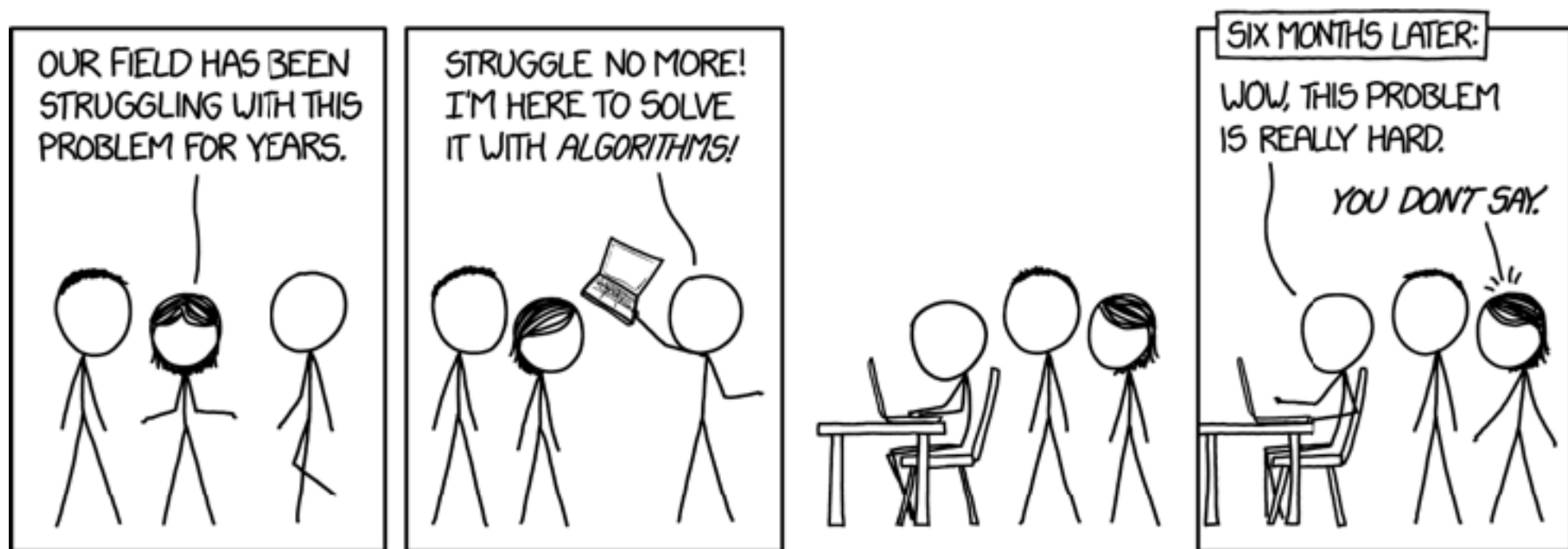


# DECOMPOSITION & HYBRIDISATION FOR COMBINATORIAL OPTIMISATION



XKCD  
#1831

# COMBINATORIAL OPTIMISATION

- ◆ maximise a real-valued objective function over a **discrete** set of solutions defined descriptively

$$\max_{s \in S} f(s), \quad S \subseteq 2^A, f: 2^A \rightarrow \mathbb{R}$$

- ◆ satisfaction problem if objective is constant
- ◆ **solving** = find an algorithm **better** than brute-force enumeration
  - **efficient** = fast, scalable, exact/near-optimal
  - **practical** = evolvable, **flexible** and robust to changes

# MANY PARADIGMS, MANY ALGORITHMS

**Integer Programming**

**(Finite-Domain)  
Constraint Programming**

**Global Optimisation**

Formal Verification

**Graph Theory**

**Metaheuristics**

**Dynamic Programming**

Formal Languages

**Logic Programming**

**NONE TO RULE THEM ALL**

**SAT**

**SMT**

Machine Learning

Control

Functional Analysis

Calculus

Discrete Algebra

Logic

Signal Processing

# PROBLEM/ALGORITHM ADEQUACY

which algorithm for my problem ?

- ◆ some formalisms are more suitable to some problems
- ◆ some algorithms are more efficient for some problems
- ◆ some algorithms are less/more efficient but more/less flexible
- ◆ many problems are **composite**, different subsystems, different articulations, e.g.:
  - several subsystems sharing decisions or constraints
  - one basic system with complicating side constraints

3 examples...



# REAL-TIME DATACENTER RESOURCE MANAGEMENT



place & move Virtual Machines, satisfy the resource capacities and user requirements

challenges:

- an integrated **packing/scheduling** problem: where to place ? when to move ?
- a non-conventional cumulative scheduling problem induced by live-migrations
- **various user constraints**: isolate/security, spread/fault tolerance, gather/performance, clear/maintenance,...
- flexibility: the problem – resource usage & user constraints – is **dynamic and evolvable**



# Example 2 ROSTERING

The image shows a hand-drawn rostering chart on a chalkboard. The chart is organized into columns for the days of the week: MONDAY, TUESDAY, WEDNESDAY, THURSDAY, and FRIDAY. Each day has a grid of time slots (rows) and employee assignments (columns). The time slots are labeled with times such as 8:20, 8:55, 9:30, 9:50, 10:25, 11:00, 11:30, and 12:05. The employee assignments are labeled with numbers 1 through 10. The activities are written in various colors (red, green, blue, yellow) in the cells. To the right of the grid is a list of 10 employees with their names and IDs:

IRS NO	Employee Name
1.	MR KIRII
2.	MR NJAG
3.	MRS MACA
4.	MRS IBU
5.	MR NJL
6.	MA NJ
7.	MR MU
8.	TR D
9.	MISS
10.	MRS

assign activities to every employee every time, cover the load, satisfy the working rules

challenges:

- two **orthogonal** problems: **schedule activities** for each employee / **assign employees** to each activity
- various **working rules**: forbidden activities/sequences, min/max delays/lengths, fixed/sliding period, hard/soft
- the problem – data, objective & constraints – varies with the context



# PUMP DESIGN IN WATER DISTRIBUTION NETWORKS



choose pumps, satisfy the demand, minimise investment+operation costs

challenges:

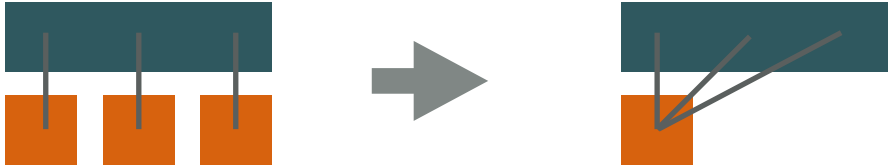
- **investment**/long-term: investment, maintenance, ageing, uncertain demand/price/availability
- **operation**/short-term: pump scheduling, non-convex hydraulic constraints
- **various networks**: looped/branched, fixed/variable-speed pumps, different kinds of valves

# HYBRID DECOMPOSITION PRINCIPLE

- ◆ in contrast to monolithic algorithms: one model, one search
- ◆ split a problem into coherent substructures by breaking their links
- ◆ apply algorithms from different paradigms to each substructure
- ◆ **recover the broken links**: iterate on the components and let them communicate together
- ◆ get an optimal/feasible/approximated solution depending of the recovery

# EXPECTATIONS

one major drawback: naive implementations have slow rates of convergence

- ◆ **flexibility**: exhibit/isolate the substructures
- ◆ **modularity**: adaptable/reusable algorithmic/software components
- ◆ **efficiency**: dedicated algorithm for each substructure
- ◆ **applicability**: generic de-composition frameworks & refinement techniques
- ◆ **scalability** (dynamic generation): solve subproblems repeatedly  $2^{2n} \gg K(2^n + 2^n) \gg \sum_k (2^n + 2^k)$
- ◆ **scalability**: symmetry breaking 
- ◆ **quality**: if not optimal, give good relaxations/approximations (better than when breaking the links)

# PRACTICAL QUESTIONS

- ◆ how to decompose ? which algorithm for which block ?
- ◆ how deep the decomposition ? breaking strong links makes the subproblems easier but the convergence slower
- ◆ how to combine different algorithms ? how to make the communication efficient ?
- ◆ proof of convergence? speed of convergence ? what if incomplete search ?

# THIS TALK IS ABOUT

- ◆ **hybrid models**, not hybrid techniques (portfolio, CBLS, CP+nogoods, ML & CP/MP,...)
- ◆ **semantic** decomposition, not structural decomposition (tree search, graph splitting,...)
- ◆ algorithm **composition**... because decomposition is the easy part
- ◆ a bit of theory, more about **applications**... on problems I have worked on (not exhaustive)
- ◆ two contexts for hybrid decomposition (not exhaustive):
  - I. **global constraints in CP** & hybridisations with automata & LP
  - II. **decomposition methods in MP** & hybridisations with CP & global optimisation

