A Constraint Modelling Pipeline: Abstract Specifications to Optimized Constraint Models

PETER NIGHTINGALE

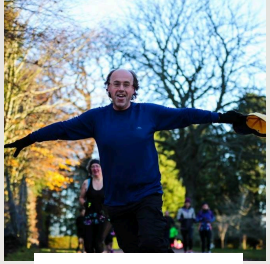
University of St Andrews | FOUNDED 1413

UNIVERSITY of York

Universitat de Girona

Özgür Akgün

Ian Miguel

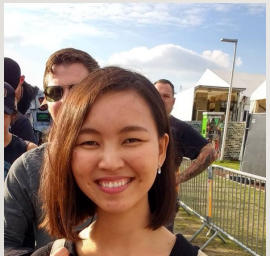


Ian Gent

Chris Jefferson

Alan Frisch

Saad Attieh



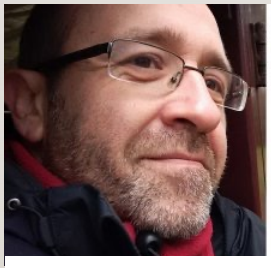
Nguyen Dang

Patrick Spracklen

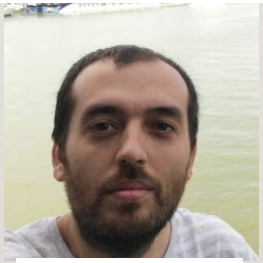
Andrea Rendl

Andras Salamon

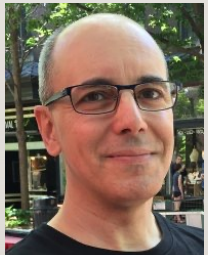
Felix Ulrich-Oltean



Mateu Villaret



Jordi Coll



Miquel Bofill

Camera shy:
Carlos Ansótegui
Josep Suy
Juan Luis Esteban

INTRODUCTION – PART 1 OF THE TALK

- Motivating *high-level* modelling and solving of combinatorial problems
  - Capturing and exploiting problem structure such as partitions, sets, functions (with nesting)
- Introducing the **Essence** language
- Brief introduction to **Conjure**
  - Refines Essence to a lower-level language, Essence Prime
  - Systematic generation of alternative models
- Direct solving of high-level Essence models using the **Athanor** local search solver

INTRODUCTION – PART 2 OF THE TALK

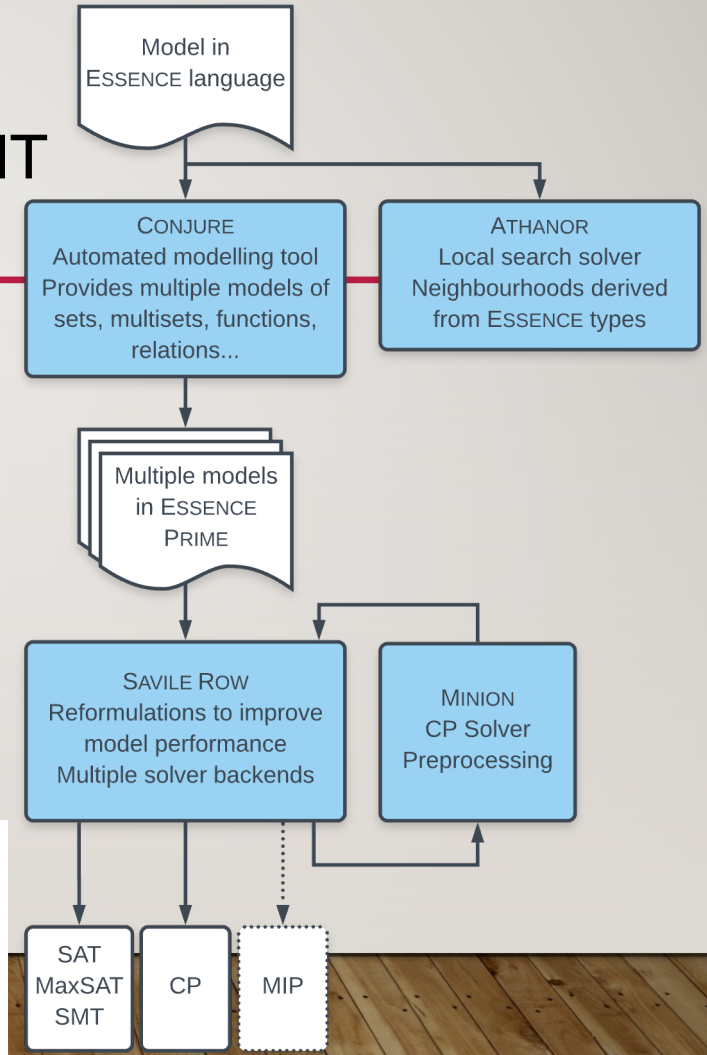
- Optimizing constraint models and encoding them for a general-purpose solver
- Introduce the **Savile Row** tool
  - Instantiates and unrolls solver-independent model
  - Reformulations for solver performance
- Case study: Optimized SAT encoding of the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP)

INTRODUCTION – A CONSTRAINT MODELLING PIPELINE

- **Conjure + Savile Row** together are a constraint modelling pipeline
- Translates high-level models in Essence down to solver-level (SAT, CP, MIP)

```
given object new type enum
given weight, value : function (total) object --> int(1..)
given maxWeight: int(1..)
find knapsack : set of object
maximising sum i in knapsack . value(i)
such that (sum i in knapsack . weight(i)) <= maxWeight
```

```
p cnf 7076 47899
-1 2 0
1 -2 3 0
-3 2 0
-3 -1 0
...
```



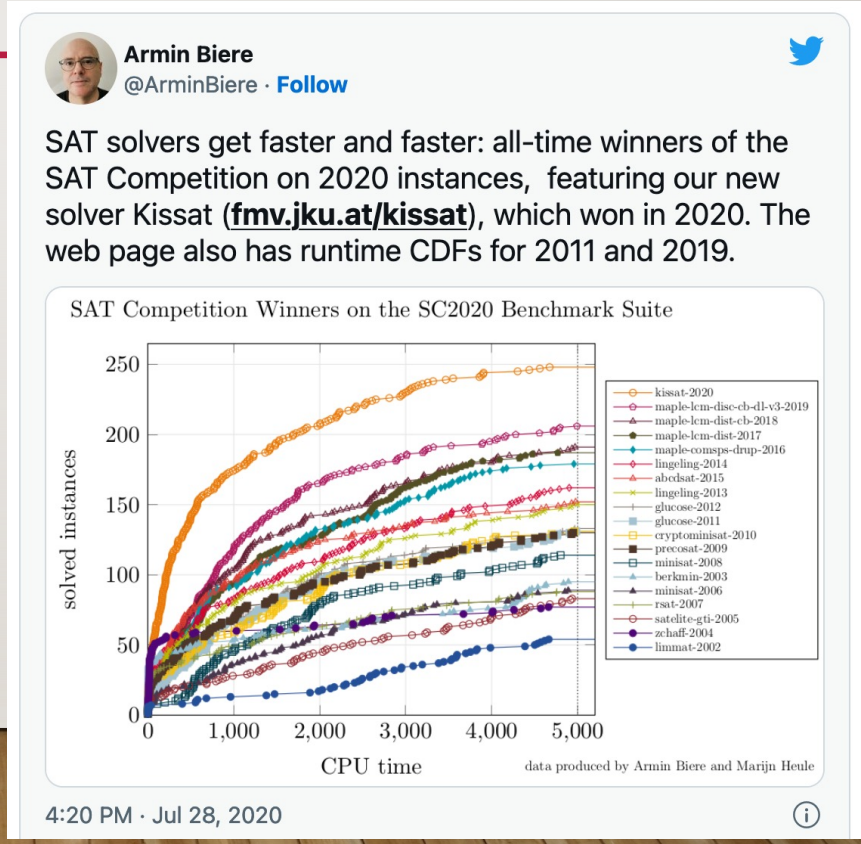
CONSTRAINT PROGRAMMING BACKGROUND: MODELLING AND SOLVING

- **CP = Modelling + Solving**
- Constraint *solving* (broadly interpreted) includes:
  - Mixed Integer Programming (MIP): linear relaxation, search, and cuts
  - SAT, SMT, MaxSAT: Conflict-directed clause learning
  - CP solvers: search, propagation, and conflict learning
  - Local search, metaheuristics
- Solvers developed by IBM, Google, Intel, Gurobi, and many more

CONSTRAINT PROGRAMMING BACKGROUND: MODELLING AND SOLVING

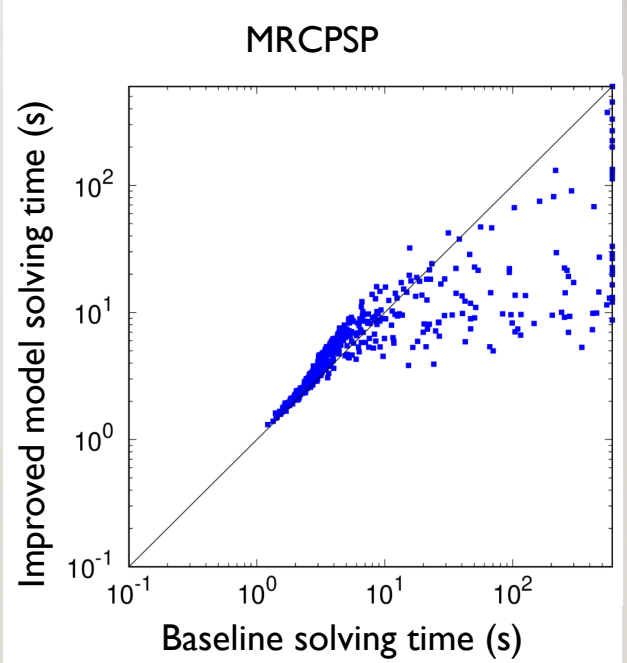
- **Remarkably powerful, decades of performance improvements**
- SAT solvers improved every year from 2002 to 2020 [1]
- MIP solvers became **1.1 million** times faster from 1991-2015 [2]

