[End,Height]),

        choose_method(Method,Trips,Deliveries,Rect,Ends,Lorries_used,Cost).
```

# Variable setup

```
prepare_trip_variables([],_).
prepare_trip_variables([A|A1],N):-
        A@succ :: 1..N,
        A@pred :: 1..N,
        A@trip  = A@nr,
        A@weights1 :: 0..A@compartments,
        A@weights2 :: 0..A@capacity,
        A@weights3 :: 0..1000,
        A@weight1 = 0,
        A@weight2 = 0,
        A@weight3 :: 0..1000,
        prepare_trip_variables(A1,N).
```

```
prepare_variables([],_,_).
prepare_variables([A|A1],N,Deliveries):-
        A@succ :: 1..N,
        A@pred :: 1..N,
        A@weight1 = A@compartments,
        A@weight2 is A@qty,
        A@weight3 :: 0..1000,
        A@trip :: 1..N,
        A@start :: 0..1000,
        A@end :: 0..1000,
        A@end #= A@start + A@weight3,
        prepare_variables(A1,N,Deliveries).
```

# Cycle constraints

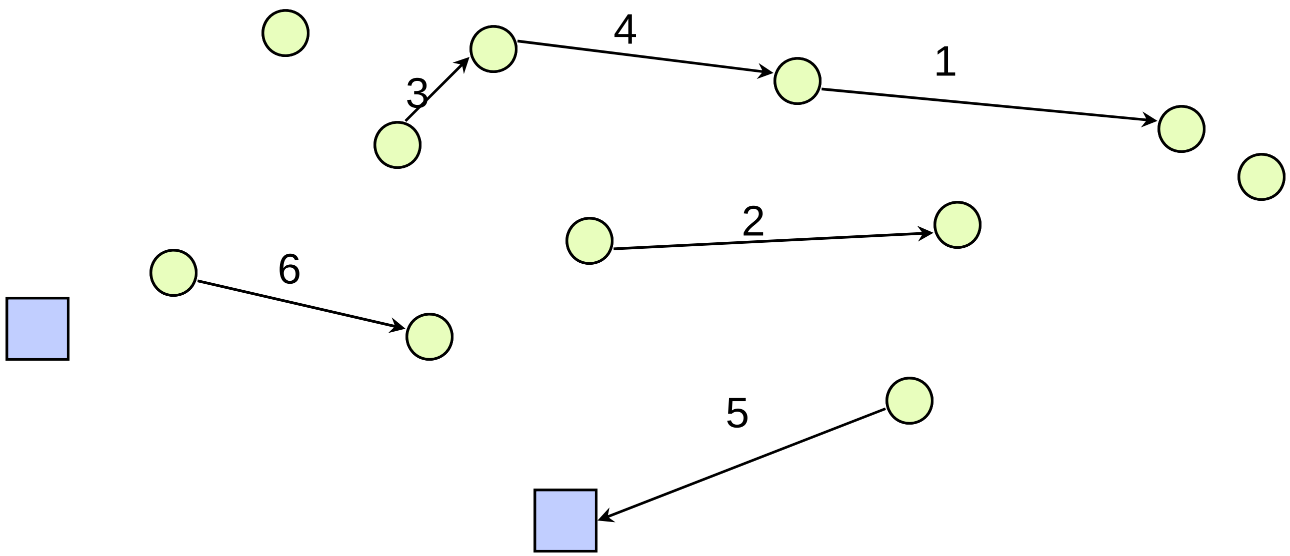
```
set_cycle(Trips,Deliveries,Size,Weights32,Trip1,Cost):-
    project(Deliveries,[[succ],[pred],[weight1],[weight2],[weight3],[trip],[start]],
        [Succ1,Pred1,Weight11,Weight21,Weight31,Trip1,Start1]),
    project(Trips,[[succ],[pred],[weight1],[weight2],[weight3],[trip],[weights1],[weights2],[weights3],[start]]
      , [Succ2,Pred2,Weight12,Weight22,Weight32,Trip2,Weights12,Weights22,Weights32,Start2]),
    length(Trips,N),
    append(Succ1,Succ2,Succ),append(Weight11,Weight12,Weight1),
    append(Weight21,Weight22,Weight2),append(Weight31,Weight32,Weight3),
    append(Trip1,Trip2,Trip),append(Start1,Start2,Start),
    append(Pred1,Pred2,Pred),
    cycle(N,Succ,Weight1,0,10000,Trip2,Weights12,Trip),
    cycle(N,Succ,Weight2,0,10000,Trip2,Weights22,Trip),
    append(Deliveries,Trips,Objs),
    create_matrix(Objs,Objs,Matrix),
    Cost :: 0..100000,
    cycle(N,Succ,Weight3,0,200,Trip2,Weights32,Trip,[#=,Start],unused,[Cost,Matrix]),
    inverse(Succ,Pred,all,all).