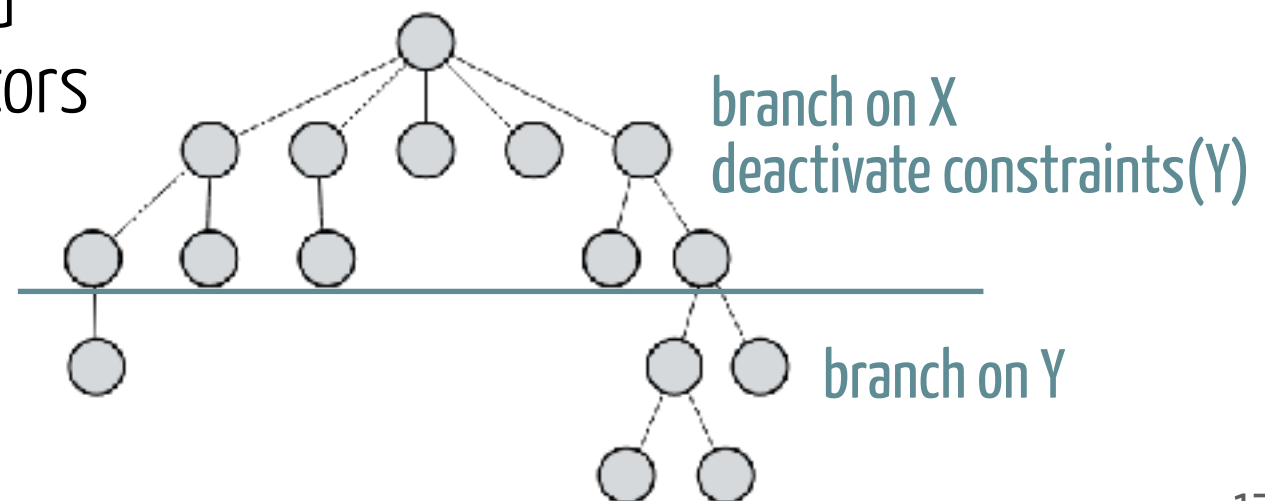


# I. CONSTRAINT PROGRAMMING WITH GLOBAL CONSTRAINTS



# CONSTRAINT PROGRAMMING: THE ESSENCE OF DECOMPOSITION

- ◆ separate a model in constraints, "solve the constraints" (filter non-solutions) independently
- ◆ the constraints communicate through the variable domains
- ◆ the consistency algorithm is a **generic cooperation scheme** gluing the constraint models
- ◆ convergent scheme if strong n-consistency
- ◆ otherwise, backtracking is required and applied globally, but decomposition can be fully enforced by a multistage branching strategy and by delaying the subproblem constraint propagators



# REAL-TIME DATACENTER RESOURCE MANAGEMENT



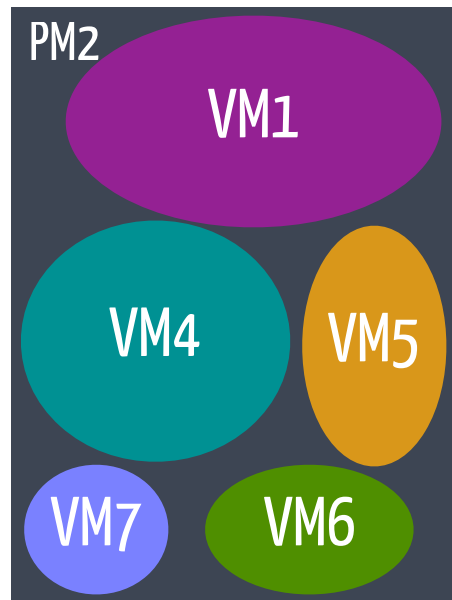
challenges:

- an integrated packing/scheduling problem: where to place/how to collocate? how and when to move?
- a non-conventional producer-consumer cumulative scheduling problem induced by live-migrations
- **various user constraints**: isolate/security, spread/fault tolerance, gather/performance, clear/maintenance,...
- flexibility: the model – resource usage & user constraints – is **dynamic and evolutive**

# Example 1: datacenter resource management

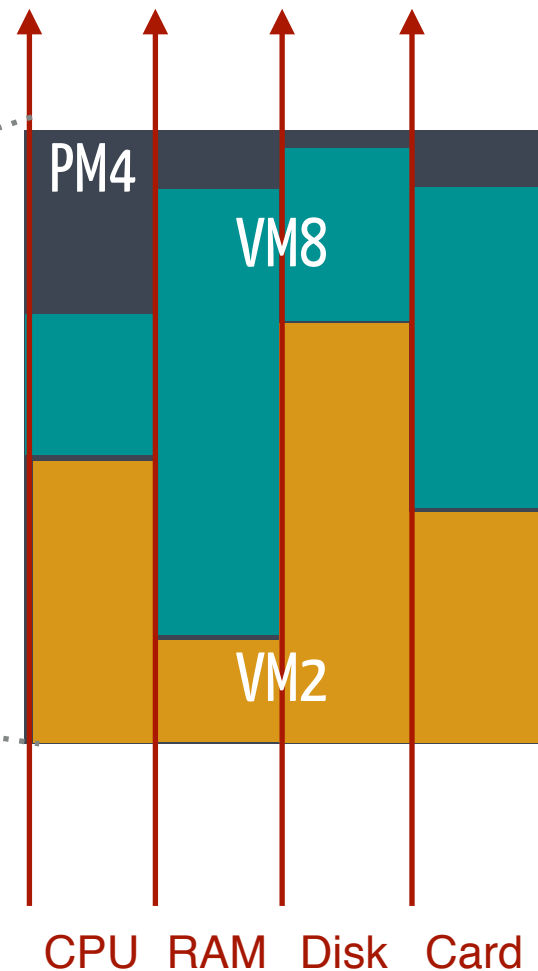
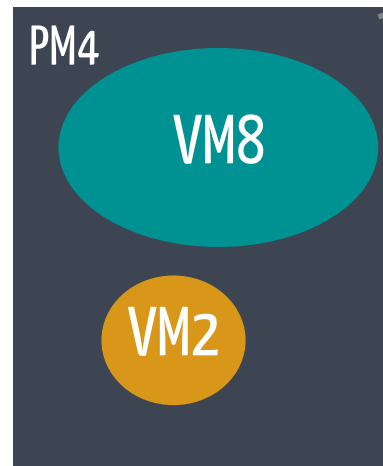
$assign : VMs \rightarrow PMs$

$$\sum_{VM \in assign^{-1}(PM)} DEMAND(VM, RES) \leq CAPACITY(PM, RES), \forall (PM, RES)$$



possible objectives:

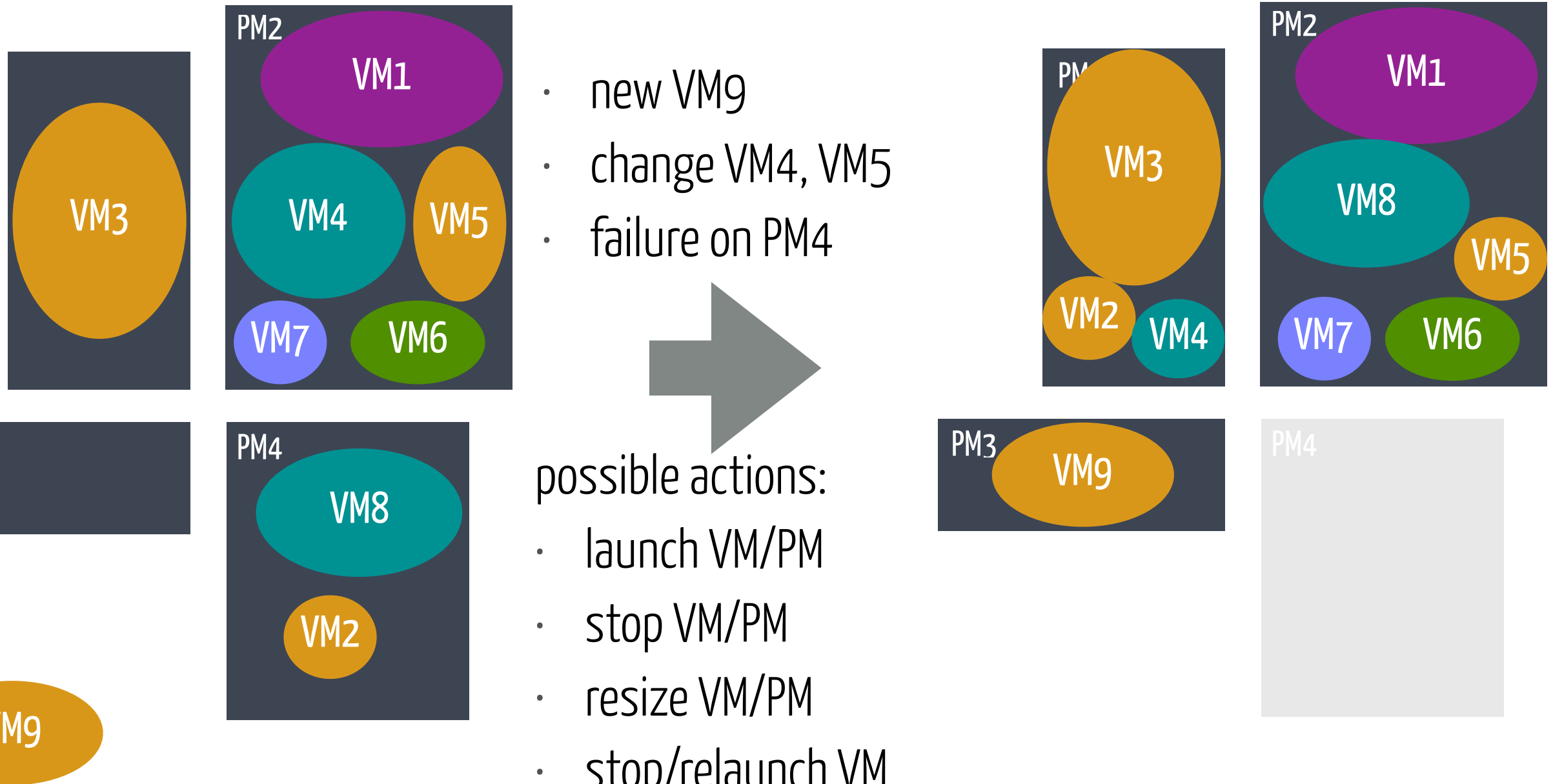
- minimise the number of PMs (energy saving)
- maximise the number of PMs (performance)
- minimise the average load over all PMs



VECTOR PACKING

but dynamic...