[1] http://fmv.jku.at/kissat/
[2] https://people.bath.ac.uk/masjhd/Others/Bixby2015a.pdf



CONSTRAINT PROGRAMMING BACKGROUND: MODELLING AND SOLVING

- **However, a good model is essential to unlock the power of the solvers**
  - For example, solving scheduling problem MRCPSP with SAT (case study later in this talk)
  - Strong baseline model, already better than other exact methods
  - Improved model scales much better, 10+ times speed-ups for hard instances
- CP research focuses on **both** modelling and solving
  - Modelling languages have been an important part of that

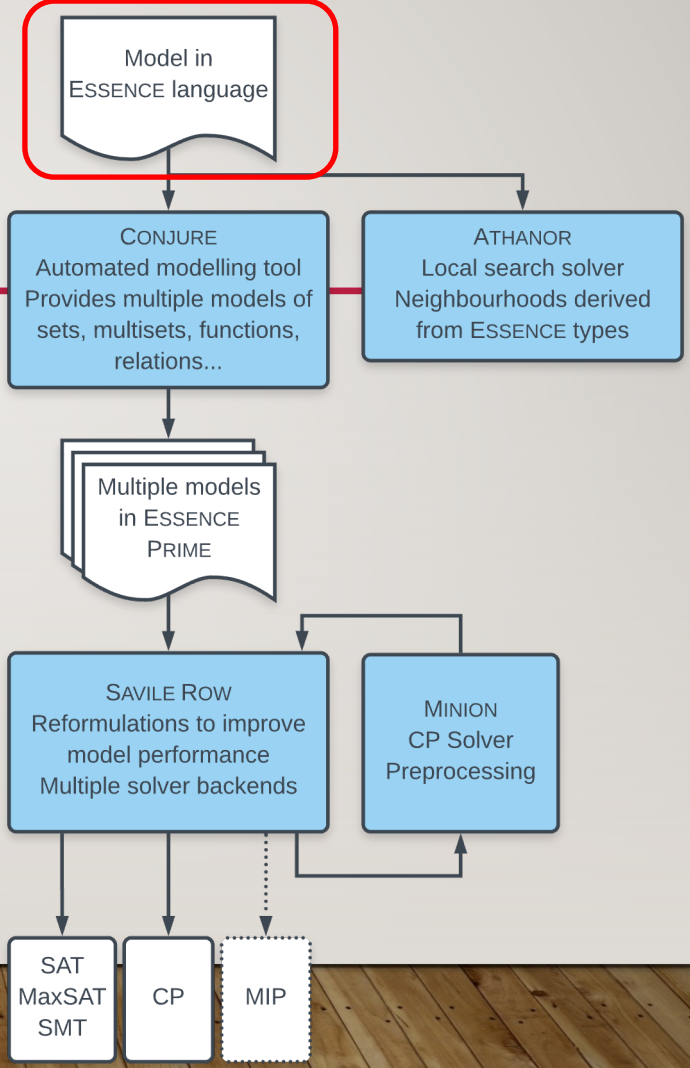


THE ESSENCE LANGUAGE

A language for high-level modelling of combinatorial problems

ESSENCE OVERVIEW

- Define a problem *class* consisting of:
  - Problem class parameters
  - Decision variables
  - The optimization function
  - Constraints
- The domain (type ++) of decision variables is the key to high-level modelling
  - Capture the problem above the level where (some) modelling decisions are made
  - Refine down to multiple models



DOMAINS IN ESSENCE

- Rich domain language is vital for capturing combinatorial structure in Essence
  - Int, bool, enumerated, matrix
  - Set, multiset
  - Sequence (of variable length)
  - Function, Relation
  - Partition
  - Graph (work in progress)
- Arbitrary nesting of these types
  - `set of set of ...` (SONET problem, for example)
  - `set of partition from ...` (Social Golfers problem)

ESSENCE EXAMPLES: KNAPSACK

- Problem class parameters: `given`
- Decision variable: `find` (set of objects)
- Optimization: `maximising`
- Constraint: `such that`

```
given object new type enum
given weight, value : function (total) object --> int(1..)
given maxWeight: int(1..)
find knapsack : set of object
maximising sum i in knapsack . value(i)
such that (sum i in knapsack . weight(i)) <= maxWeight
```

ESSENCE EXAMPLES: KNAPSACK

- Problem class parameters: `given`
- Decision variable: `find` (set of objects)
- Optimization: `maximising`
- Constraint: `such that`

```
given object new type enum
given weight, value : function (total) object --> int(1..)
given maxWeight: int(1..)
find knapsack : set of object
maximising sum i in knapsack . value(i)
such that (sum i in knapsack . weight(i)) <= maxWeight
```

ESSENCE EXAMPLES: KNAPSACK

- Problem class parameters: given
- Decision variable: find (set of objects)
- Optimization: maximising
- Constraint: such that

```
given object new type enum
given weight, value : function (total) object --> int(1..)
given maxWeight: int(1..)
find knapsack : set of object
maximising sum i in knapsack . value(i)
such that (sum i in knapsack . weight(i)) <= maxWeight
```

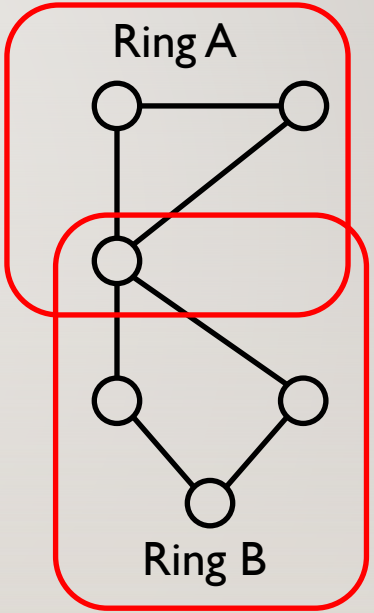
ESSENCE EXAMPLES: KNAPSACK

- Problem class parameters: given
- Decision variable: find (set of objects)
- Optimization: maximising
- Constraint: such that

```
given object new type enum
given weight, value : function (total) object --> int(1..)
given maxWeight: int(1..)
find knapsack : set of object
maximising sum i in knapsack . value(i)
such that (sum i in knapsack . weight(i)) <= maxWeight
```

ESSENCE EXAMPLES: SONET PROBLEM

- A classic network design problem (simplified version)
- Given:
  - A set of nodes
  - A set of pairs of nodes that must be connected (*demand pairs*)
- Find:
  - A set of fibre-optic rings (sets of nodes), such that every demand pair is connected via at least one ring
- Optimize:
  - Minimize the number of node-to-ring connections



ESSENCE EXAMPLES: SONET PROBLEM

- Key type is a set of sets of the decision variable `network`
- Outer set is an *unordered* collection of rings – set type avoids introducing symmetry
  - Compare to `matrix` which imposes an order on its elements
- Each inner set represents one ring

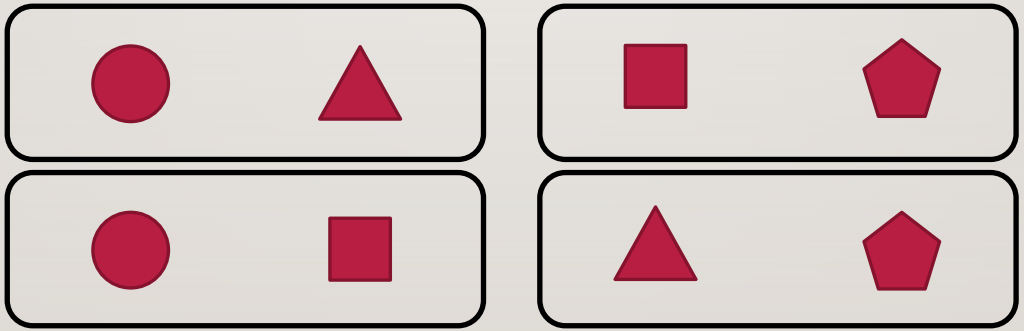
```
given nnodes, nrings, capacity : int(1..)
letting Nodes be domain int(1..nnodes)
given demand : set of set (size 2) of Nodes
find network : set (maxSize nrings) of
               set (minSize 2, maxSize capacity) of Nodes
minimising sum ring in network . |ring|
such that forAll pair in demand .
    exists ring in network . pair subsetEq ring
```

ESSENCE EXAMPLES: SOCIAL GOLFERS PROBLEM

- Every week, golfers divide themselves into *g* groups of size *s*
- Maximally social schedule – each person meets an entirely new set of people each week

Week 1

Week 2



ESSENCE EXAMPLES: SOCIAL GOLFERS PROBLEM

- Every week, golfers divide themselves into *g* groups of size *s*
- Maximally social schedule – each person meets an entirely new set of people each week
- **Partition** type (in place of `set of sets`) allows high-level modelling of together/apart
  - Special-purpose partition *representations* (in Conjure and Athanor)
  - Special type annotations, functions (e.g. `together(...)`)