```

# Search Strategies

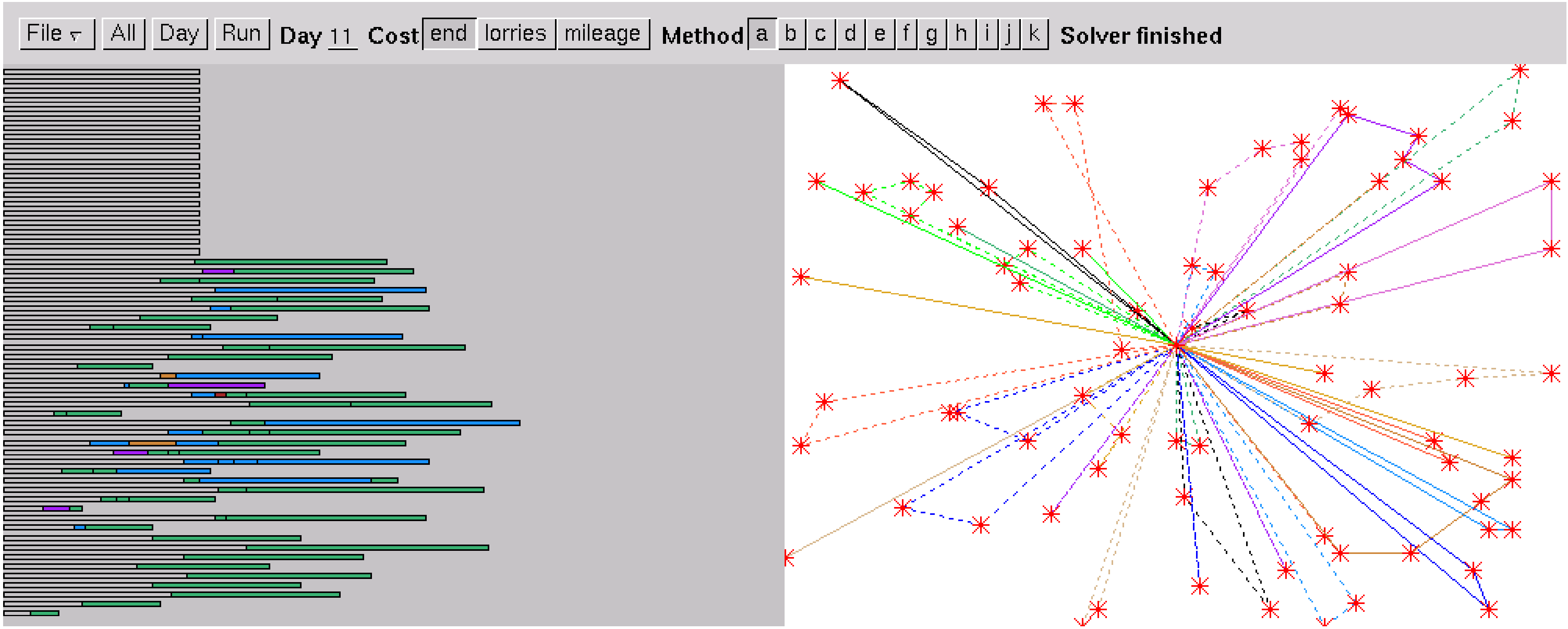
- Successor labeling
- Snake
- Multi-snake

# Successor labeling

- find node to expand by heuristic
- find successors based on heuristic
- cycles not build left to right

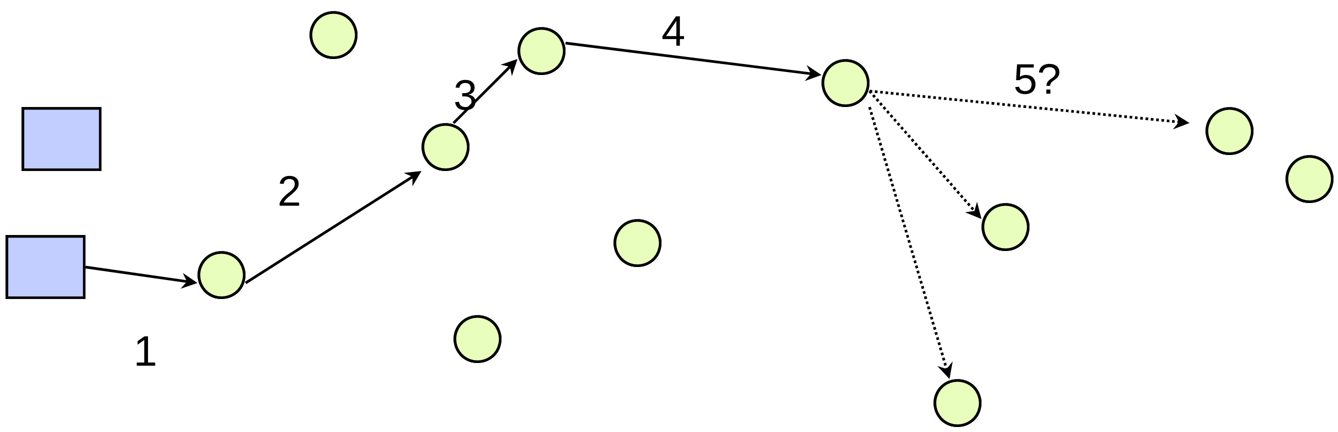


# Solution a

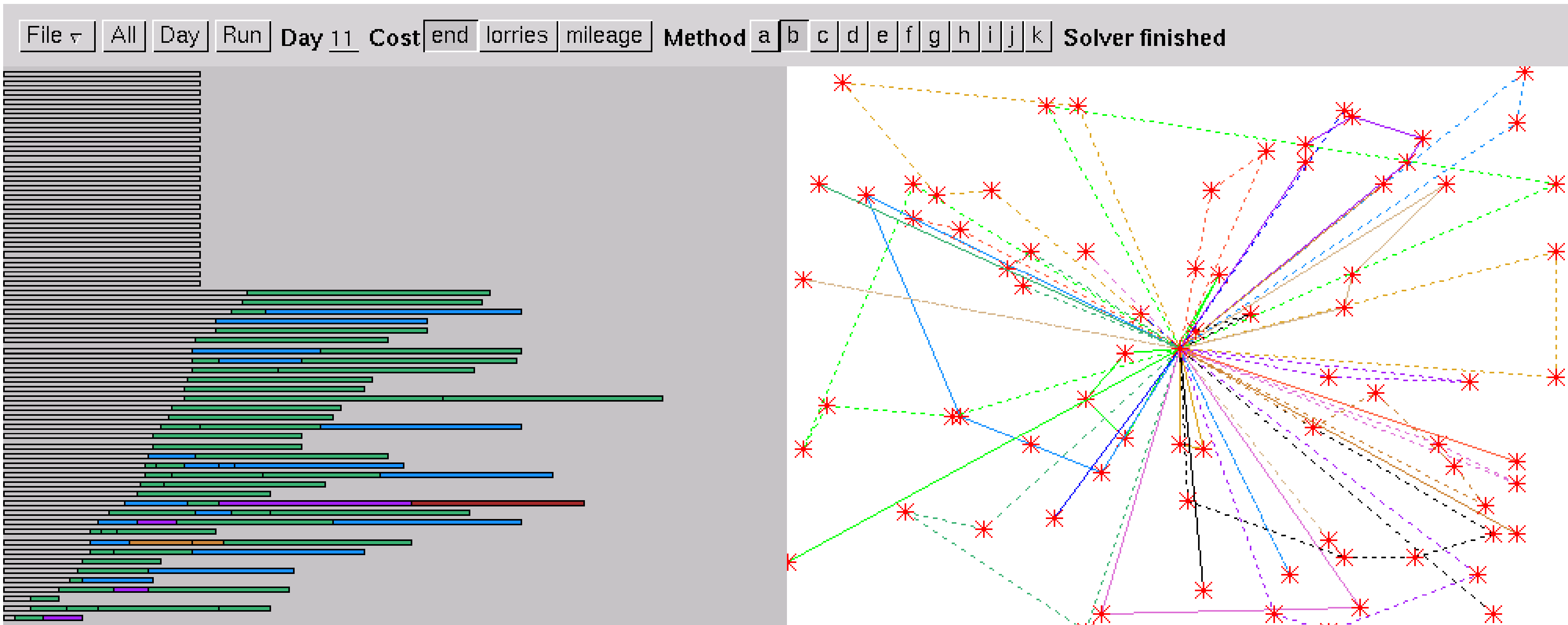

File
All
Day
Run
Day 11
Cost
end
lorries
mileage
Method
a
b
c
d
e
f
g
h
i
j
k
Solver finished

# Snake

- Only one cycle “open” at a time
- Always select end of current line to expand
  - switch to new cycle when previous cycle closed
- Select successor with heuristic
  - e.g. use node with minimum distance

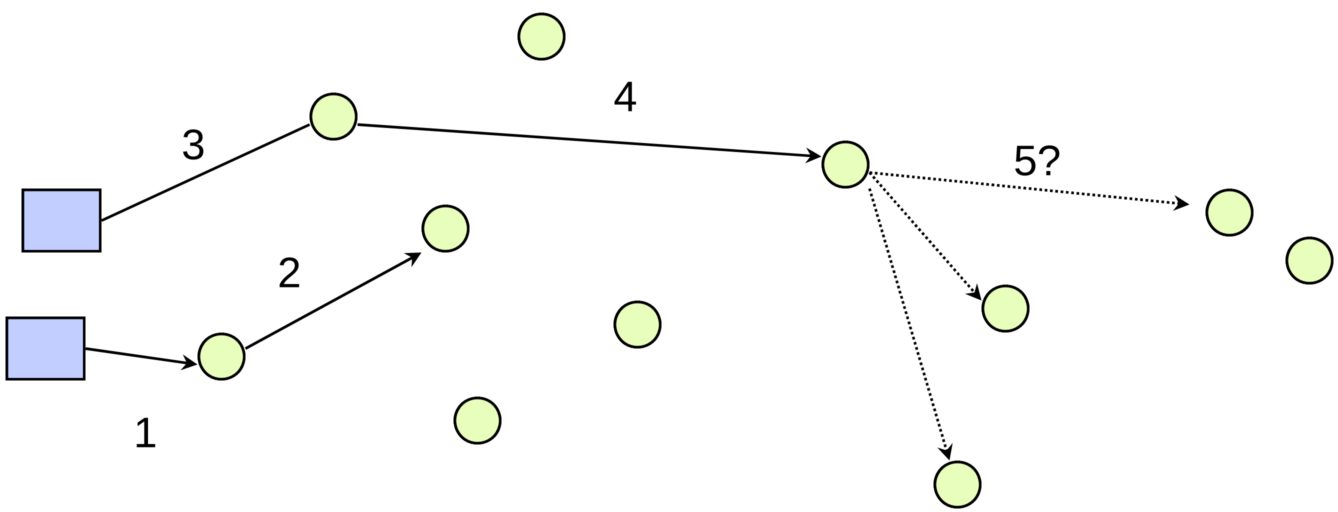


# Solution b

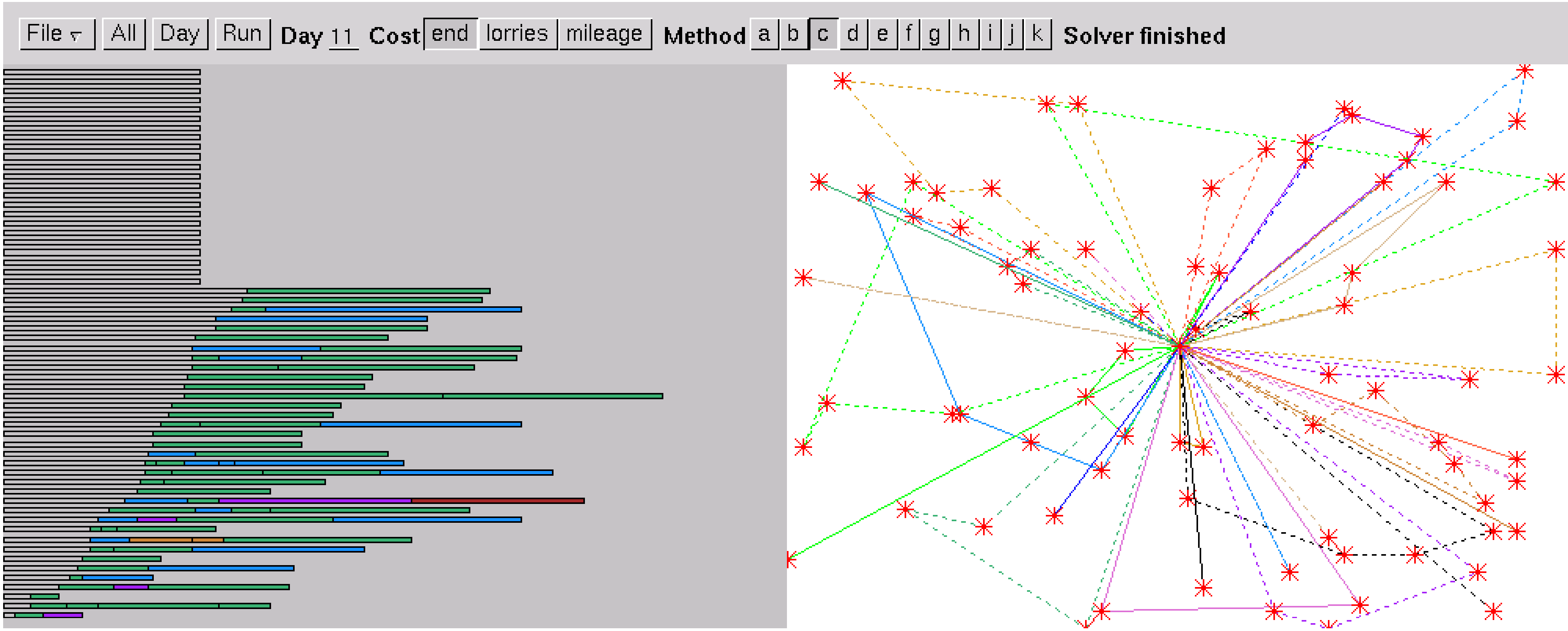

File
All
Day
Run
Day 11
Cost
end
lorries
mileage
Method
a
b
c
d
e
f
g
h
i
j
k
Solver finished

# Multi-snake

- select which line to expand based on heuristic (deterministic)
- select successor with heuristic (non-deterministic)