# Example 1: datacenter resource management



- new VM9
- change VM4, VM5
- failure on PM4

possible actions:

- launch VM/PM
- stop VM/PM
- resize VM/PM
- stop/relaunch VM
- live/direct migrate VM

VECTOR **REPACKING**

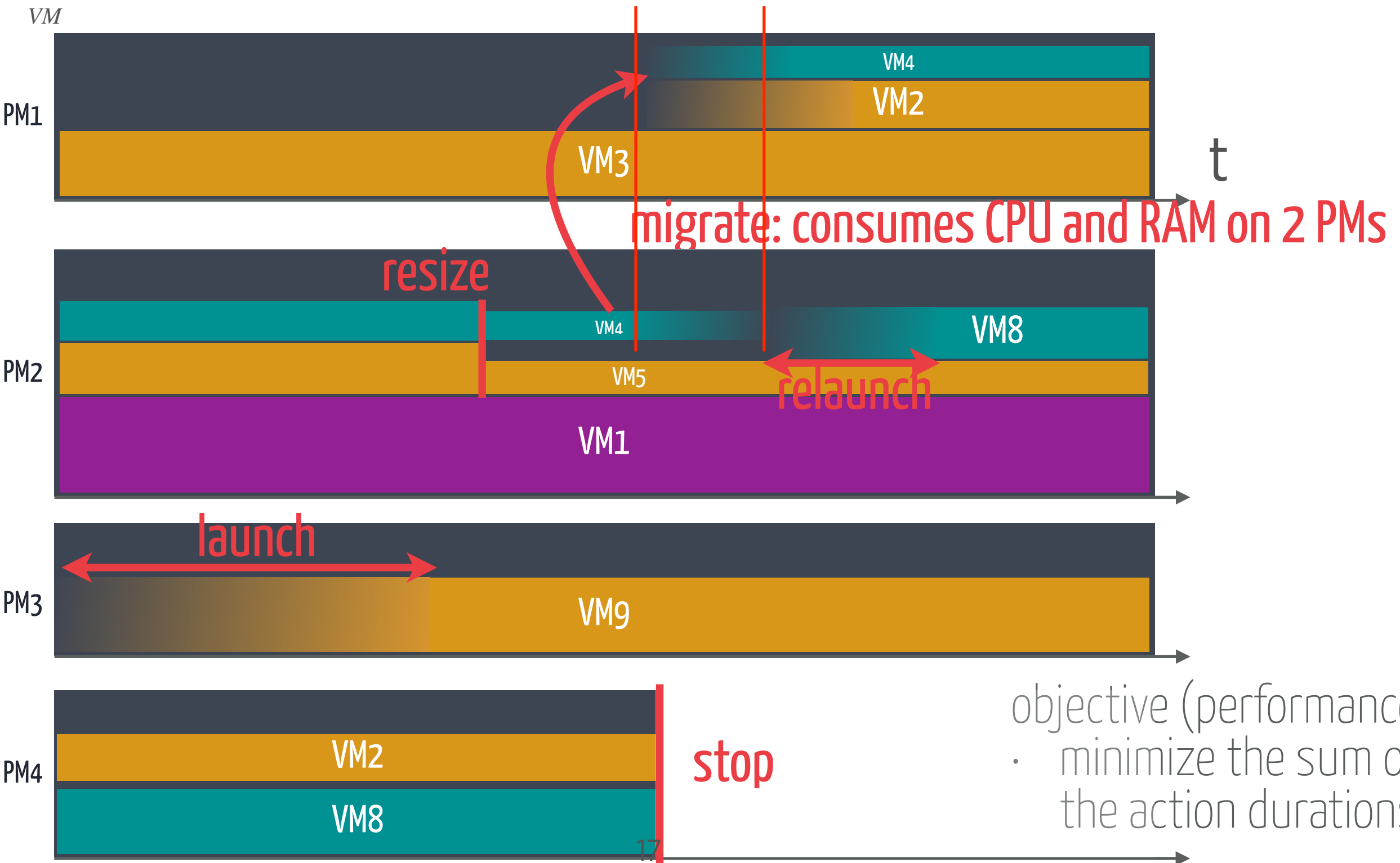
but actions are not all immediate...

# Example 1: datacenter resource management

## PRODUCER-CONSUMER SCHEDULING

start : VMs  $\rightarrow$  ACTIONs  $\times$  PMs  $\times$  TIMEs

$$\sum_{VM} DEMAND(start(VMs), PM, RES, T) \leq CAPACITY(PM, RES, T), \forall (PM, RES, T)$$





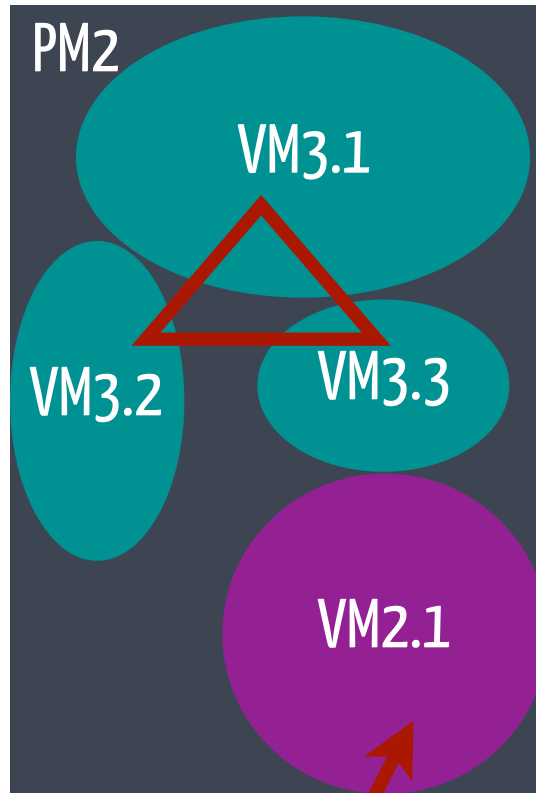
# Example 1: datacenter resource management

## USER SIDE CONSTRAINTS

spread,  
among,  
ban,  
fence,  
gather,  
root,  
lonely,  
quarantine  
capacity,  
splitAmong  
preserve,  
overSubsc  
offline,  
noldles,  
mostlySpr

$$\text{assign}(VM1) \cap \text{assign}(VMs \setminus VM1) = \emptyset$$

**security**  
→  
(isolate)

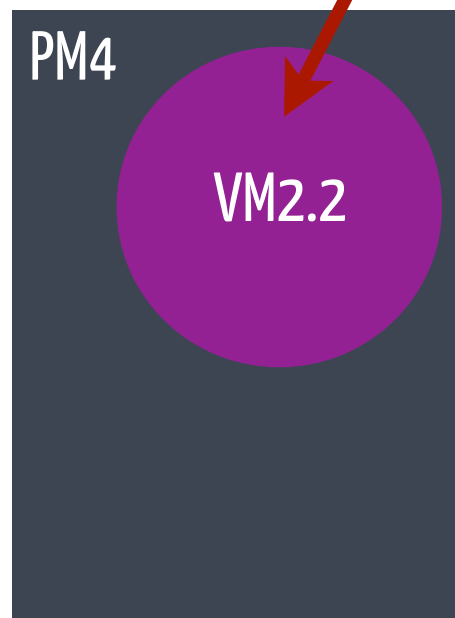


**performance**  
(gather)  
 $\text{card}(\text{assign}(VM3)) = 1$



**maintenance** ↑  
(offline)

$$\text{assign}(VMs) \cap PM3 = \emptyset$$



**fault tolerance**  
(spread)  
 $\text{alldifferent}(\text{assign}(VM2))$

# Example 1: datacenter resource management

## OPPORTUNITY OF DECOMPOSITION

- ◆ we need a fast (<10min), scalable solution, not necessary optimal
- ◆ we need to freely & automatically compose side constraints in a flexible non-intrusive way i.e. without altering the global structure
- ◆ identified components: (packing +  $\sum$ side constraints) <-----> scheduling

# Example 1: datacenter resource management

## A CP-BASED APPROACH

---

- ◆ CP fits well with both **non-conventional packing** and **scheduling** components
- ◆ CP/consistency is appropriate to integrate any kind/any number of **side constraints** without impacting the two main component models or the search algorithm (no need to expose extra variables)
- ◆ side constraints: concise models relying on an **extensive evolutive catalog** of predicates (alldifferent, cardinality, element, nvalue,...) with efficient implementations in CP solvers
- ◆ automatic model composition: DSL + automatic translation + automatic parametrisation: alternative algorithms
- ◆ reduce the search space heuristically by fixing VMs/PMs whose current assignment does not violate any new constraints: define checkers



# Example 1: datacenter resource management

## A CP-BASED APPROACH

---

- ◆ CP fits well with both **non-conventional packing** and **scheduling** components
- ◆ CP/consistency is appropriate to integrate any kind/any number of **side constraints** without impacting the two main component models or the search algorithm (no need to expose extra variables)
- ◆ **Global Constraints** (diagonal label): concise models relying on an **extensive evolutive catalog** of predicates (different, cardinality, element, nvalue,...) with efficient implementations in CP solvers
- ◆ **automatic model composition: DSL + automatic translation + automatic parametrisation: alternative algorithms**
- ◆ reduce the search space heuristically by fixing VMs/PMs whose current assignment does not violate any new constraints: **define checkers**

# GLOBAL CONSTRAINTS: THE ESSENCE OF HYBRIDISATION & MODULARITY

- ◆ **improve consistency** by composition [Regin94]  $alldifferent(x_1, \dots, x_n) \gg \bigwedge_{i \neq j} x_i \neq x_j$
- ◆ actually a perfect tool to implement hybrid decomposition [since ALICE78, CHIP88]:
  - **expressivity**: 1 substructure = 1 predicate
  - **encapsulation**: hide internal computations
  - **hybridisation**: implement any algorithm from any paradigm
  - **modularity**: reusable, adaptable, parametrable (portfolio algorithms)
  - **additional services**: propagators, checker, solver, violation counters, nogood computation, handlers on internal information, probability,...