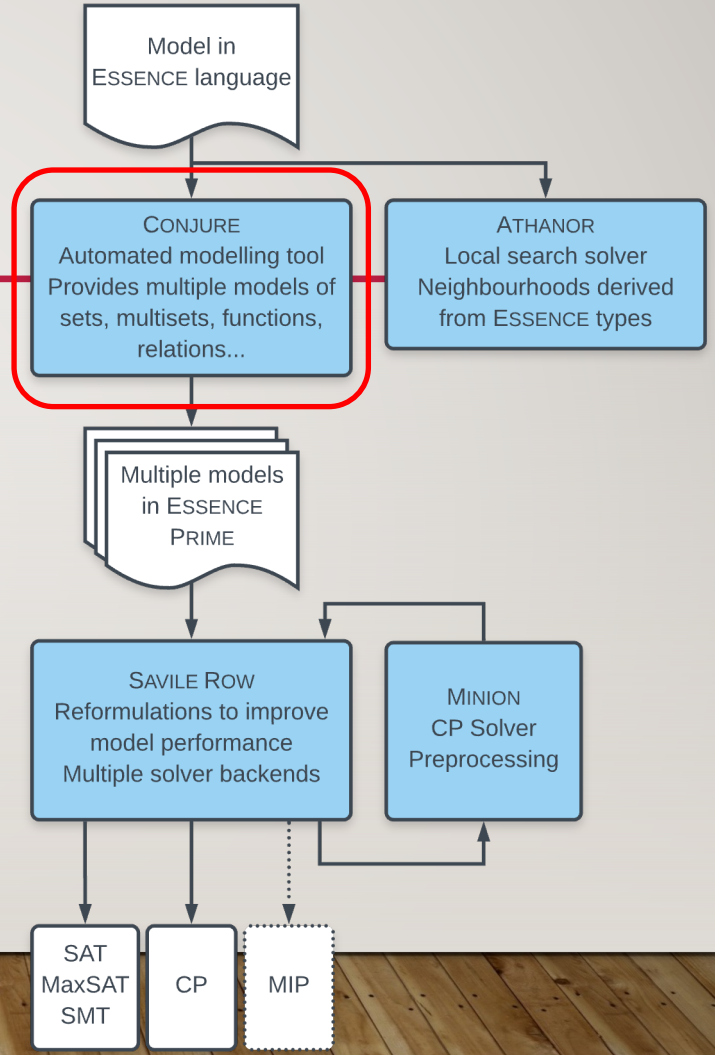
```
given w, g, s : int(1..)
letting Golfers be new type of size g * s
find sched : set (size w) of
             partition (regular, numParts g, partSize s)
             from Golfers
such that forAll g1, g2 : Golfers, g1 != g2 .
    (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

CONJURE

Refining Essence specifications to constraint models

CONJURE

- Refines Essence down to solver-independent models in lower-level language Essence Prime
- Allows systematic exploration of modelling choices
  - Representation of decision variables (viewpoints)
  - Representation of parameters
  - Formulation of constraints/objective
- Automatically generates symmetry-breaking constraints
  - Essential for performance of some solvers
- Automatic *channelling* of representations



CONJURE EXAMPLE: KNAPSACK

- A set of int (after enum → int)

```
find knapsack : set of object
maximising sum i in knapsack . value(i)
```

Explicit repn of set

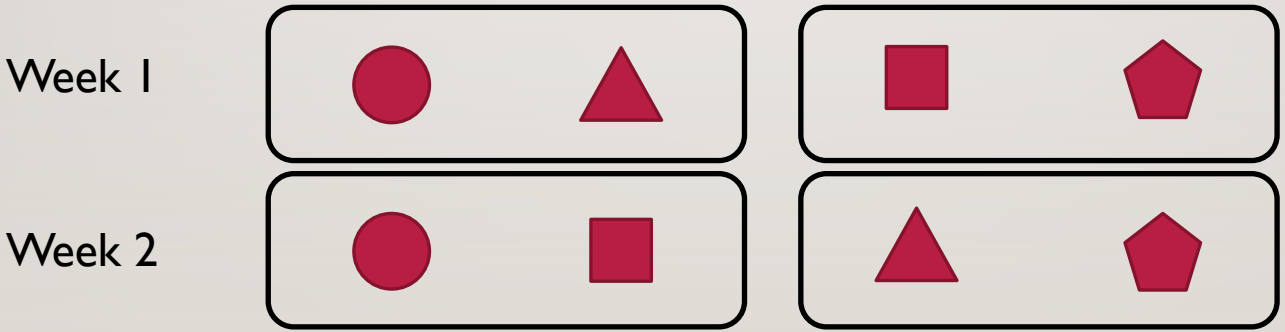
```
find knapsack : matrix [int(1..n)] of int(0..n)
maximising sum i in int(1..n).
        (knapsack[i]!=0)*value[knapsack[i]]
such that ...
```

Occurrence repn of set

```
find knapsack : matrix [int(1..n)] of bool
maximising sum i in int(1..n).
        knapsack[i]*value[i]
```

CONJURE EXAMPLE: SOCIAL GOLFERS

- A set containing partitions of golfers
- Choose representations for the outer set, and the partitions



CONJURE EXAMPLE: SOCIAL GOLFERS

- Only one practical representation for the outer set: *Explicit*

```
find sched : set (size w) of
             partition (regular, numParts g, partSize s) from Golfers
```

Explicit repn of set

```
find sched : matrix [int(1..w)] of
             partition (regular, numParts g, partSize s) from Golfers
such that ... $ symmetry breaking
```

- *Occurrence* would require numbering every possible partition!

CONJURE EXAMPLE: SOCIAL GOLFERS

- Two possible representations of partition

```
find sched : matrix [int(1..w)] of
             partition (regular, numParts g, partSize s) from Golfers
such that ...
```

PartitionAsSet

```
find sched : matrix [int(1..w)] of
             set (size g) of set (size s)
             of Golfers
such that ...
```

Occurrence

Find a cell number for each golfer.

Cells are numbered $\{1 \ldots g\}$.

```
find sched : matrix [int(1..w)] of
             matrix [int(1..g*s)] of int(1..g)
such that ...
```

CONJURE: EVALUATION

- Key question is whether Conjure generates good models (among many bad ones)
- Allows automatic model selection and model portfolio construction per problem class
  - Achieved by hand-crafted heuristics and racing
- Literature search and manual comparison of models for 42 problem classes
  - All problem classes on CSPLib.org with non-trivial decision variable domain
- Caveat: comparing model *kernels*
  - Decision variables used, constraint formulation, and symmetry-breaking constraints
  - *Not* dominance-breaking constraints, nor implied constraints

CONJURE: EVALUATION

SONET problem

- Generates from 8 to 1024 models
- Produces a published model (complete with symmetry breaking constraints) [1]

Social Golfers problem

- Generates from 3 to 9 models
- Produces a 3D matrix model from the literature [2]

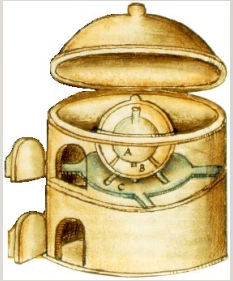
[1] B. M. Smith, Symmetry and search in a network design problem, CPAIOR 2005
[2] M. Sellmann, W. Harvey, Heuristic constraint propagation–using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem, CPAIOR 2002

THE ATHANOR SOLVER

Directly solving Essence with local search

THE ATHANOR SOLVER

- Athanor takes an entirely different approach to Conjure
  - No need to choose representation(s) or refine the model
- Constraint-Based Local Search (CBLS) lifted to Essence
  - Operates directly on assignments of Essence variables
  - Has compact representations for sets, msets, functions, relations, sequences, etc
  - Automatically derives neighbourhoods from high-level types
- An athanor is a furnace for performing alchemy



ADVANTAGES OF ATHANOR: HIGH-LEVEL NEIGHBOURHOODS

- Athanor automatically derives neighbourhoods from the decision variable types
  - Moves in neighbourhoods are meaningful at the Essence level
- For example, in the SONET problem (`set of sets`):
  1. Select an inner set, **remove an element** (improves objective)
  2. Select an inner set, **add an element** (may satisfy demand pairs)
  3. Select two sets, **move an element from one to the other**

{ {1,~~2~~,4}, {2,3,5} }

{ {1,2,4}, {2,3,4,5} }

{ {1,2,4}, {2,3,4,5} }

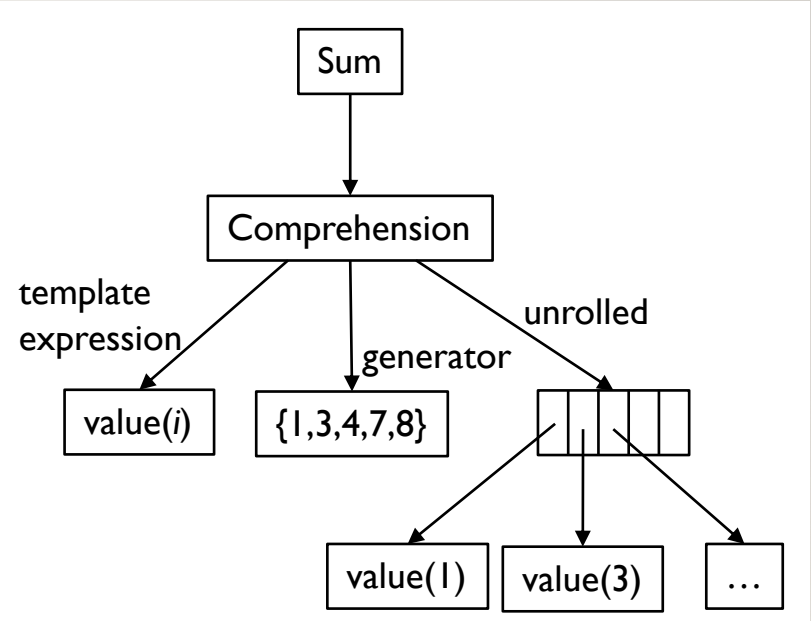
ADVANTAGES OF ATHANOR: EFFICIENT REPRESENTATION

- Revisiting the knapsack example, assume that:
  1. There are 1 million objects
  2. At most 10 can be in the knapsack in any feasible solution (weight constraint)
- 0/1 model has 1 million variables
  - Constraint and objective function both contain all variables – inefficient
- Athanor represents the value of the set, the objective, and the constraint **sparsely**
  - Scales with the actual size of the set, not the maximum size