- multiple cycles “open” at same time

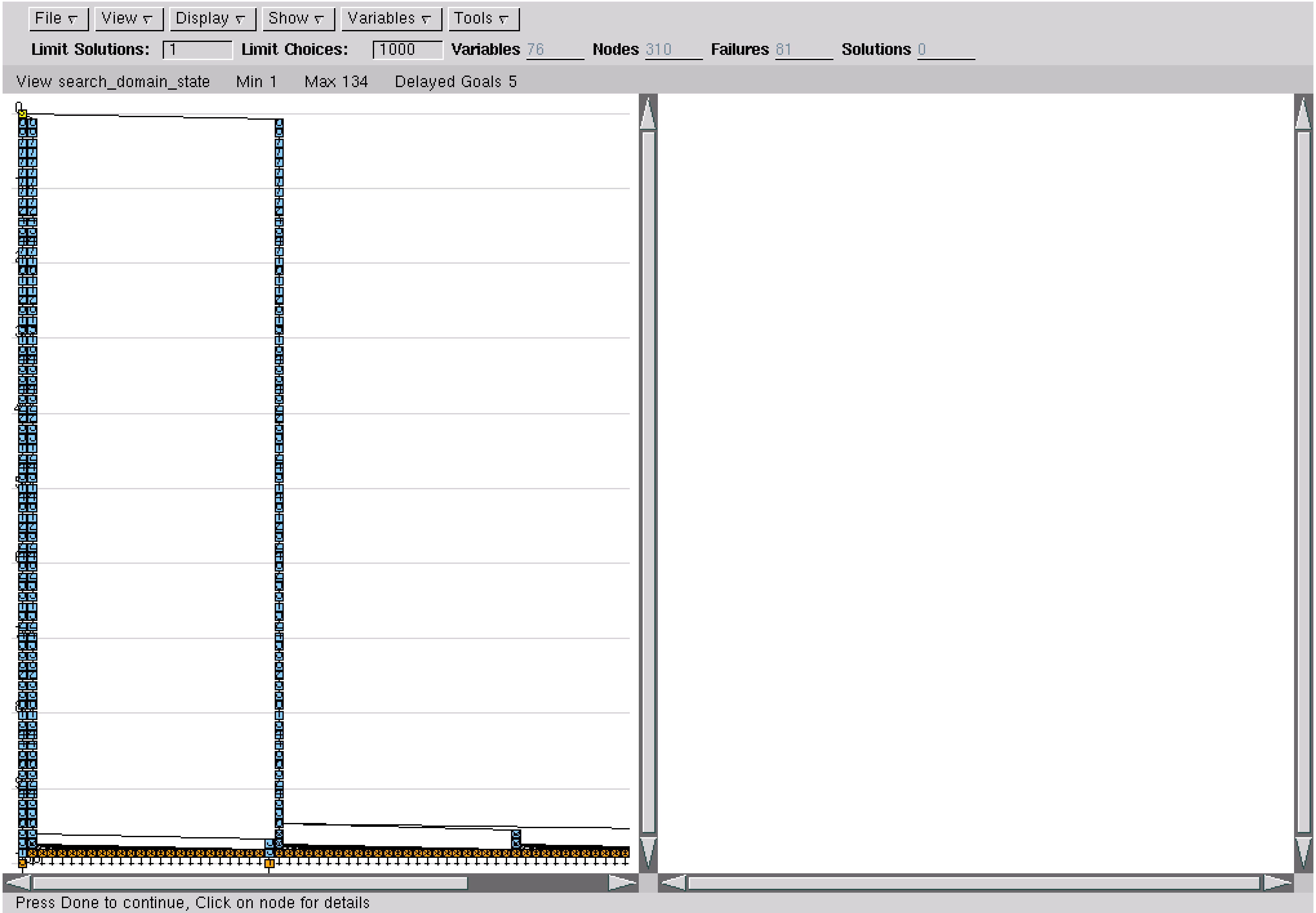


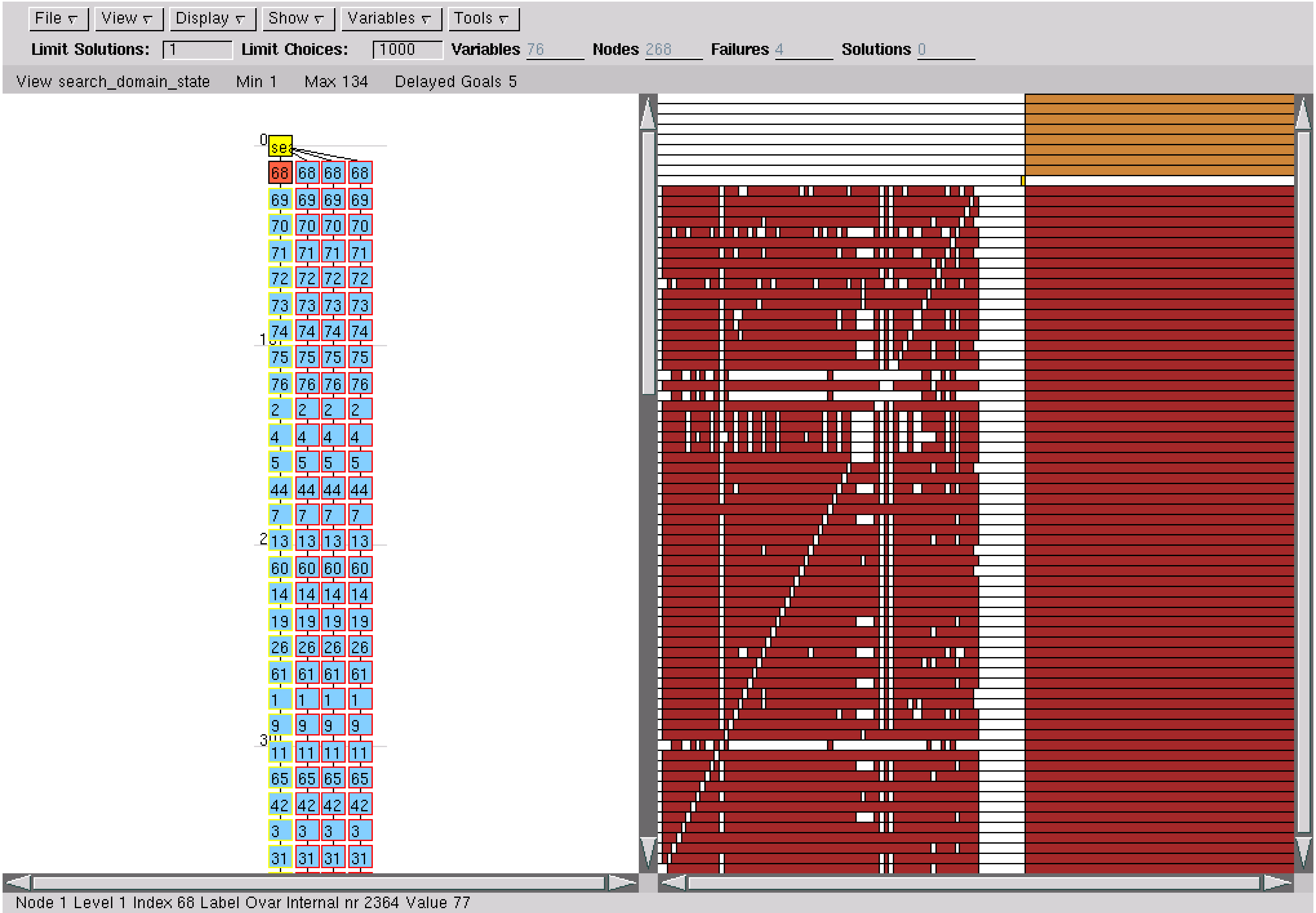
# Solution c

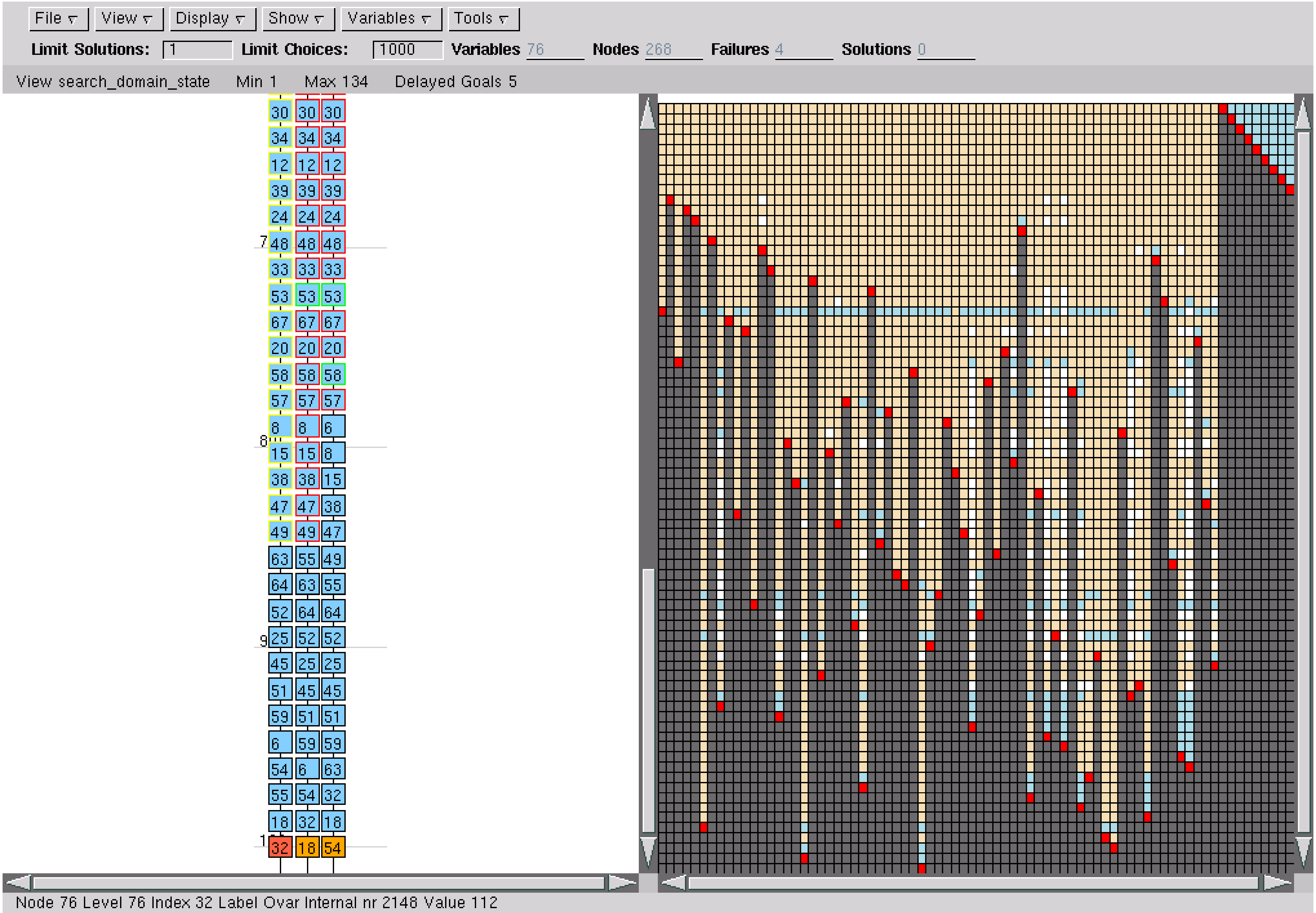

File
All
Day
Run
Day 11
Cost
end
lorries
mileage
Method
a
b
c
d
e
f
g
h
i
j
k
Solver finished

# Intuition

- **Successor labeling**
  - build tours which low mileage
  - may need more vehicles
- **Snake**
  - get most out of a resource in work done
  - may be quite bad in mileage objective
- **Multi-snake**
  - use all resources equally well
  - works best with high ratio delivery/trip
- **Bin packing strategies**
  - ignore location continuity
  - success depends on data set


File
View
Display
Show
Variables
Tools
Limit Solutions: 1
Limit Choices: 1000
Variables 76
Nodes 310
Failures 81
Solutions 0
View search_domain_state
Min 1
Max 134
Delayed Goals 5
Press Done to continue, Click on node for details


File
View
Display
Show
Variables
Tools