# Example 2 (NURSE) ROSTERING

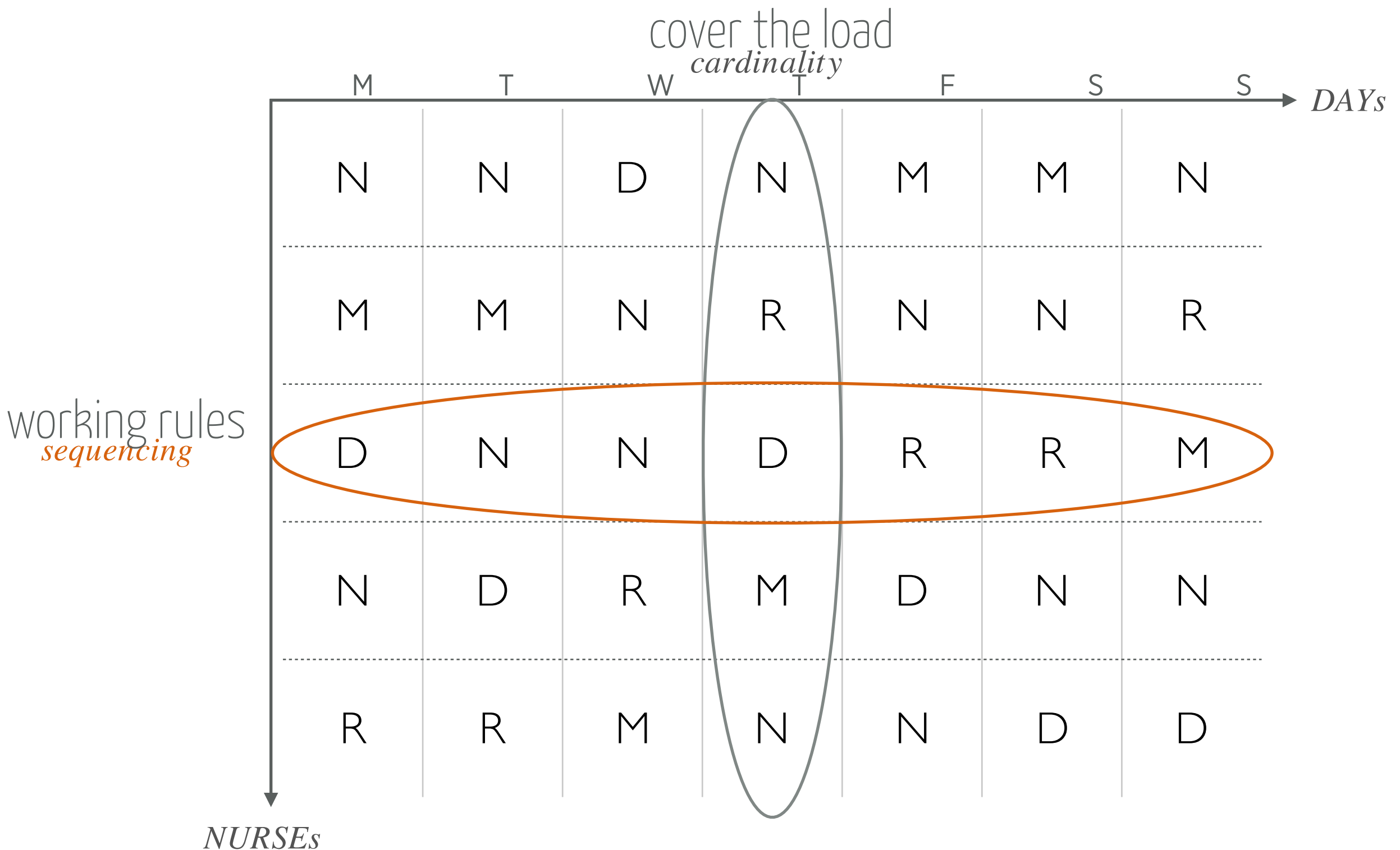
assign activities to every employee every time, cover the load, satisfy the working rules

challenges:

- two orthogonal problems: **schedule activities for each employee** / assign employees to each activity
- **numerous and heterogeneous working rules**, hard and soft
- the model (data, objective and constraints) **changes from instances to instances**

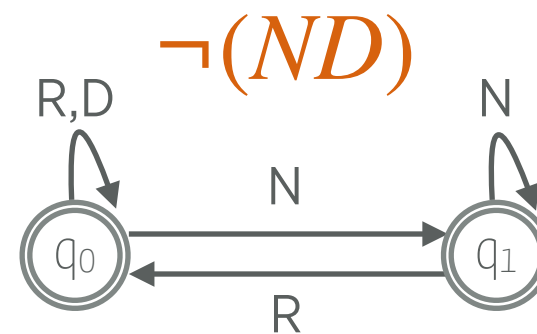
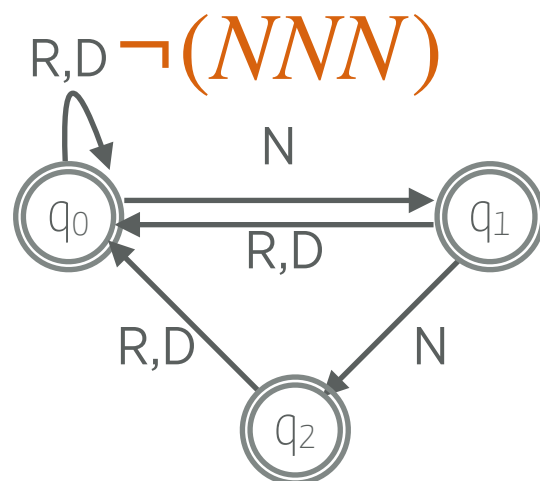
# Example 2: nurse rostering

$assign : NURSEs \times DAYs \rightarrow ACTs = \{Morning, Day, Night, Rest\}$



# CSP & FORMAL LANGUAGES

- ◆ a nurse schedule is a **word** on the **alphabet** of activities  $RRDNRDD \in \{D, N, R\}^7$
- ◆ a working rule defines a **language** on this alphabet: the set of words which satisfy the rule  
"max two consecutive Nights" "Rest after Nights"
- ◆ a language is **regular** iff it can be represented as a finite automaton/regular expression



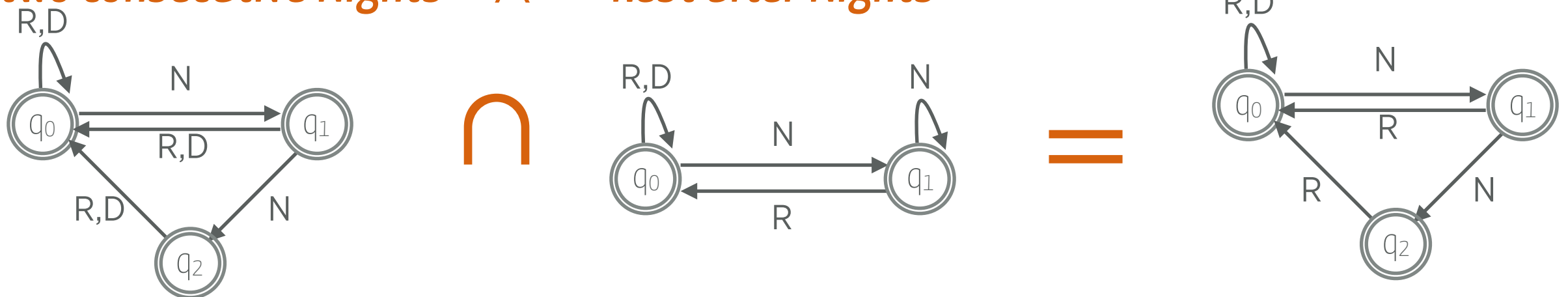


# AUTOMATIC MODEL COMPOSITION WITH AUTOMATA

- finite automata are concise and **composable** models: fast intersection/complement/minimisation... algorithms, keep the model concise

- the set of feasible schedules is a language resulting from **intersection**

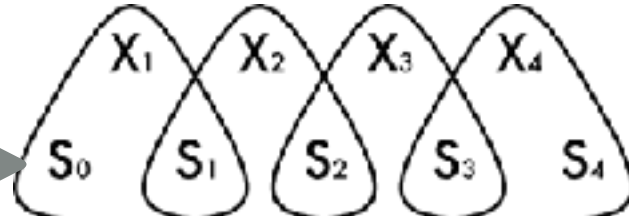
"max two consecutive Nights"  $\wedge$  "Rest after Nights"