```
given object new type enum
given weight, value : function (total) object --> int(1..)
given maxWeight: int(1..)
find knapsack : set of object
maximising sum i in knapsack . value(i)
such that (sum i in knapsack . weight(i)) <= maxWeight
```

ADVANTAGES OF ATHANOR: EFFICIENT REPRESENTATION

- **Sparse representation** of domain values
  - Suppose $knapsack = \{1,3,4,7,8\}$
  - Internal representation: variable-length array [1,3,4,7,8]
- **Dynamic unrolling** of constraints and objective
  - Abstract syntax tree combines original (template) and unrolled expression
  - e.g. knapsack objective function (right)

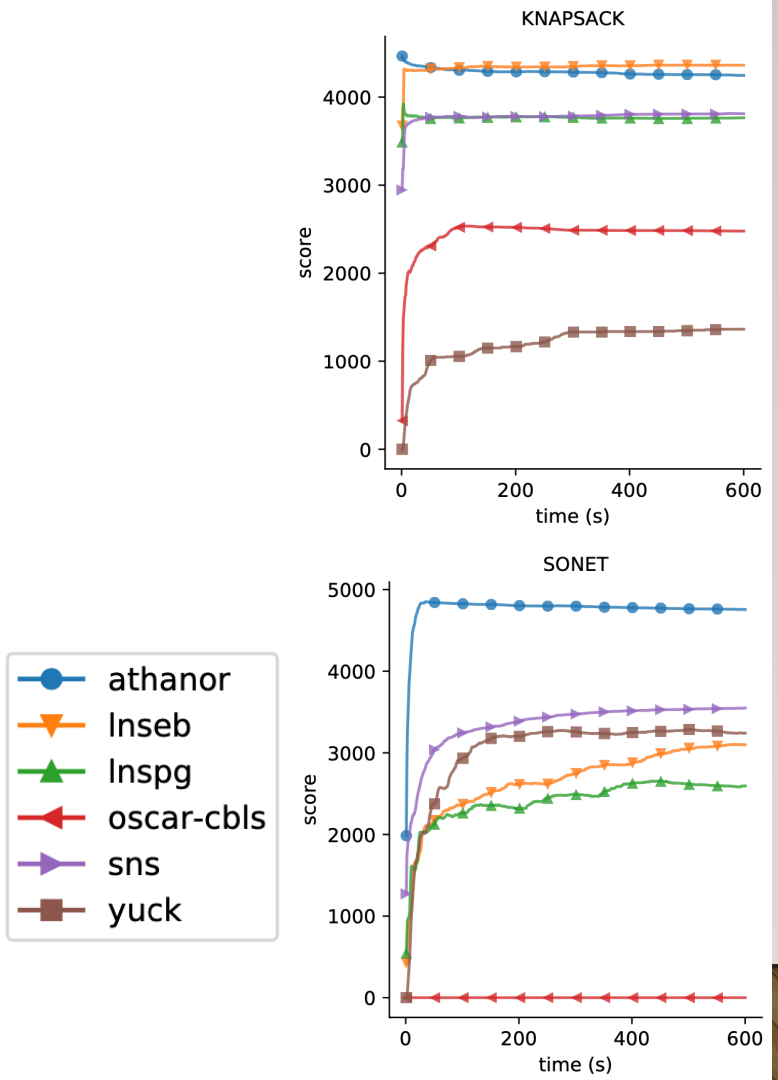


EVALUATION

- We compared Athanor to:
  - Constraint-based local search solvers (that have automatically derived neighbourhoods)
  - Automatic Large Neighbourhood Search methods (no hand-crafted neighbourhoods)
  - Systematic solvers – recent competition leaders
  - Structured Neighbourhood Search (high-level neighbourhoods derived from Essence, searched with a conventional CP solver)
- Hypotheses:
  - Neighbourhoods derived from high-level types improve search efficiency
  - Athanor scales better on large problem instances

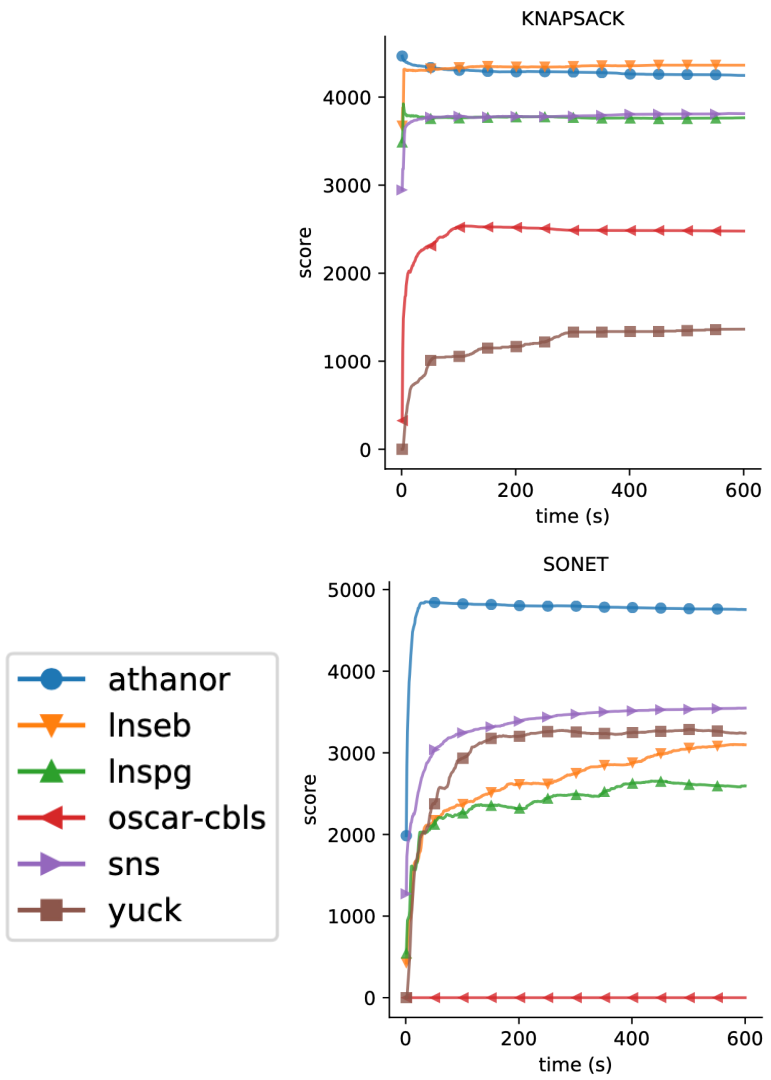
EVALUATION: ATHANOR VS LOCAL SEARCH, LNS

- Scoring system is relative
  - Solvers compete for a set of points
  - A solver's score can go up or down over time



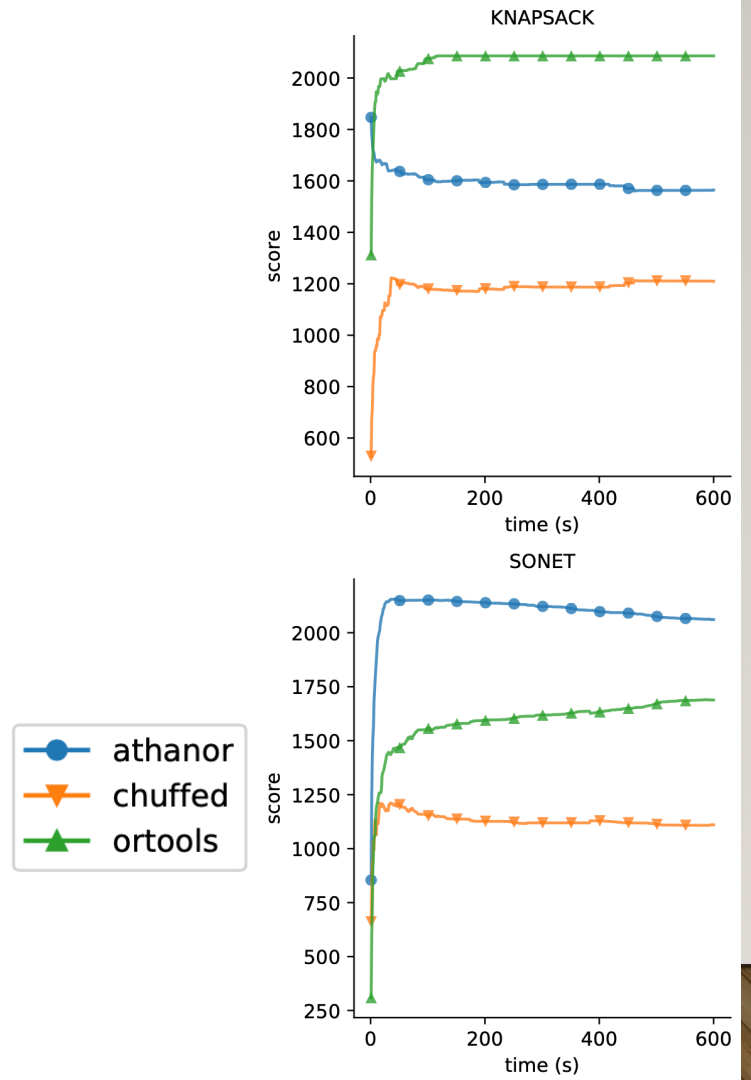
EVALUATION: ATHANOR VS LOCAL SEARCH, LNS

- Knapsack has a simple Essence type (`set`)
  - Athanor performs well but not best
- SONET has a more interesting type (`set of set`)
  - Athanor substantially outperforms all other solvers
- Supports hypothesis 1: Neighbourhoods derived from high-level types improve search efficiency
  - More benefit on the problem with the richer Essence type



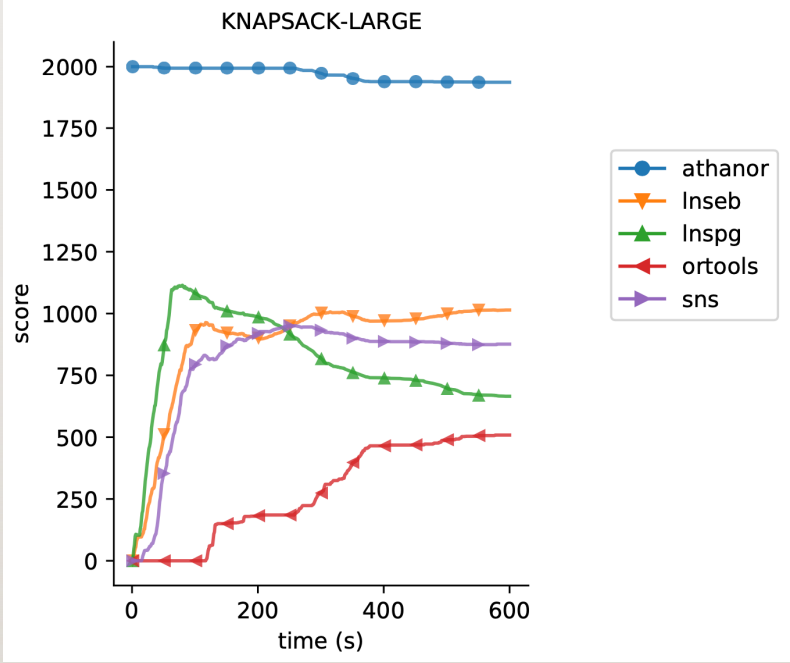
EVALUATION: ATHANOR VS SYSTEMATIC SOLVERS

- On knapsack, with relatively simple Essence type (`set`):
  - OR-Tools substantially better than Athanor
- On SONET, with a more interesting type (`set of set`):
  - Athanor performs well within 10 minute time-limit
  - OR-Tools gaining ground, may eventually overtake



EVALUATION: SCALING ON LARGE KNAPSACK

- New knapsack benchmarks with 100,000 objects
  - Compare to 5000 objects in earlier Knapsack experiments
- Athanor finds an initial solution in 0.01 s then makes steady progress
- Yuck, Chuffed, Oscar/CBLS don't find an initial solution (omitted from plot)
- As before, LNS-EB performs well
- Results support hypothesis 2: Athanor scales better on large problem instances

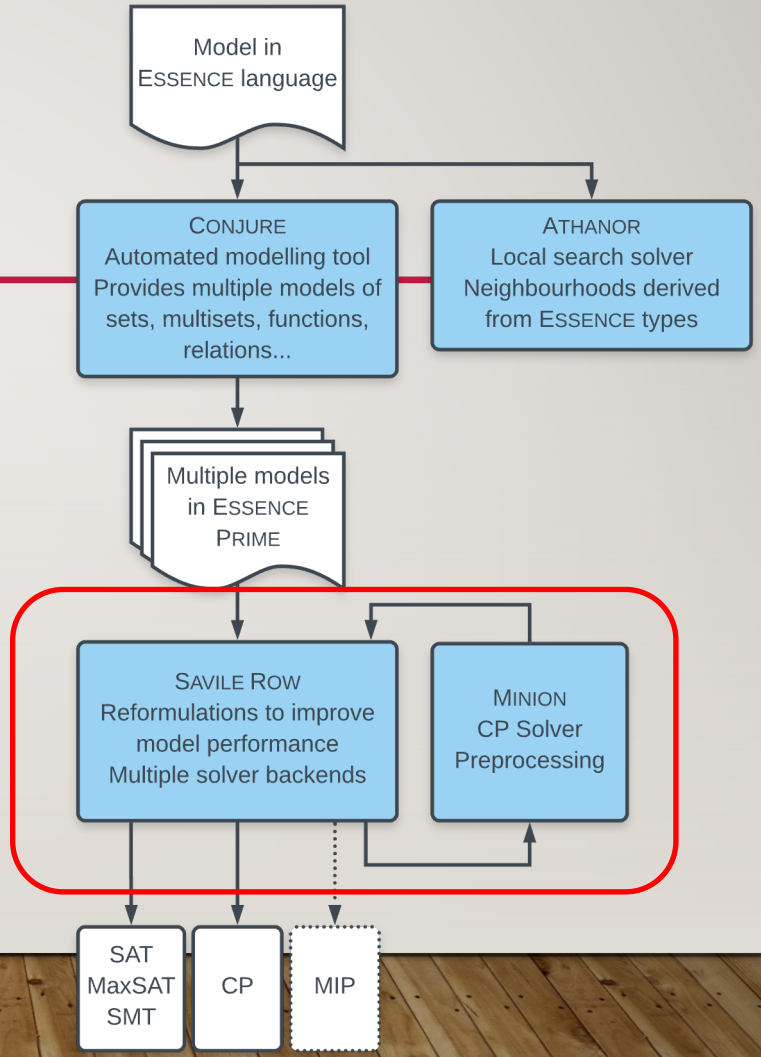


SAVILE ROW

Optimizing constraint models and encoding them for a general-purpose solver

SAVILE ROW OVERVIEW

- Starts with a *problem class* model containing:
  - Integer or Boolean decision variables
  - Matrices (no sets, multisets, sequences, functions…)
  - Quantifiers and comprehensions for concise expression of constraints
- Instantiates (makes a problem instance from) the class-level model
- Unrolls all quantifiers, comprehensions
- Optional *reformulations*
- Flattens and encodes for target solver



SAVILE ROW OVERVIEW: ESSENTIAL FUNCTIONS KNAPSACK EXAMPLE

- First, substitute in parameters (**given**)