Limit Solutions: 1
Limit Choices: 1000
Variables 76
Nodes 268
Failures 4
Solutions 0
View search_domain_state
Min 1
Max 134
Delayed Goals 5
Node 1 Level 1 Index 68 Label Ovar Internal nr 2364 Value 77



File
View
Display
Show
Variables
Tools
Limit Solutions: 1
Limit Choices: 1000
Variables 76
Nodes 268
Failures 4
Solutions 0
View search_domain_state
Min 1
Max 134
Delayed Goals 5
Node 76 Level 76 Index 32 Label Ovar Internal nr 2148 Value 112

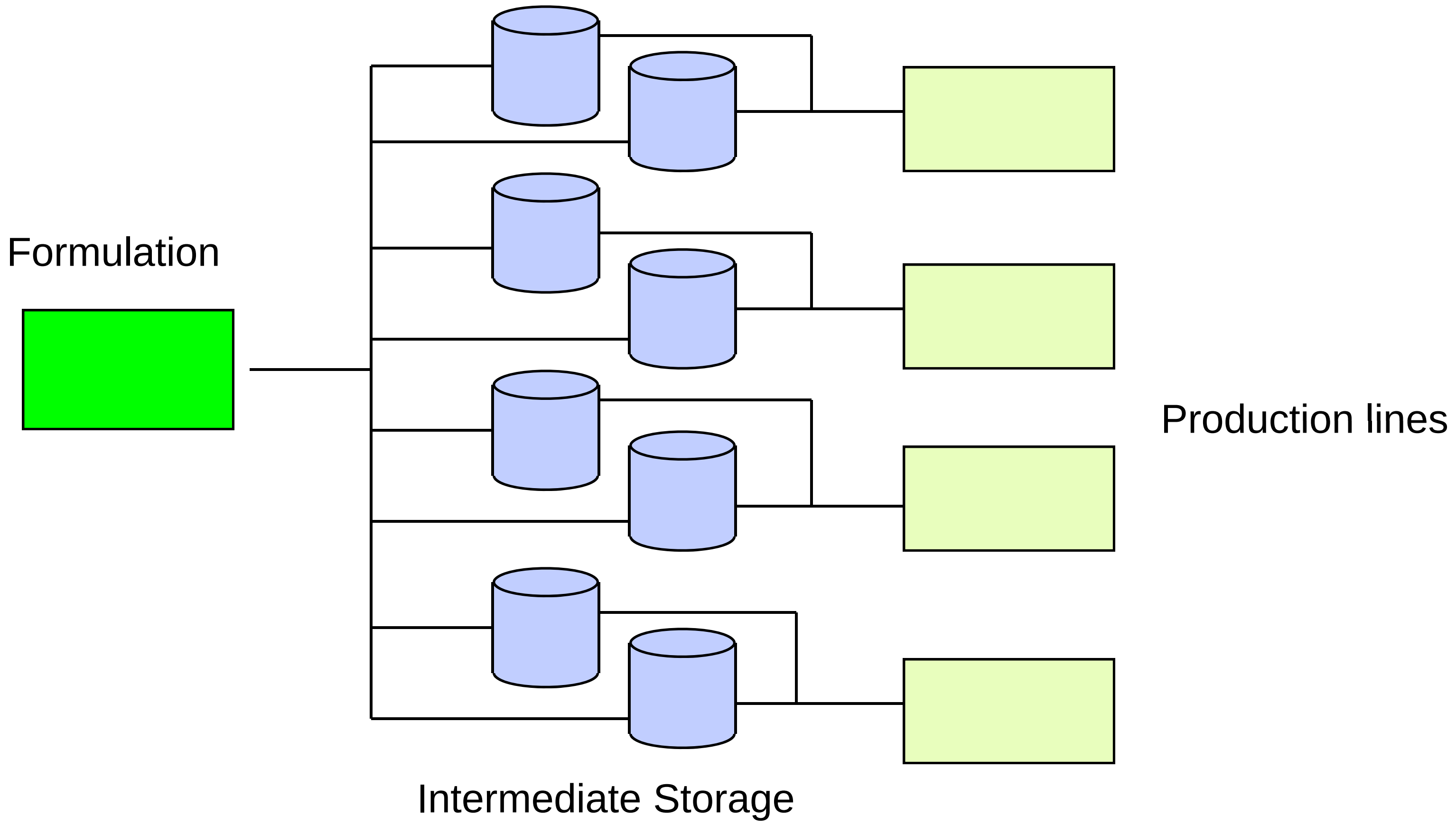
# Extensions

- Possible back loads
- Multiple factories
- Multiple trips
  - complete working day/days
  - wave model
- Owned/hired lorries
- Work rules for drivers

# Example 3: Production Sequencing

- Produce batches of product for use on production lines
- Assume full qty must be available before starting line task
- Line schedule run before, start times known
- Batch based production at N batches/hr
- Slot based timing
  - if slot is missed, this batch must be skipped