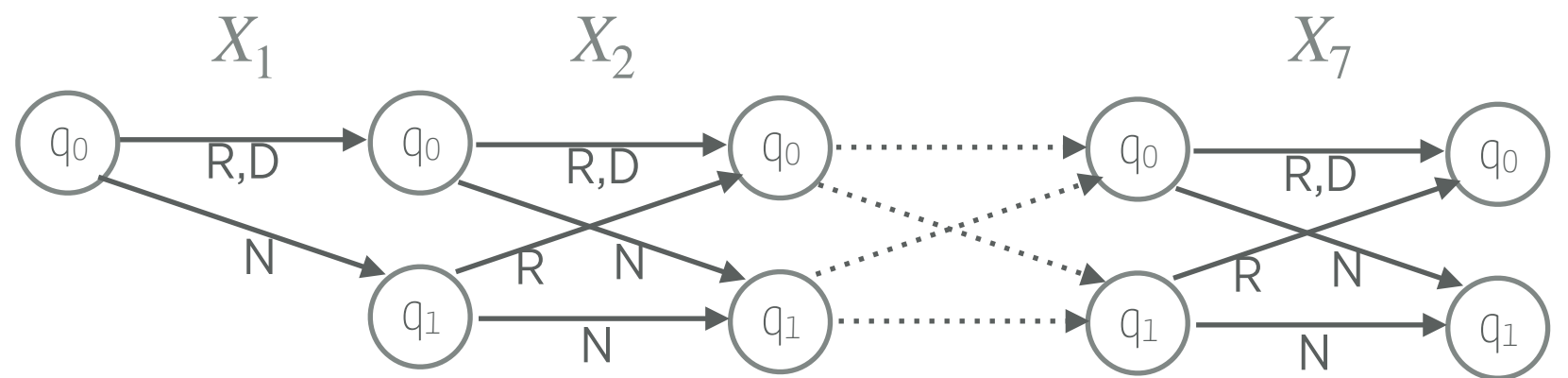
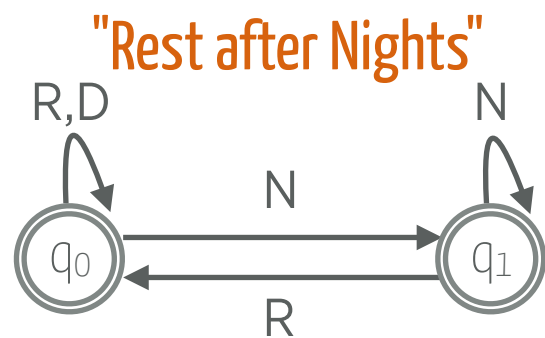
- automata intersection = **constraint conjunction** = pre-filtering

# FILTERING WITH AUTOMATA

- key ideas: (1) variable values = transition labels, assignment  $X_1X_2\dots X_n = \text{word/path}$   
 (2) word length is fixed

- decomposition:**  $\text{automaton}(X_1, \dots, X_n, \pi)$  [Beldiceanu04]  Berge-acyclic:  
AC = GAC

- dynamic programming:**  $\text{regular}(X_1, \dots, X_n, \pi)$  [Pesant04]

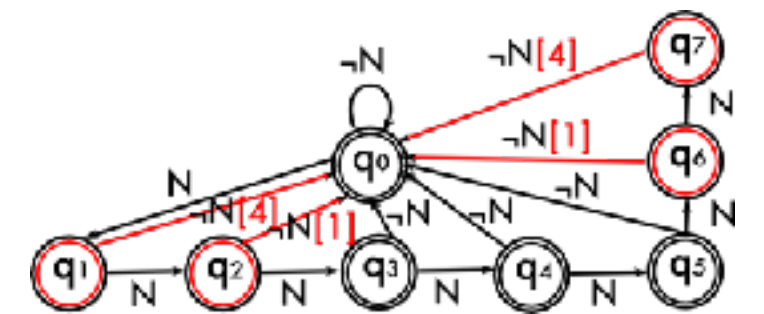
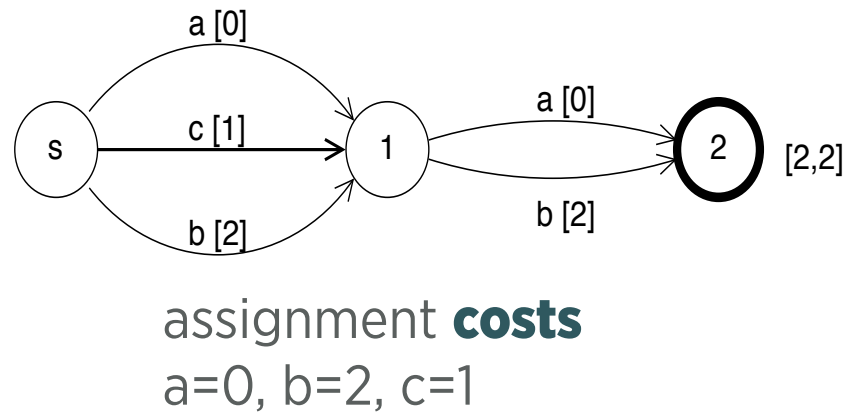
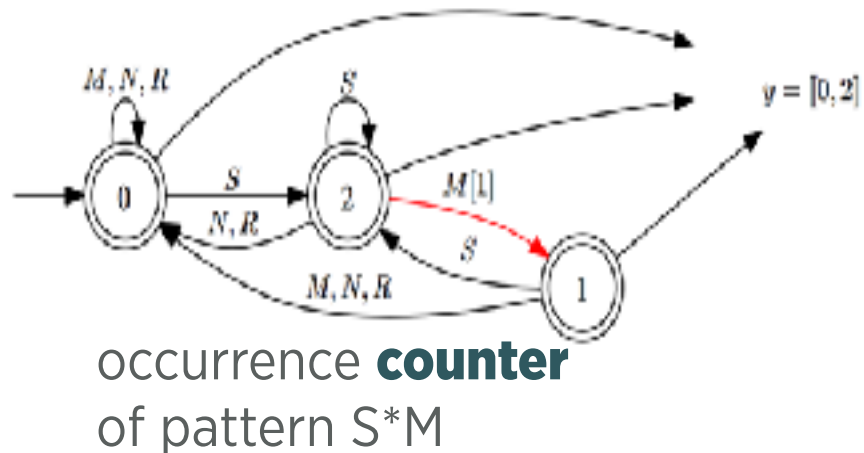


- unfold the automaton as a layered DAG of size  $n$
- remove the non final states in the last layer
- remove arcs which do not belong to any path from the first to the last layer
- synchronise the domain of  $X_i$  with the labels of the arc set in layer  $i$ : remove arcs or domain values

- GAC, same complexity  $O(n \times |Transitions|)$

# GENERALISATION TO WEIGHTED AUTOMATA

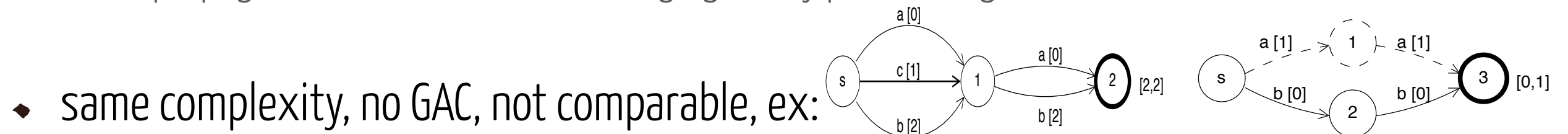
- add weights on the transitions and a bounded variable  $Z$ : a solution  $(Z, X_1, \dots, X_n)$  is an accepted word  $X_1X_2 \dots X_n$  with the sum of the transition weights equal to  $Z$
- increase the expressivity or make the model more compact



- filtering with **decomposition** [Beldiceanu04]  $\left\{ \begin{array}{l} (S_i, X_i, S_{i+1}, W_i) \in \text{WeightedTransitions} \\ \sum_i W_i = Z \end{array} \right.$
- dynamic programming cost-regular**  $(Z, X_1, \dots, X_n, (\pi, w))$  [Demasseey05]

maintain the shortest and longest paths:

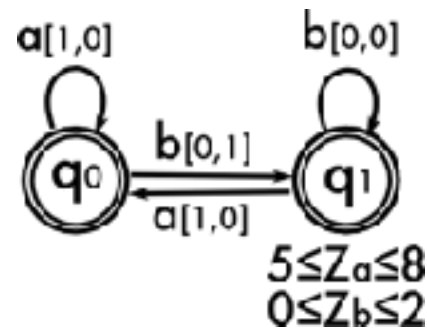
- label the nodes in the layered graph: min/max lengths from the first layer/to the last layer
- synchronise the bounds of  $Z$  with the min/max path length values
- backpropagation: remove arcs not belonging to any path of length within the bounds of  $Z$





# MULTICOST-REGULAR

- ◆ **vectors of weights:** to combine different counters/penalties/costs



occurrences of **a**  
and of sequences of **b**

- ◆ composition algorithms for finite automata can be efficiently extended [Menana10]
- ◆ underlying optimisation problem: resource-constrained shortest/longest paths are **NP-hard**
- ◆ filtering: extend automaton  $\left\{ \begin{array}{l} (S_i, X_i, S_{i+1}, W_i^0, \dots, W_i^K) \in \text{WeightedTransitions} \\ \sum_i W_i^k = Z^k, \forall k \end{array} \right.$
- ◆ or apply cost-regular independently on each dimension  $\bigwedge_k \text{costregular}(Z^k, X, (\Pi, w^k))$   
save memory & time vs. posting k costregular constraints
- ◆ or consider a tighter relaxation: **lagrangian relaxation** (filtering by dynamic programming would require to maintain too many labels)
- ◆ adapt lagrangian relaxation-based **variable fixing** to constraint filtering [Sellmann04]