```
given n: int(1..)
given weight, value : matrix [int(1..n)] of int(0..)
given maxWeight: int(1..)
find knapsack : matrix [int(1..n)] of bool
maximising sum i in int(1..n) . knapsack[i]*value[i]
such that (sum i in int(1..n) . knapsack[i]*weight[i]) <= maxWeight
```

```
find knapsack : matrix [int(1..3)] of bool
maximising sum i in int(1..3) . knapsack[i]*[7,6,5][i]
such that (sum i in int(1..3) . knapsack[i]*[10,2,7][i]) <= 11
```

- Now it's a problem instance, but cannot yet be given to a solver

SAVILE ROW OVERVIEW: ESSENTIAL FUNCTIONS KNAPSACK EXAMPLE

- Unroll quantifiers and comprehensions
- Break up matrices of decision variables (**find** variables)
- Partial evaluation

```
find knapsack : matrix [int(1..3)] of bool
maximising sum i in int(1..3) . knapsack[i]*[7,6,5][i]
such that (sum i in int(1..3) . knapsack[i]*[10,2,7][i]) <= 11
```

```
find kn_1, kn_2, kn_3 : bool
maximising 7*kn_1 + 6*kn_2 + 5*kn_3
such that 10*kn_1 + 2*kn_2 + 7*kn_3 <= 11
```

SAVILE ROW OVERVIEW: ESSENTIAL FUNCTIONS KNAPSACK EXAMPLE

- Flattening – assume the back-end solver does not allow a sum to be maximised
- Define a new auxiliary variable to maximise

```
find kn_1, kn_2, kn_3 : bool
maximising 7*kn_1 + 6*kn_2 + 5*kn_3
such that 10*kn_1 + 2*kn_2 + 7*kn_3 <= 11
```

```
find kn_1, kn_2, kn_3 : bool
find aux1 : int(0..18)
maximising aux1
such that
aux1 = 7*kn_1 + 6*kn_2 + 5*kn_3,
10*kn_1 + 2*kn_2 + 7*kn_3 <= 11
```

SAVILE ROW OVERVIEW: ESSENTIAL FUNCTIONS KNAPSACK EXAMPLE

- Now we have a suitable model for CP, MIP
- More complex models might also require *decomposition* of constraints such as allDifferent