- Limited storage capacity for intermediate products

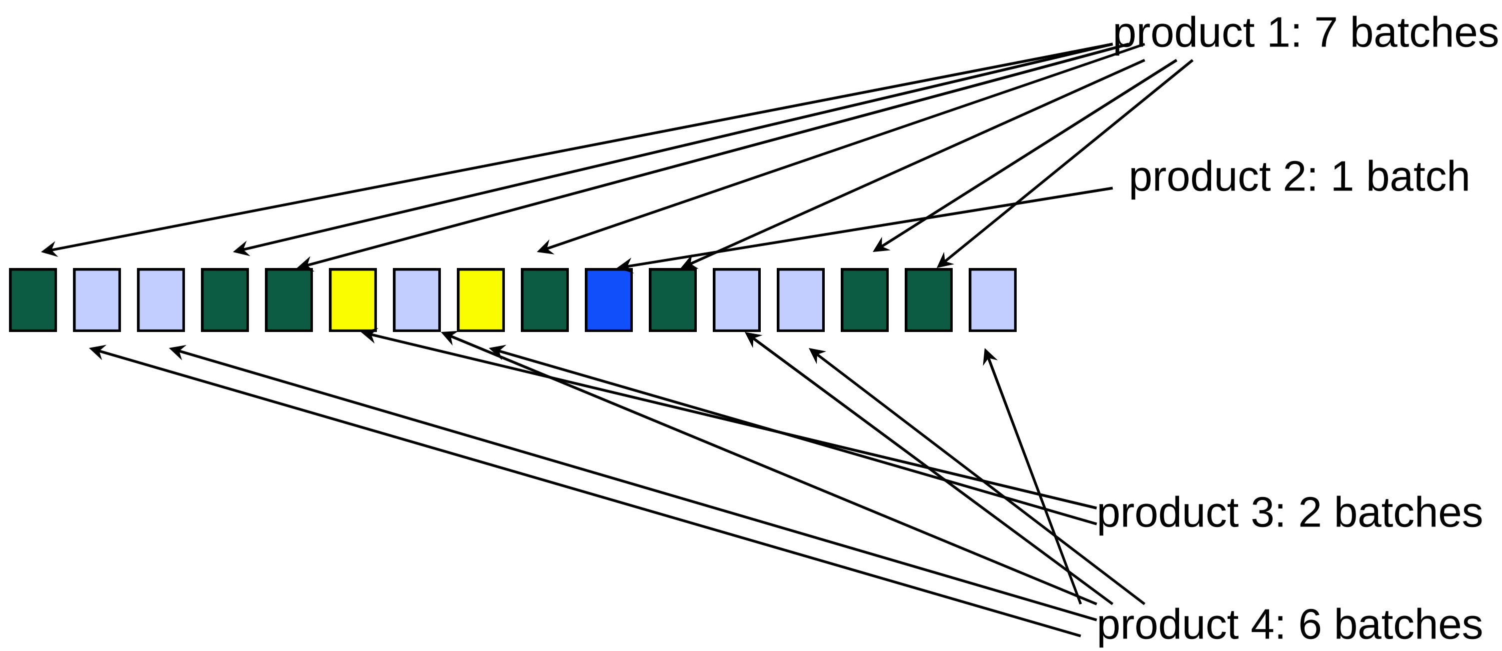
# Process



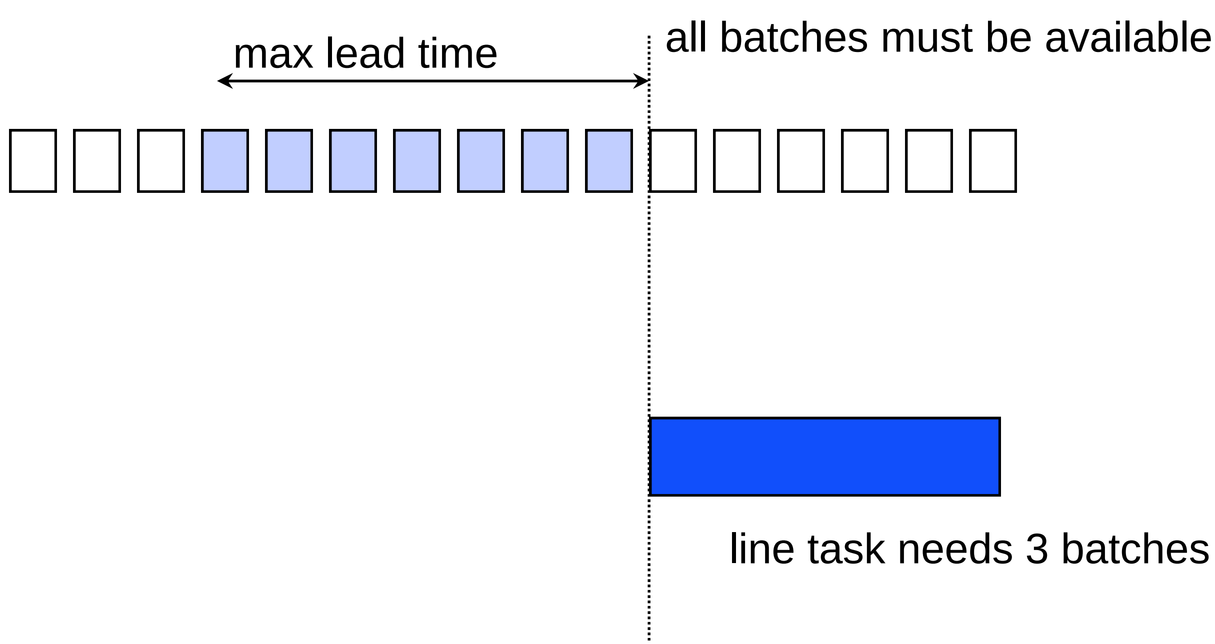
# Constraints

- Correct number of batches for each product to be made
- Batches for order must be made before start of line task
- Not too early as this needs too  much storage
- No contamination problem in sequence
- Making multiple batches for same product is easier (preference)

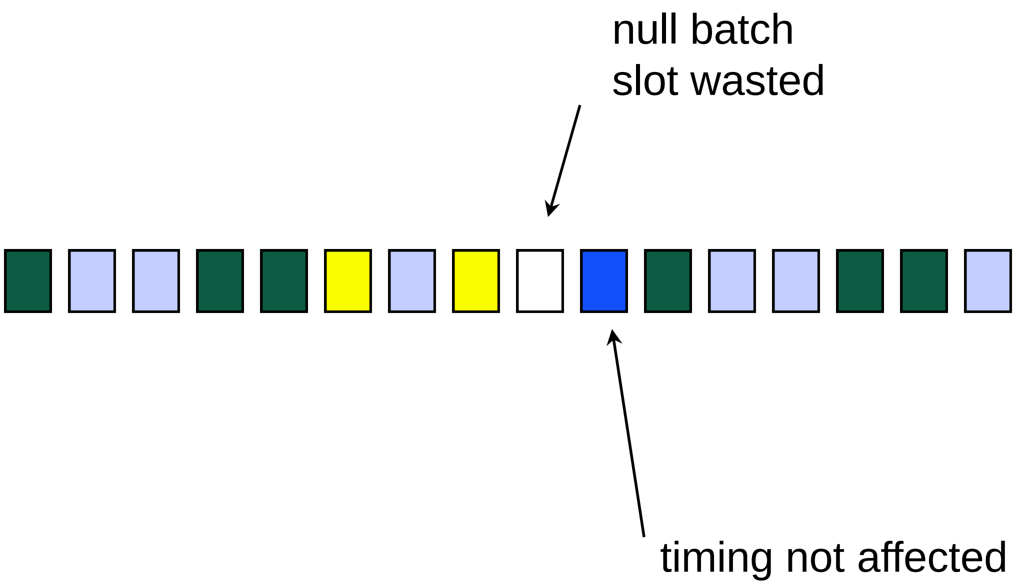
# Number of batches



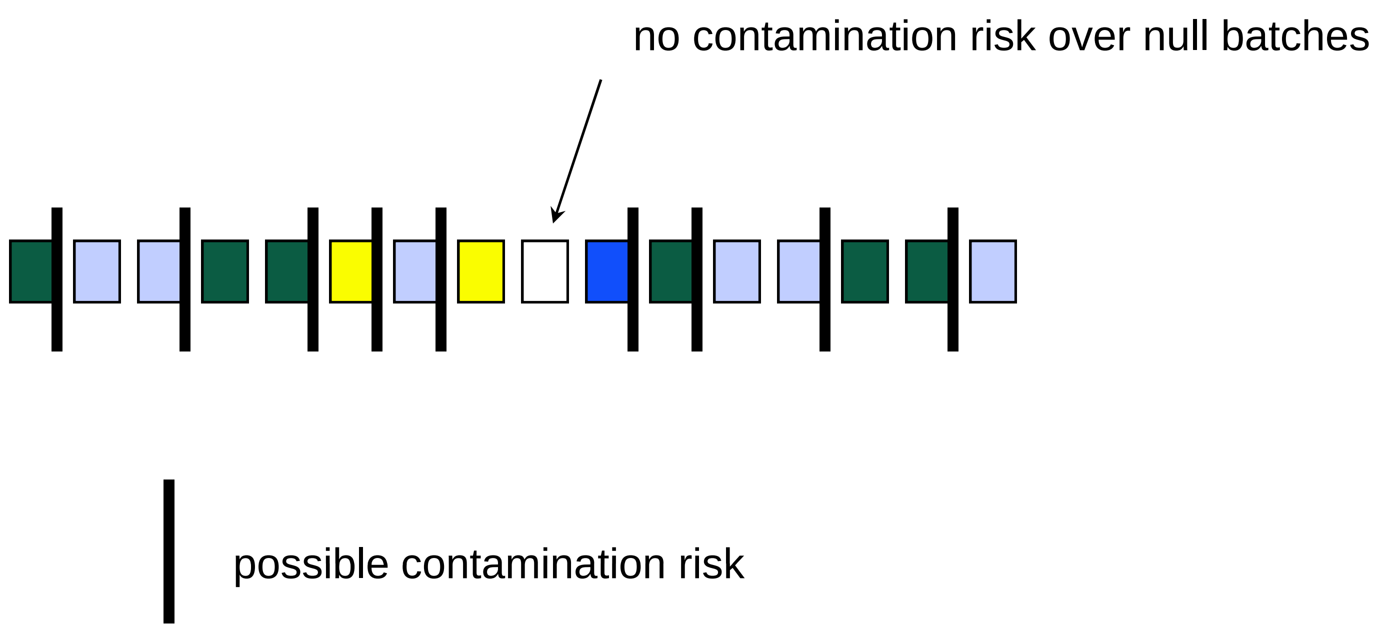
# Time windows



# Null batches



# Contamination



# Objectives

- In-time production
- No contamination
- Min stock of intermediate product
- Repeated batches

# Data

- Orders
  - product,
  - size (ignored),
  - qty,
  - day,
  - time required(from line schedule)
- Batch size
- Batch duration
- Contamination information

# Model

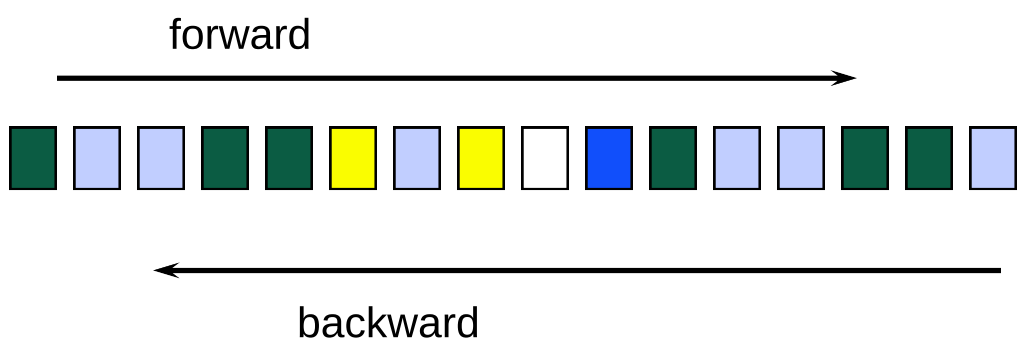
- **Variables**