# LAGRANGIAN RELAXATION-BASED FILTERING

- resource constrained shortest path ILP model:  $x_{ai} = 1 \iff X_i = \text{label}(a), \forall \text{arc } a \text{ in layer } i$

$$z^0 = \min f^0(x) = \sum_{a,i} w_{ai}^0 x_{ai}$$

$$\underline{Z}^k \leq \sum_{a,i} w_{ai}^k x_{ai} \leq \bar{Z}^k, \forall k = 1..K$$

complicating constraints

$$x \in \text{Paths} \subseteq \{0,1\}^{A \times n}$$

$$z^0(u) = \min_{x \in \text{Paths}} f_u^0(x) = \sum_{a,i} w_{ai}^0 x_{ai}$$

$$+ \sum_k u_k^- (\underline{Z}^k - \sum_{a,i} w_{ai}^k x_{ai})$$

$$+ \sum_k u_k^+ (\sum_{a,i} w_{ai}^k x_{ai} - \bar{Z}^k)$$

- dualisation** = relax and penalise violations  $\longrightarrow$  **lagrangian subproblem** with **multipliers**  $u$

- a shortest path problem providing a lower bound:  $z^0(u) \leq z^0, \forall u \geq 0$

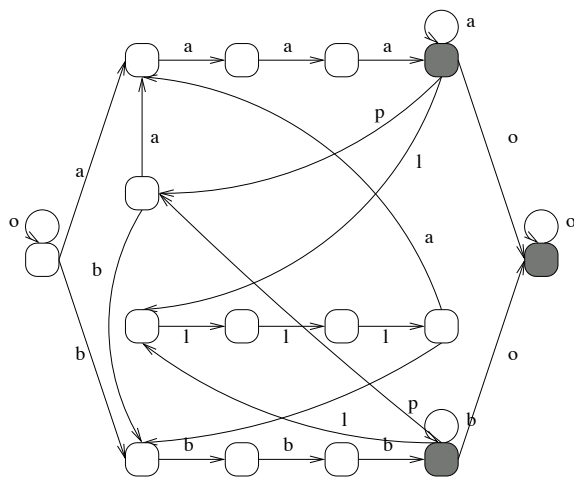
- then filtering:  $z^0(u) \leq z^0 \leq Z^0 \Rightarrow \underline{Z}^0 := \max(\underline{Z}^0, z^0(u))$

- and back-propagation:  $z^0(u)_{x_{ai}=1} > \bar{Z}^0 \Rightarrow z_{x_{ai}=1}^0 > \bar{Z}^0 \Rightarrow$  filter arc  $a$  in layer  $i$

- this is **cost-regular filtering** with weights:  $w_{ai}^0 + \sum_k (u_k^+ - u_k^-) w_{ai}^k, \forall a, i$

# MULTICOST-REGULAR ALGORITHM

- ◆ apply cost-regular filtering for different values of  $k$  and  $u$
- ◆ for a given  $k$ ,  $\max z^k(u)$  gives the best lower bound, thus the best filtering for  $Z^k$  but not necessary the best back-propagation:  $z^k(u) \geq z^k(u') \not\Rightarrow z^k(u)_{x_{ei}=1} \geq z^k(u')_{x_{ei}=1}$
- ◆ default parameters in the Choco implementation: fix  $u = 0$  for all  $k \geq 1$ , solve maximum 30 iterations of the subgradient algorithm to maximise  $z^0(u)$
- ◆ not GAC, not even monotonic, parameter-dependant, a rather high complexity,... but it works:



$n=96$  variables, 20 instances/#act, 10 min  
15 (#act=1) to 505 (#act=50) transitions

#act	cost-regular + GCC				multicost-regular			
	solved	proof(s)	best(s)	#nodes	solved	proof(s)	best(s)	#nodes
1	100%	0.3	0.2	225	100%	0.0	0.0	41
2	100%	0.6	0.3	393	100%	0.1	0.1	68
4	100%	2.9	2.3	1199	100%	0.2	0.1	67
8	100%	17.9	13.2	3597	100%	0.3	0.2	52
10	100%	50.0	47.7	7615	100%	0.4	0.4	63
15	100%	58.1	47.1	6233	100%	0.8	0.7	63
20	100%	58.1	44.0	4577	100%	1.2	1.0	64
30	80%	153.1	127.4	6080	100%	1.8	1.5	62
50	40%	460.0	65.6	6747	100%	5.0	4.8	65


# TO GO FURTHER WITH LANGUAGES & CP

- ◆ expose internal informations to derive branching strategies. ex:  $\text{regret}(X_i=v) = \text{length difference between a shortest path through an arc labelled } v \text{ in layer } i \text{ and a shortest path}$  (how to **compose branching strategies** ?)
- ◆ automata are for **automatic models**: algorithms to generate multi-weighted automata with a DSL dedicated to nurse rostering, to compose/filter automata, to relax (soft constrs) [Menana11]
- ◆ beyond CP: automatic **linearisation** [Côté13]
- ◆ **meta-constraint**: automata can not only model sequencing/counting rules but CSP solution sets, i.e. global constraints, see >300 constraints of the global constraint catalog (automaton model is dependant on the order of the variables) <https://sofdem.github.io/gccat/>
- ◆ alternative models and algorithms: grammar constraint (free-context languages) and **multi-valued decision diagrams** (concise models)



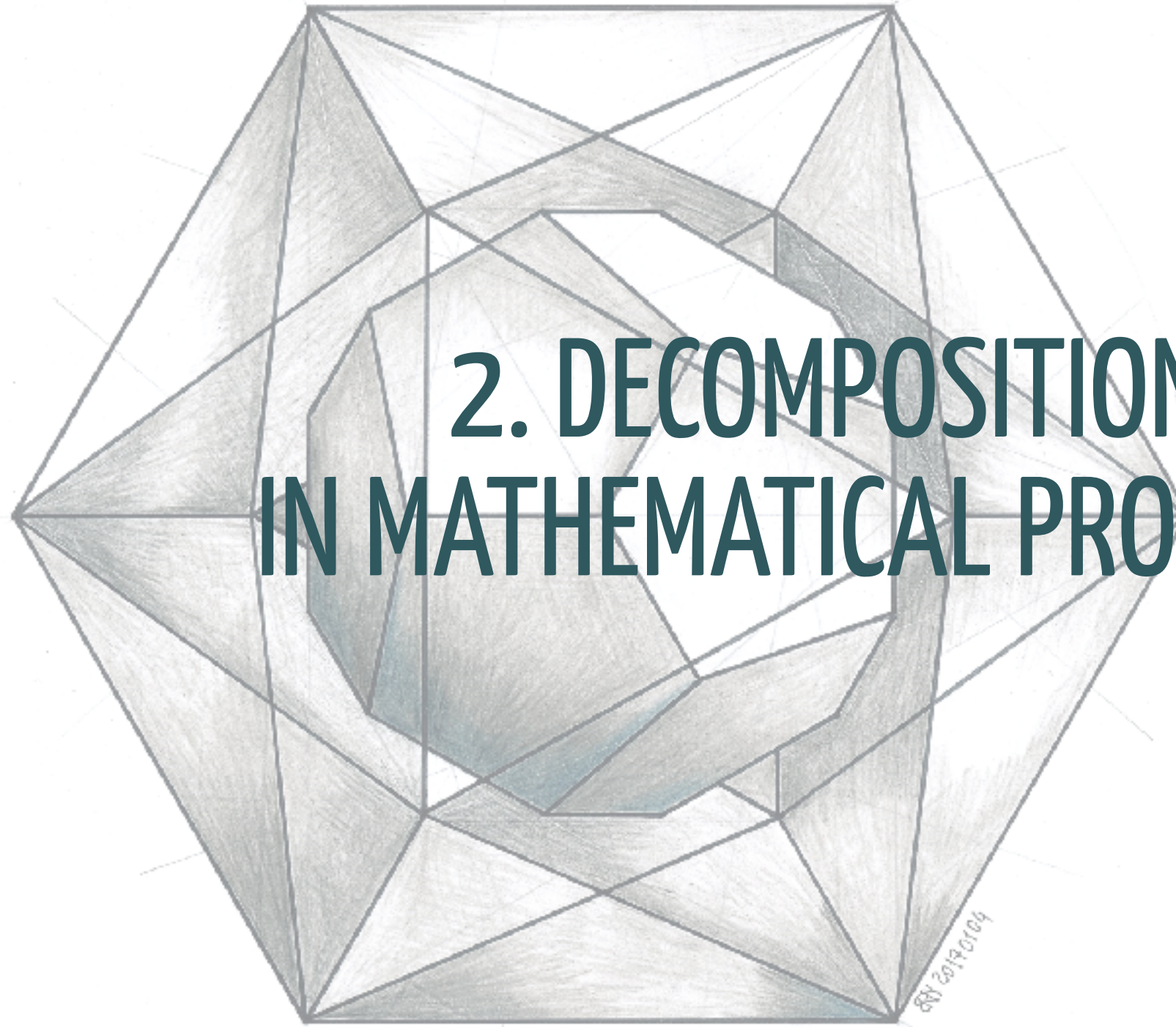
# OPEN QUESTIONS



- ◆ branch on packing then on scheduling vars
- ◆ deactivate heavy scheduling ctr while packing
- ◆ packing  scheduling

A handwritten roster or schedule grid for Monday through Friday. The grid is divided into columns for each day and rows for time slots. The time slots are labeled with times like 8:20, 8:55, 9:30, 9:50, 10:25, 10:50, 11:00, 11:30. The employee names are written in the cells, such as MR KIRII, MR NJAG, MRS MACA, MRS IAU, MR NJI, MAI NJ, MR MU, TR NJ, MISS, and MRS. The grid is filled with handwritten text, including numbers and names, representing a detailed schedule.