```
find kn_1, kn_2, kn_3 : bool
find aux1 : int(0..18)
maximising aux1
such that
aux1 = 7*kn_1 + 6*kn_2 + 5*kn_3,
10*kn_1 + 2*kn_2 + 7*kn_3 <= 11
```

- SAT requires a final encoding step – constraints encoded as sets of SAT clauses

EFFECTIVE SAT ENCODING OF A CHALLENGING SCHEDULING PROBLEM: A CASE STUDY

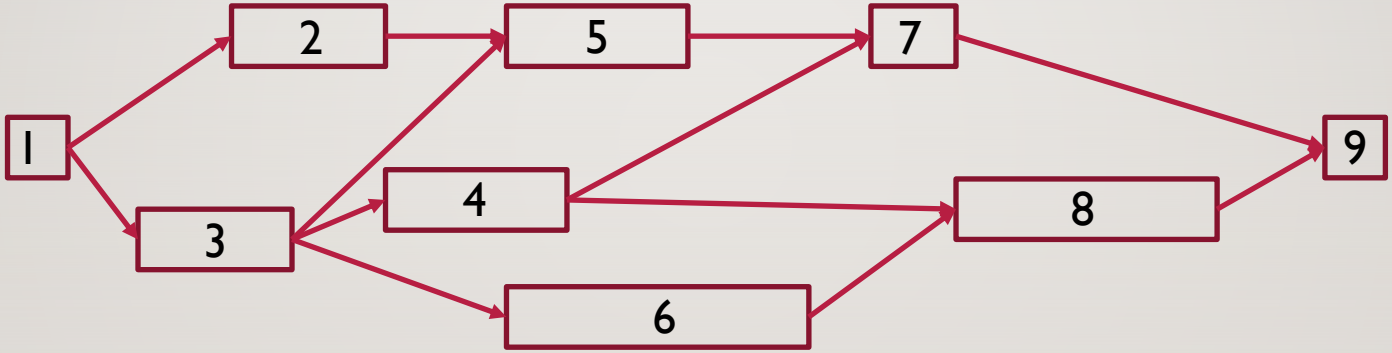
An example of reformulation in Savile Row

MRCPSP

- Schedule a set of tasks
  - Decide the start time of each task
  - Each task has a duration, and is not interruptible
  - Precedence constraints restrict the order of tasks (i.e. task *A* finishes before *B* starts)
  - Renewable resource constraints restrict which tasks can be executed concurrently
- Each task has multiple modes of execution, affecting duration and resource usage
  - Decide the mode of each task
  - Non-renewable resource constraints restrict the modes

MRCPSP – PRECEDENCES

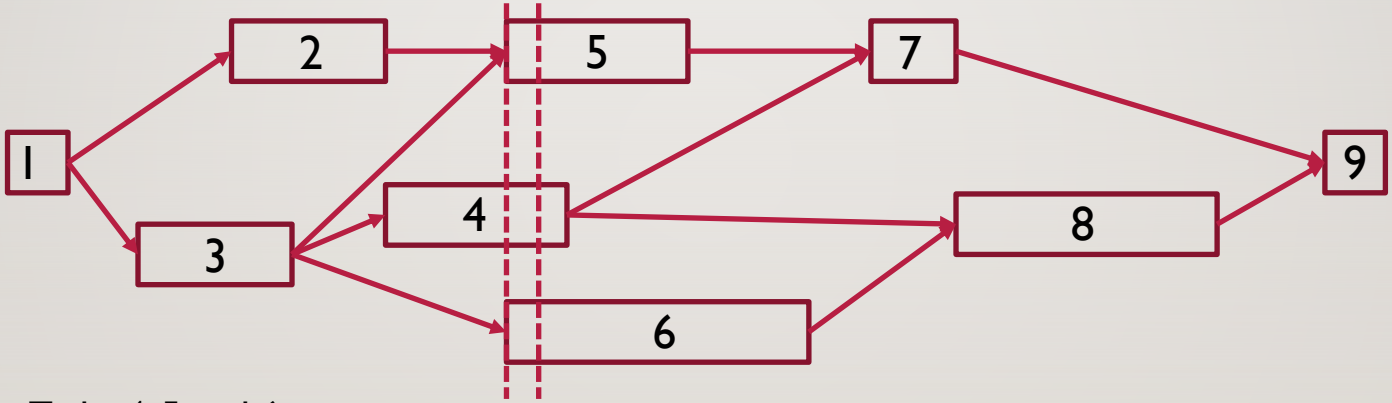
- Precedence constraints form a DAG



- Tasks 1 and 9 are dummy ‘start’ and ‘end’ tasks

MRCPSP – RENEWABLE RESOURCES

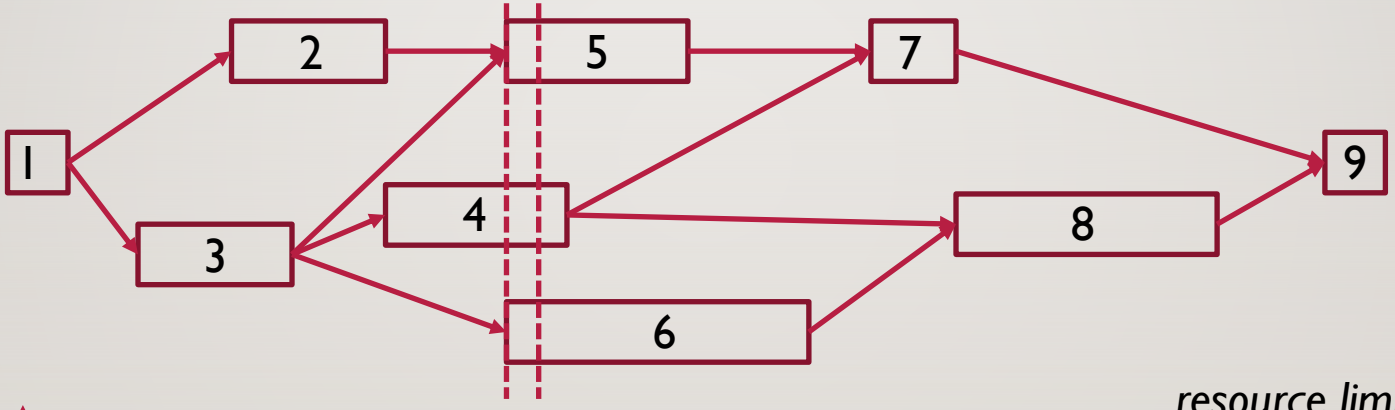
- Renewable resource constraints at each time step

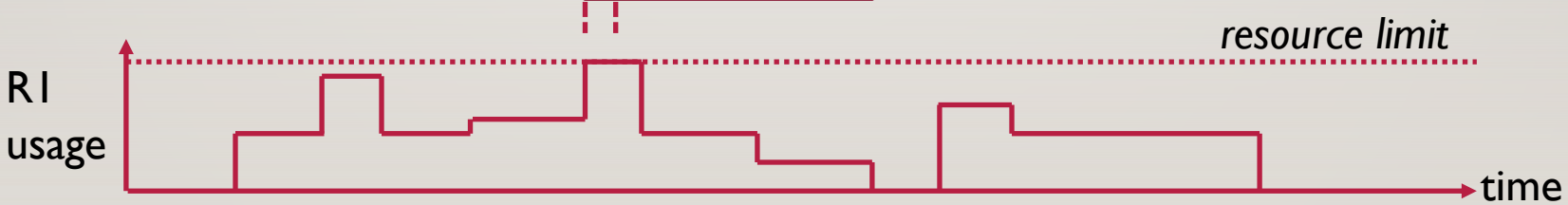


- Tasks 4, 5 and 6 are active
- **For each renewable resource**: running tasks 4, 5, and 6 in parallel does not exceed the limit on the resource (e.g. electrical supply)

MRCPSP – RENEWABLE RESOURCES

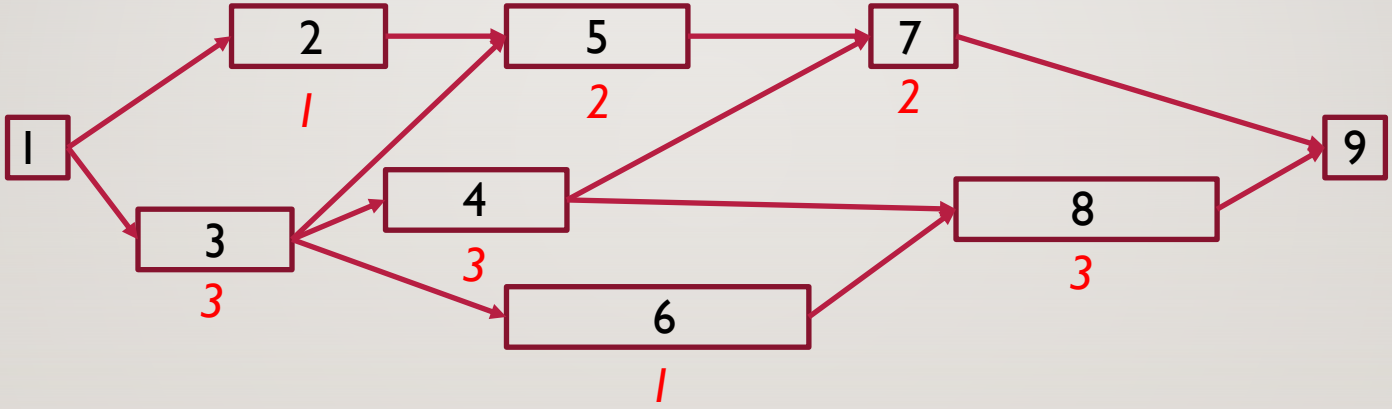
- Renewable resource usage often depicted with a histogram





MRCPSP – NON-RENEWABLE RESOURCES

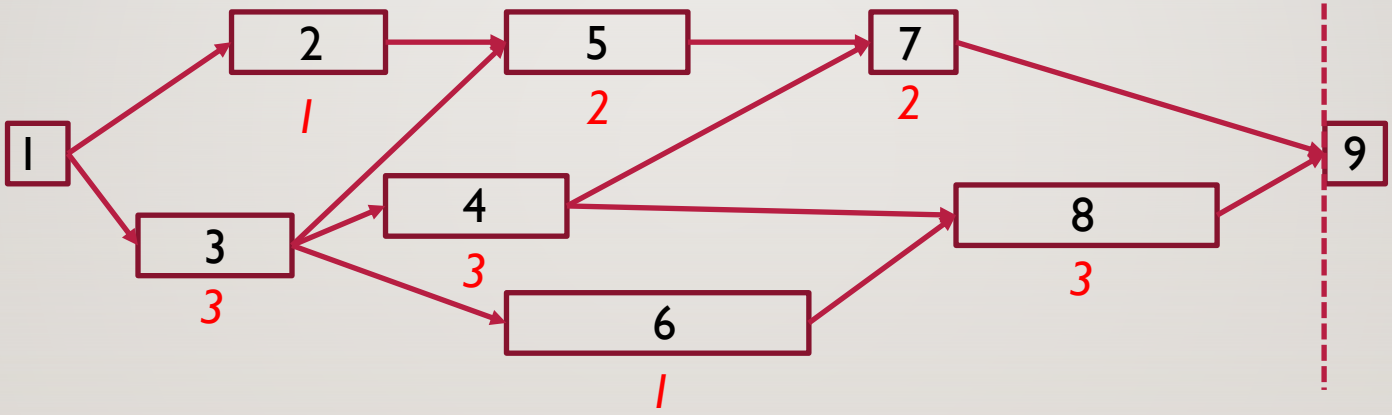
- Each task has a mode (in red)



- **For each non-renewable resource**: running all tasks in the chosen modes must not exceed the limit on the resource (e.g. money)

MRCPSP – OPTIMIZATION

- Minimize the *makespan* – start time of the final dummy task



MRCPSP – THE MODEL

- Key decision variables: whether a task *j* is active, in mode *m*, at time *t*:
  - $x_{j,m,t}$
- Variables for each task *j*: start time s_j and mode m_j
- Resource constraints dominate the encoding into SAT:

$$\forall t \in \{1 \dots horizon\}. \forall r \in \{1 \dots rRes\}.$$
$$\sum j \in \{1 \dots nTasks\}. \sum m \in \{1 \dots nModes\}.\ usage_{j,m,r} * x_{j,m,t} \leq limit_r$$

- The resource constraints are pseudo-Boolean inequalities, a well-studied class

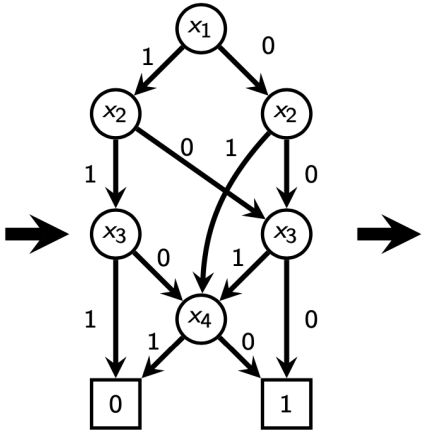
DECISION DIAGRAM ENCODING

- Encode the pseudo-Boolean resource constraints via a *decision diagram* (BDD)
- A popular encoding with useful properties

$x_1 + x_2 \leq 1$

$x_3 + x_4 \leq 1$

$2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7$



$(\overline{x_1} \vee \overline{x_2})$ $(\overline{x_3} \vee \overline{x_4})$

$(v_{2,1} \vee \overline{x_1} \vee \overline{v_1})$ $(v_{2,2} \vee \overline{v_1})$

$(v_{3,1} \vee \overline{x_2} \vee \overline{v_{2,1}})$ $(v_{3,2} \vee \overline{v_{2,1}})$

$(v_4 \vee \overline{x_2} \vee \overline{v_{2,2}})$ $(v_{3,2} \vee \overline{v_{2,2}})$

$(v_F \vee \overline{x_3} \vee \overline{v_{3,1}})$ $(v_4 \vee \overline{v_{3,1}})$