  - one variable per batch
  - domain over all possible products/ null (empty) batch special value
- **Constraints**
  - one among constraint with multiple support to express demand
  - one among constraint per order to express time window
  - one sequence constraint to express forbidden successors
- **Search method**
  - forward/backward labeling

# Alternative model

- Cycle based model possible
  - one cycle describes sequence of products
  - duration one for all batches
  - time windows correspond to position in cycle
- Better contamination control/propagation of contamination
- Performance problem with many batches

# Search Strategies

- **Assign forward**
  - **safety margins (+)**
  - **stock level (-)**
- **Assign backward**
  - **stock level (+)**
- **Selection based on heuristic**



# 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
  - snake family for problems with cycle constraint
  - often possible to improve by using custom programming
- Test cases show some stability of model for varying data
  - very important for practical applications

# Production Scheduling

- Semi-process industry scheduling
- Alternative machines
- Varying speed / quality
- In-time production
- Setup / Sequencing constraints
- Starting point for multiple extensions
- Uses diffn, cumulative, cycle
- Not for campaign based production (cycles)
  - difference in importance of stock cost / shelf life

# Lorry Transport

- Distribution problem
  - one source, multiple sinks
  - trip based
  - sequence control
- Model very similar to problem 1

# Production Sequencing

- Slot based batch production
- Simple intermediate stock view
- Unit duration
- Model with among / sequence constraints
- Alternative model with cycle

## Problems discussed

- Stand assignment
  - aircraft parking assignment
- Resource restricted scheduling
  - Alvarez problems
- Personnel assignment
  - nurse scheduling
- Flight rotation planning
  - plane rotations
- Production scheduling
  - semi-process industry
- Distribution scheduling
  - bulk delivery
- Production sequencing
  - batch based production

# Constraints used

| | cumulative | diffn | cycle | precedence | among | sequence | inverse |
|---|---|---|---|---|---|---|---|
| apache | | X | | | | | |
| alvarez | X | | | X | | | |
| nurse | | | | | X | X | |
| flight | | | X | | | | |
| production scheduling | X | X | X | | | | X |
| transport | X | X | X | | | | X |
| sequencing | | | | | X | X | |

# Next steps

- Evaluate results
  - try different CLP systems
    - data available
  - test other methods
  - more test data
- Wish list
  - crew scheduling
  - layout problems

# Thanks

- helpful comments
  - A. Aggoun
  - N. Beldiceanu
  - E. Bourreau
  - V. Bauche
- experience from past and current projects
  - P. Kay
  - P. Charlier
  - T. Cornelissens
- for the loan of the projector
  - P. Deransart