- ◆ one multcost-regular for each employee
- ◆ one global-cardinality for each day, for each act
- ◆ **better integration ? also with objective ?**



## **2. DECOMPOSITION METHODS IN MATHEMATICAL PROGRAMMING**

BY 2019/01/04



# INTEGER LINEAR PROGRAMMING IN A TINY NUTSHELL

- integer linear program ILP (mixed MILP if not all variables are integer)

$$z = \min\{cx \mid Ax \geq b, x \in \mathbb{Z}^n\}, c \in \mathbb{R}^n, A \in \mathbb{R}^m \times \mathbb{R}^n, b \in \mathbb{R}^m$$

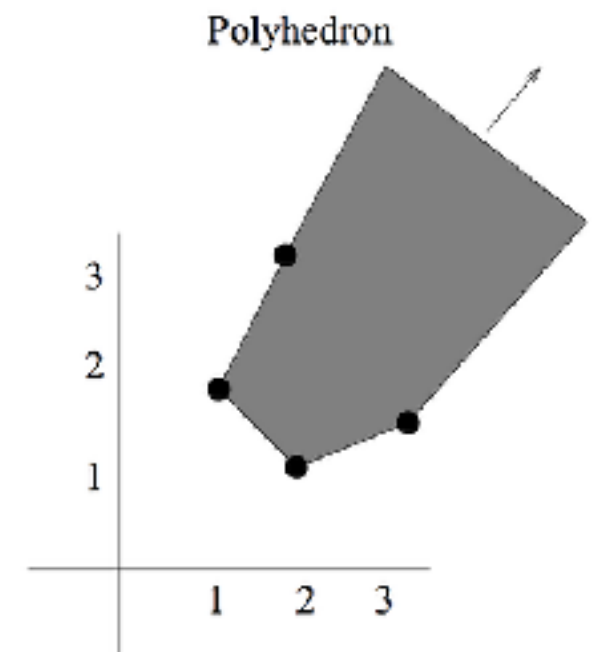
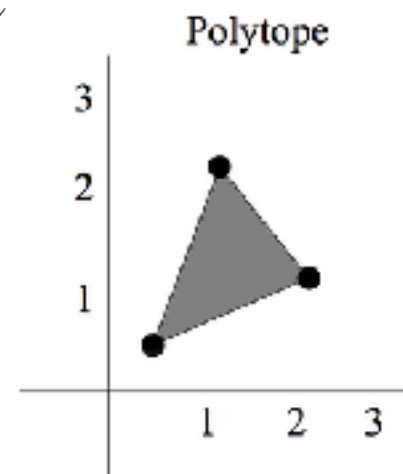
- $X = \{x \in \mathbb{Z}^n \mid Ax \geq b\}$  are the integer points of a **polyhedron**  $\bar{X} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$

- Weyl-Minkowski: for some  $x^1, \dots, x^p \in \mathbb{R}^n$  (extreme points)  $r^1, \dots, r^q \in \mathbb{R}^n$  (extreme rays):

$$\bar{X} = \left\{ \sum_s \lambda_s x^s + \sum_t \mu_t r^t \mid \sum_s \lambda_s = 1, \lambda \in \mathbb{R}_+^p, \mu \in \mathbb{R}_+^q \right\}$$

- solving the LP relaxation is easy:

$$\bar{z} = \min\{cx \mid x \in \bar{X}\} = -\infty \text{ or } \min_s cx^s$$



# INTEGER LINEAR PROGRAMMING IN A TINY NUTSHELL

- integer linear program ILP (mixed MILP if not all variables are integer)

$$z = \min\{cx \mid Ax \geq b, x \in \mathbb{Z}^n\}, c \in \mathbb{R}^n, A \in \mathbb{R}^m \times \mathbb{R}^n, b \in \mathbb{R}^m$$

- $X = \{x \in \mathbb{Z}^n \mid Ax \geq b\}$  are the integer points of a **polyhedron**  $\bar{X} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$

- Weyl-Minkowski: for some  $x^1, \dots, x^p \in \mathbb{R}^n$  (extreme points)  $r^1, \dots, r^q \in \mathbb{R}^n$  (extreme rays):

$$\bar{X} = \left\{ \sum_s \lambda_s x^s + \sum_t \mu_t r^t \mid \sum_s \lambda_s = 1, \lambda \in \mathbb{R}_+^p, \mu \in \mathbb{R}_+^q \right\}$$

- solving the LP relaxation is easy:

$$\bar{z} = \min\{cx \mid x \in \bar{X}\} = -\infty \text{ or } \min_s cx^s$$

- strong duality occurs in LP

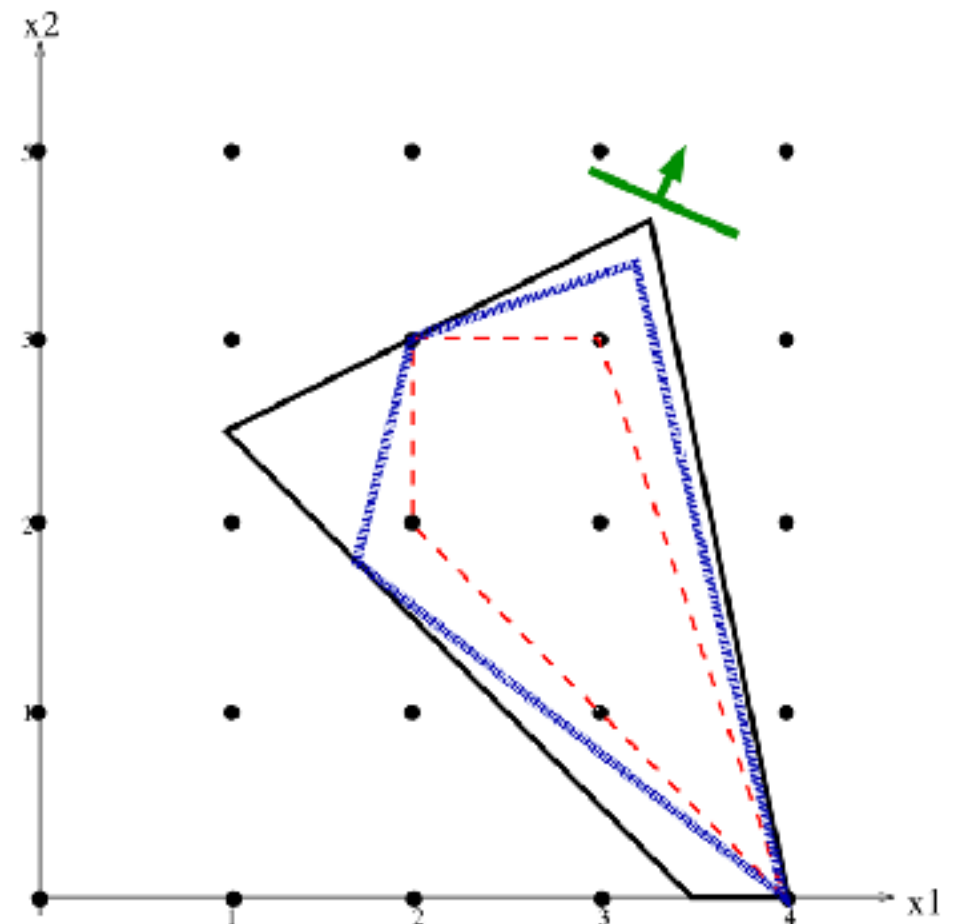
$$\bar{z} = \max\{ub \mid uA = c, u \in \mathbb{R}_+^m\}$$