$(v_4 \vee \overline{x_3} \vee \overline{v_{3,2}})$ $(v_T \vee \overline{v_{3,2}})$

$(v_F \vee \overline{x_4} \vee \overline{v_4})$ $(v_T \vee \overline{v_4})$

(v_1) (v_T) $(\overline{v_F})$

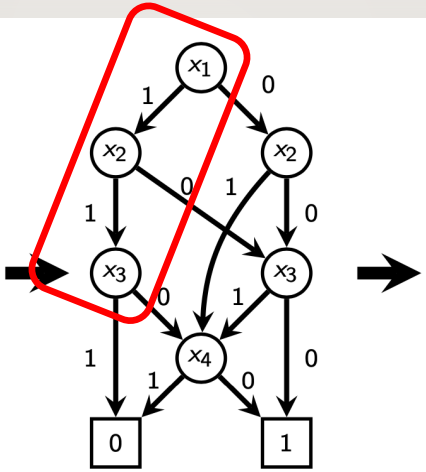
DECISION DIAGRAM ENCODING

- Key observation: BDD can have redundant paths when considering other constraints
- x_1 and x_2 cannot both be true!

$x_1 + x_2 \leq 1$

$x_3 + x_4 \leq 1$

$2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7$



$(\overline{x_1} \vee \overline{x_2})$ $(\overline{x_3} \vee \overline{x_4})$

$(v_{2,1} \vee \overline{x_1} \vee \overline{v_1})$ $(v_{2,2} \vee \overline{v_1})$

$(v_{3,1} \vee \overline{x_2} \vee \overline{v_{2,1}})$ $(v_{3,2} \vee \overline{v_{2,1}})$

$(v_4 \vee \overline{x_2} \vee \overline{v_{2,2}})$ $(v_{3,2} \vee \overline{v_{2,2}})$

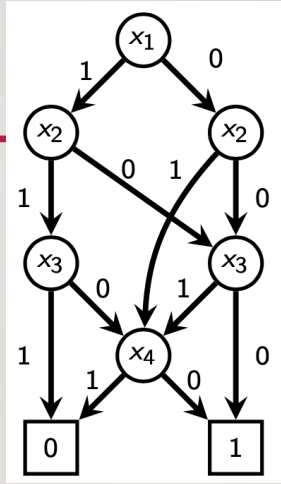
$(v_F \vee \overline{x_3} \vee \overline{v_{3,1}})$ $(v_4 \vee \overline{v_{3,1}})$

$(v_4 \vee \overline{x_3} \vee \overline{v_{3,2}})$ $(v_T \vee \overline{v_{3,2}})$

$(v_F \vee \overline{x_4} \vee \overline{v_4})$ $(v_T \vee \overline{v_4})$

(v_1) (v_T) $(\overline{v_F})$

DECISION DIAGRAM ENCODING WITH AT-MOST-ONE GROUPS

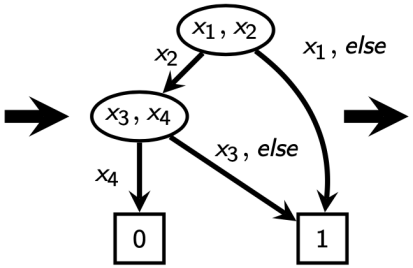


- $\{x_1, x_2\}$ and $\{x_3, x_4\}$ are at-most-one groups (AMO groups)
  - At most one literal in an AMO group may be true in any solution
- Original BDD (right) reduced to Multi-value Decision Diagram (MDD) below
  - Final SAT encoding reduced ~ 2 times

$x_1 + x_2 \leq 1$

$x_3 + x_4 \leq 1$

$[2x_1 + 3x_2] + [4x_3 + 5x_4] \leq 7$



$(\overline{x_1} \vee \overline{x_2})$ $(\overline{x_3} \vee \overline{x_4})$

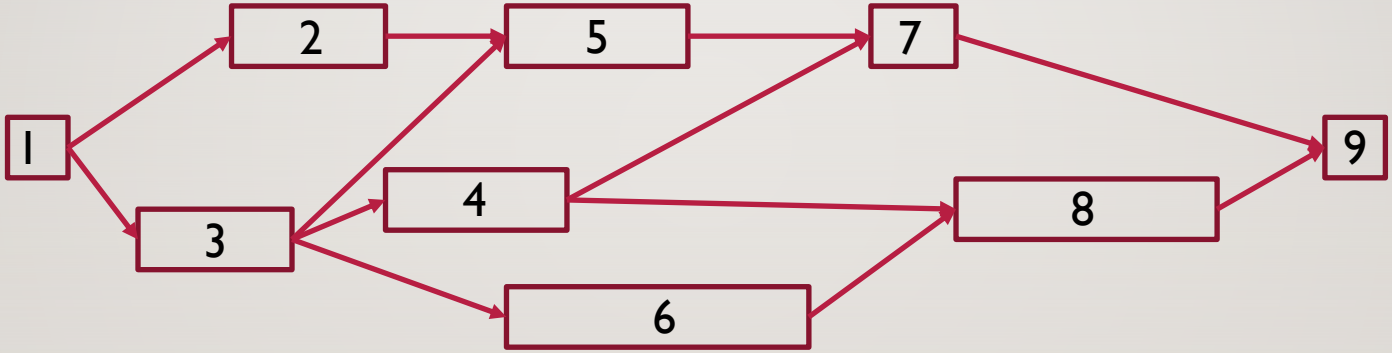
$(v_2 \vee \overline{x_2} \vee \overline{v_1})$ $(v_T \vee \overline{v_1})$

$(v_F \vee \overline{x_4} \vee \overline{v_2})$ $(v_T \vee \overline{v_2})$

(v_1) (v_T) $(\overline{v_F})$

MRCPSP – AT-MOST-ONE GROUPS

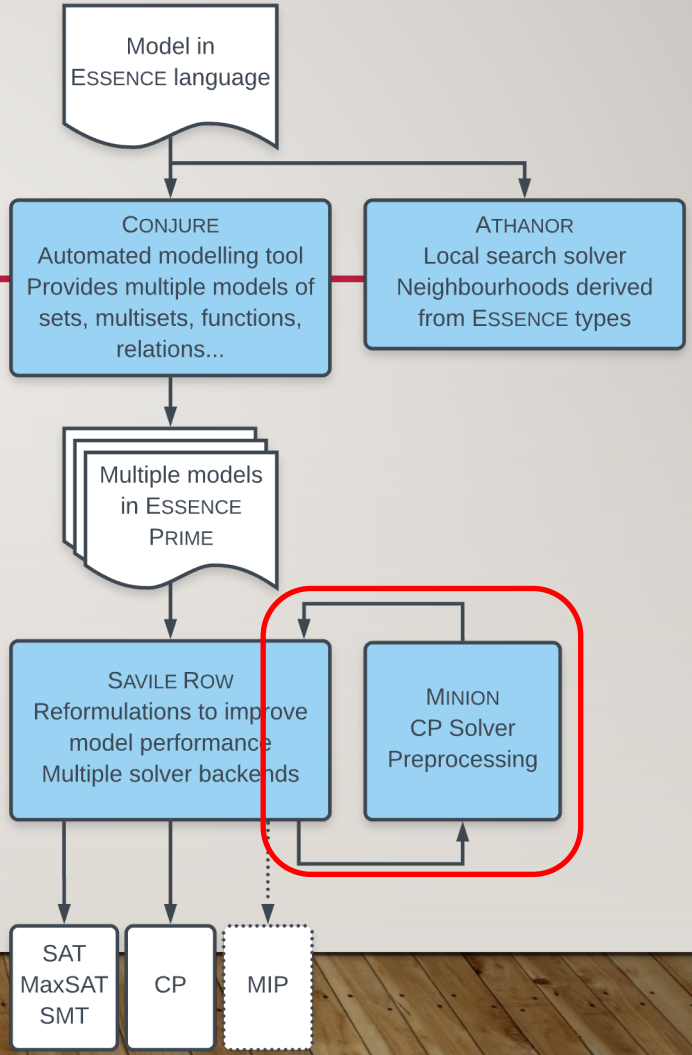
- Several sources of AMOs: modes, precedences, the resource constraints themselves



- Task 6 *runs in one mode*: $x_{6,1,t}$ and $x_{6,2,t}$ are mutually exclusive (for every timestep t)
- Tasks 6 and 8 *cannot overlap*: $x_{6,m1,t}$ and $x_{8,m2,t}$ are mutually exclusive (for every $t, m1, m2$)

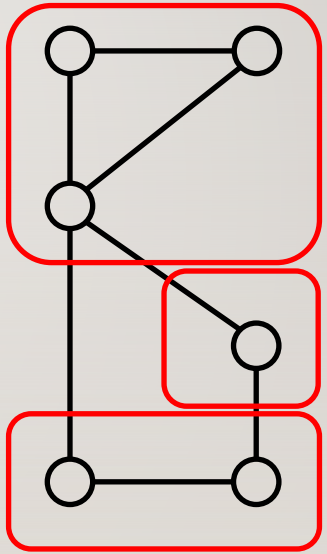
STEP 1: MUTEX DETECTION

- Automatic mutex detection is achieved by a *solver feedback loop*
  - Model is sent to CP solver Minion
  - Mutexes found by assigning Boolean variables and running Minion's propagation loop
  - Entire problem instance is involved: Boolean and integer variables, global constraints, …
- Small additional cost when combined with *domain filtering* (on by default)



STEP 2: CLIQUE COVER

- We need to compute AMO groups for a constraint from mutexes
- Define a graph
  - Vertices are literals in the constraint
  - Edges are mutexes discovered by propagation
- Clique cover partitions the literals into AMO groups
  - Simple greedy algorithm is sufficient
  - Does not always take advantage of all mutexes

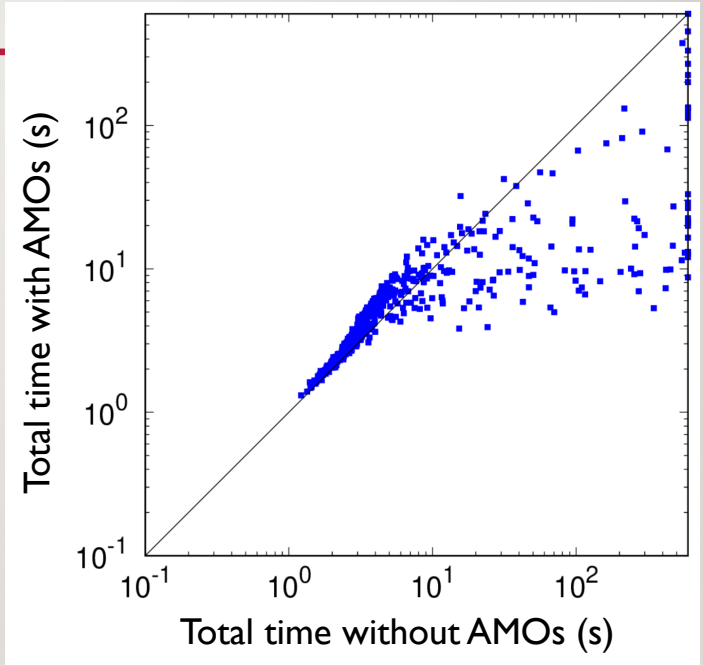


EVALUATION: MRCPSP

- We evaluated the reformulation using a standard benchmark set (j30, 552 satisfiable instances)
- 10 minute time-limit, Open-WBO MaxSAT solver
- Straightforward comparison of total time (Savile Row time + solver time) with and without automatic AMO detection

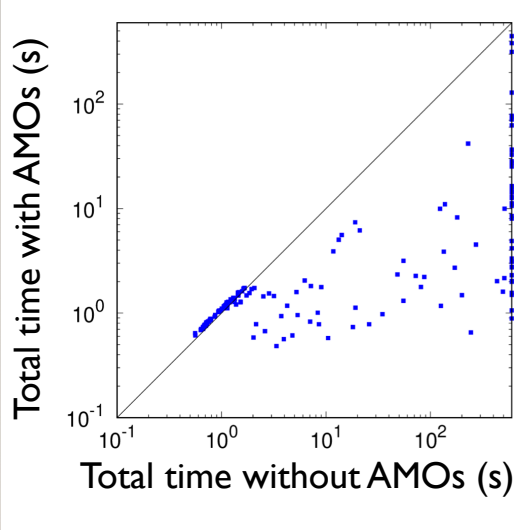
EVALUATION: MRCPSP

- Some easier instances show overhead
  - Up to ~20 s
- Much better scaling on difficult instances
  - 10 times or more speed-up for many instances
  - More instances solved within 10 minutes
- BDD encoding *without* AMOs already better than other exact methods [1]

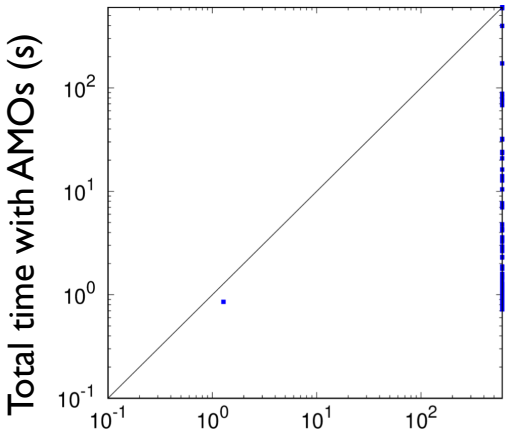


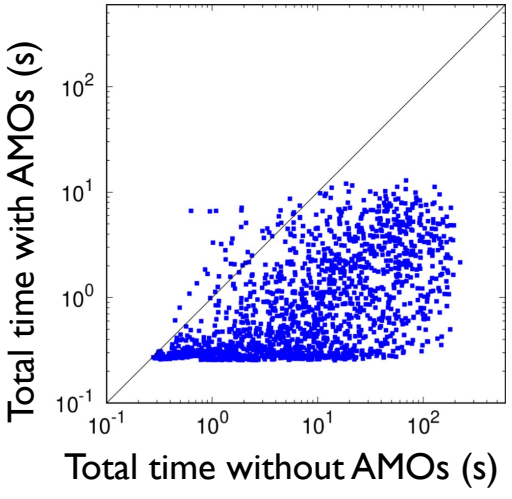
[1] Scheduling Through Logic-Based Tools, Jordi Coll Caballero, PhD thesis, 2019

EVALUATION: OTHER PROBLEMS



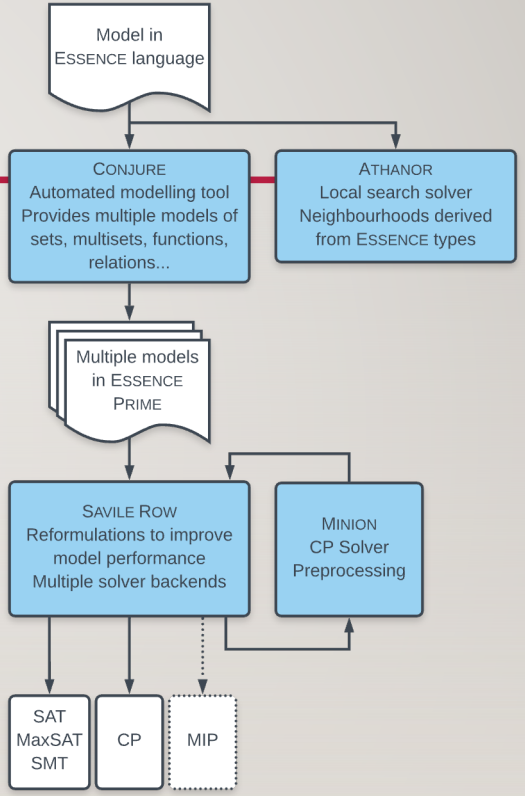
- Pseudo-Boolean inequalities are a staple in CP modelling
- AMO groups are not uncommon, several other problem classes benefit:
  - Combinatorial Auctions (left)
  - Nurse Scheduling Problem (upper right)
  - Multiple-choice Multidimensional Knapsack Problem (lower right)





CONCLUSIONS

- An overview of a constraint modelling pipeline
- An argument for the benefits of high-level modelling
  - Multiple diverse models produced by Conjure
  - Local search neighbourhoods generated by Athanor
- At the lower level, the potential for reformulation to improve model performance
  - With an example of one reformulation, applied to MRCPSP



THANK YOU!

- Özgür Akgün, Alan M. Frisch, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale, *Conjure: Automatic Generation of Constraint Models from Problem Specifications*, Artificial Intelligence, 2022
- Saad Attieh, Nguyen Dang, Christopher Jefferson, Ian Miguel, Peter Nightingale, *Athanor: High-Level Local Search Over Abstract Constraint Specifications in Essence*, IJCAI 2019
- Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, Patrick Spracklen, *Automatically Improving Constraint Models in Savile Row*, Artificial Intelligence, 2017.
- Miquel Bofill, Jordi Coll, Peter Nightingale, Josep Suy, Felix Ulrich-Oltean, and Mateu Villaret, *SAT Encodings for Pseudo-Boolean Constraints Together With At-Most-One Constraints*, Artificial Intelligence, 2022.
- Carlos Ansótegui, Miquel Bofill, Jordi Coll, Nguyen Dang, Juan Luis Esteban, Ian Miguel, Peter Nightingale, András Z. Salamon, Josep Suy, Mateu Villaret, *Automatic Detection of At-Most-One and Exactly-One Relations for Improved SAT Encodings of Pseudo-Boolean Constraints*, CP 2019.

QUESTIONS? COMMENTS?

Quote from *The Arkwrights: Spinners of Fortune*

Sough. The great-uncle of Florence Nightingale, this eccentric sporting squire, known throughout the county as 'Mad Peter Nightingale', had gained notoriety as a dare-devil horseman, a rider in midnight steeplechases and a layer of wagers, given to hard drinking and low company.[60] This was not the only friction with