- LP bound to prune nodes in branch-and-bound

$$\bar{z} \leq z$$

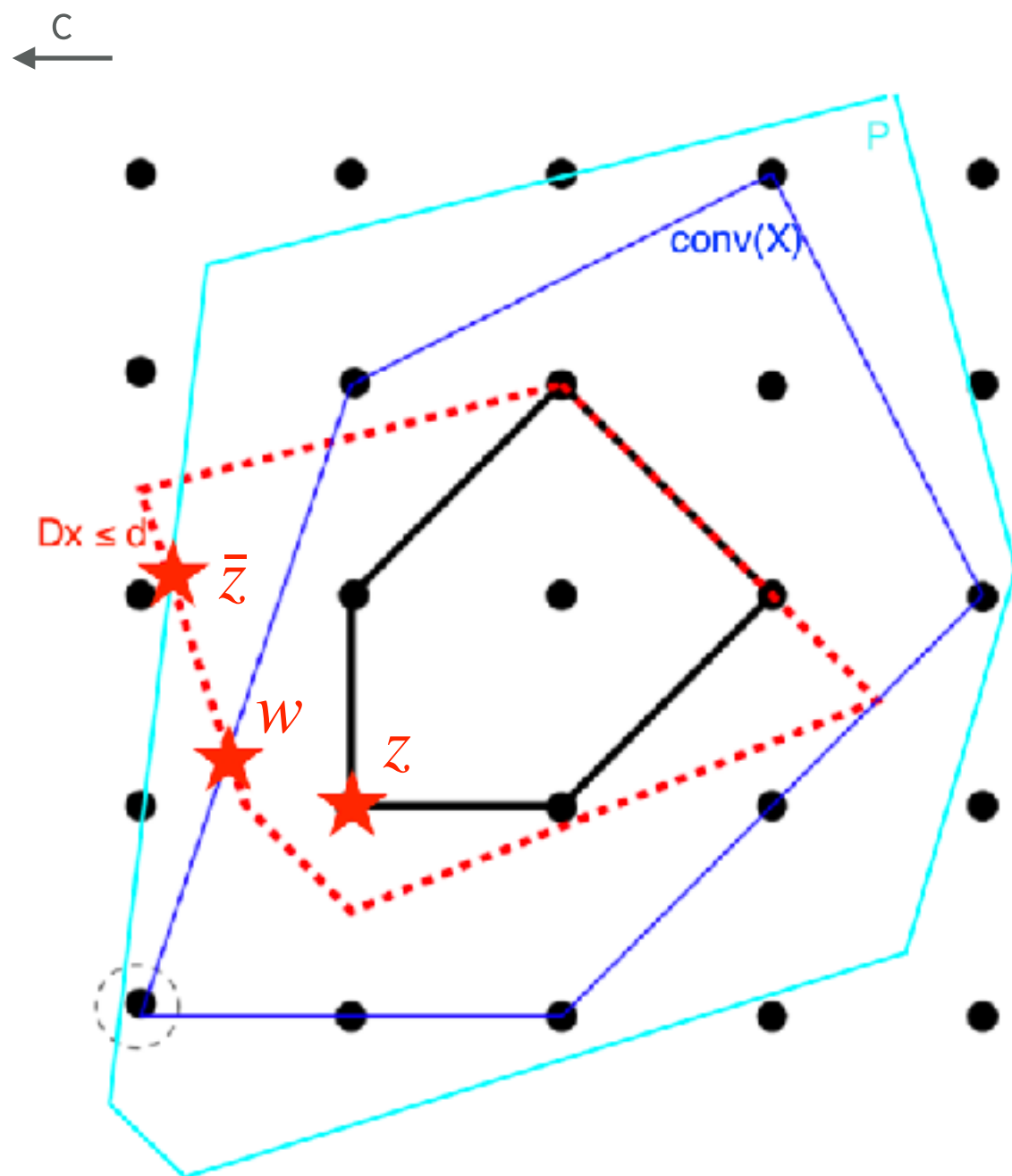
- different models, different bounds

ideal model, tightest bound:  $\bar{X} = \text{conv}(X)$





# LAGRANGIAN RELAXATION & COLUMN GENERATION



# Example 2 (continuation) NURSE SCHEDULING

assign activities to every employee every time, cover the load, satisfy the working rules

challenges:

- **two orthogonal problems: schedule activities for each employee / assign employees to each activity**
- numerous and heterogeneous working rules, hard and soft
- the model (data, objective and constraints) changes from instances to instances

# A MIXED INTEGER LINEAR PROGRAM (MILP)

- assignment cost  $c_{ai}$  and minimum cover  $d_{ai}$  for each activity  $a$  and period  $i$ ,
- $x_{eai}=1$  if employee  $e$  is assigned to activity  $a$  in period  $i$

$$z = \min f(x) = \sum_{e,a,i} c_{ai} x_{eai}$$

$$\sum_e x_{eai} \geq d_{ai} \quad \forall a, i$$


**coupling constraints**

$$x_e \in Schedules \subseteq \{0,1\}^{A \times n} \quad \forall e$$

$$z(u) = \min f_u(x) = \sum_{e,a,i} c_{ai} x_{eai}$$

$$+ \sum_{a,i} u_{ai} (d_{ai} - \sum_e x_{eai})$$

$$x_e \in Schedules \subseteq \{0,1\}^{A \times n} \quad \forall e$$

- not a good formulation: hard to linearise the working rules, in a compact way, many symmetries, potentially weak LP bound
- **lagrangian relaxation** : for any  $u \geq 0$  solving  $z(u)$  is to find a schedule for each employee  $e$  of minimum assignment costs  $(c_{ai} - u_{ai})$   one multcost-regular

# LAGRANGIAN RELAXATION

- solve  $z(u)$  gives a **lower bound** and a **near-feasible solution**: some cover constraints may not be satisfied. If the solution is feasible then it is not necessary optimal for  $z$ .
- the **lagrangian dual** is the problem to find the best lower bound  $w = \max\{z(u) \mid u \geq 0\}$
- is  $w$  far from  $z$  (**dual gap**)? how to get feasible solutions from  $z(u)$ ? how to compute  $w$ ?

$$z = \min\{cx \mid Dx \geq d, x \in X\} \quad \text{assume: } X = \{x \in \mathbb{Z}^n \mid Ax \geq b\}, \text{conv}(X) = \text{conv}(x^s)_{s=1..p}$$

$$z(u) = \min\{cx + u(d - Dx) \mid x \in X\} = \min_{s=1..p}\{cx^s + u(d - Dx^s)\}$$

$$w = \max_{u \geq 0} z(u) = \max_{u \geq 0, y}\{y \mid y \leq cx^s + u(d - Dx^s), \forall s = 1..p\}$$

$$= \min_{\lambda \geq 0}\left\{\sum \lambda_s cx^s \mid \sum \lambda_s (d - Dx^s) \leq 0, \sum \lambda_s = 1\right\} \quad (\text{strong duality in LP})$$

$$= \min\{cx \mid Dx \geq d, x \in \text{conv}(X)\}$$

- $\bar{z} \leq w \leq z$  with  $\bar{z} = w$  if  $\bar{X} = \text{conv}(X)$
- what to dualise: (1) **coupling constraints** so that  $z(u)$  decomposes  
(2) **not all complicating constraints** so that LR bound > LP bound

# HOW TO COMPUTE FEASIBLE SOLUTIONS

1. replace the LP relaxation **in branch-and-bound** by lagrangian relaxation
2. repair the constraint violations in the subproblem solutions **with local search:**
  - ex (rostering): subproblem solution = valid schedule for every employee
  - try to repair the violated covers  $\sum_e x_{eai} < d_{ai}$  by swapping activities



# HOW TO SOLVE ? IN THE DUAL SPACE

- ◆ the **lagrangian function** is **concave non-smooth (piecewise linear)**

$$u \mapsto z(u) = \min_{s=1..p} \{ cx^s + u(d - Dx^s) \}$$

- ◆ maximise by computing iteratively  $z(u_k), k = 0, \dots, K$

- ◆ **subgradient** algorithm:

- follow a possible ascent direction (subgradient)  $u_{K+1} = \max(0, u_K + t_K(d - Dx^K))$
- simple to implement but lack of ascent, slow convergence, no stopping test  $0 \in \delta z$

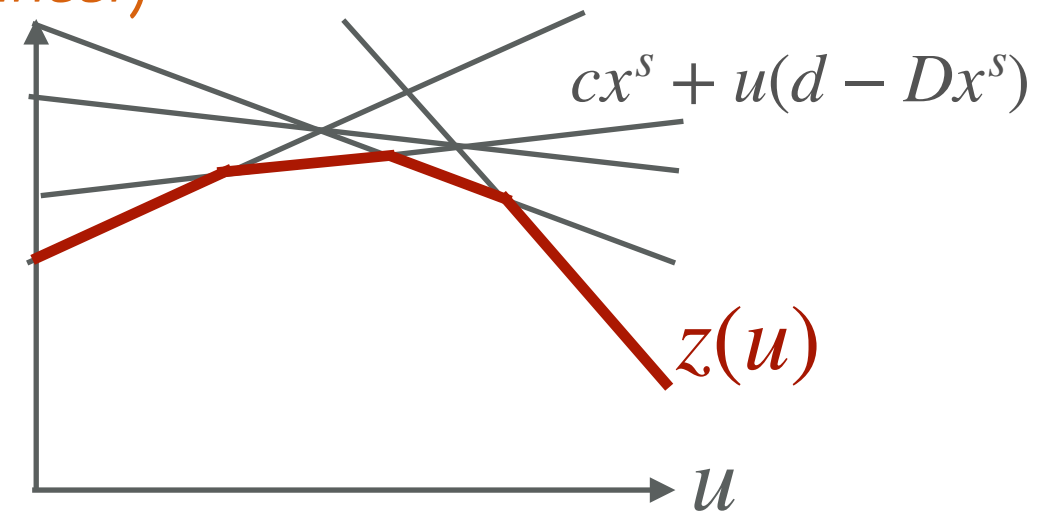
- ◆ **Kelley cutting-plane** algorithm:  $u_{K+1} \in \operatorname{argmax}_{u \geq 0, y} \{ y \mid y \leq cx^k + u(d - Dx^k), \forall k \leq K \}$

- simple to implement, better (finite) convergence but unstable and the LP becomes huge

- ◆ **bundle methods**: stabilisation and aggregation to remedy these drawbacks, e.g. (proximal):

$$u_{K+1} \in \operatorname{argmax}_{u \geq 0, y} \{ y + \mu_K \|u - u_K^*\| \mid y \leq cx_*^k + u(d - Dx_*^k) + \epsilon_k \}$$

- ◆ ellipsoid/analytic center, in-out,...





# HOW TO SOLVE ? IN THE PRIMAL SPACE

$$w = \max_{u \geq 0} z(u) = \max_{u \geq 0, y} \{y \mid y \leq cx^s + u(d - Dx^s), \forall s = 1..p\}$$

$$= \min_{\lambda \geq 0} \left\{ \sum_s \lambda_s cx^s \mid \sum_s \lambda_s (d - Dx^s) \leq 0, \sum_s \lambda_s = 1 \right\}$$

**Dantzig-Wolfe decomposition** (separate convexity for each e)

$$z = \min f(x) = \sum_{e,a,i} c_{ai} x_{eai}$$

$$\sum_e x_{eai} \geq d_{ai} \quad \forall a, i$$

$$x_e \in Schedules \subseteq \{0,1\}^{A \times n} \quad \forall e$$



$$z = \min f(x) = \sum_{s,e,a,i} \lambda_{es} c_{ai} x_{ai}^s$$

$$\sum_s \lambda_{es} = 1 \quad \forall e$$

$$\sum_s \lambda_{es} x_{ai}^s \geq d_{ai} \quad \forall a, i$$

$$\lambda_{es}^{s,e} \in \{0,1\} \quad \forall e, \forall s \in S$$

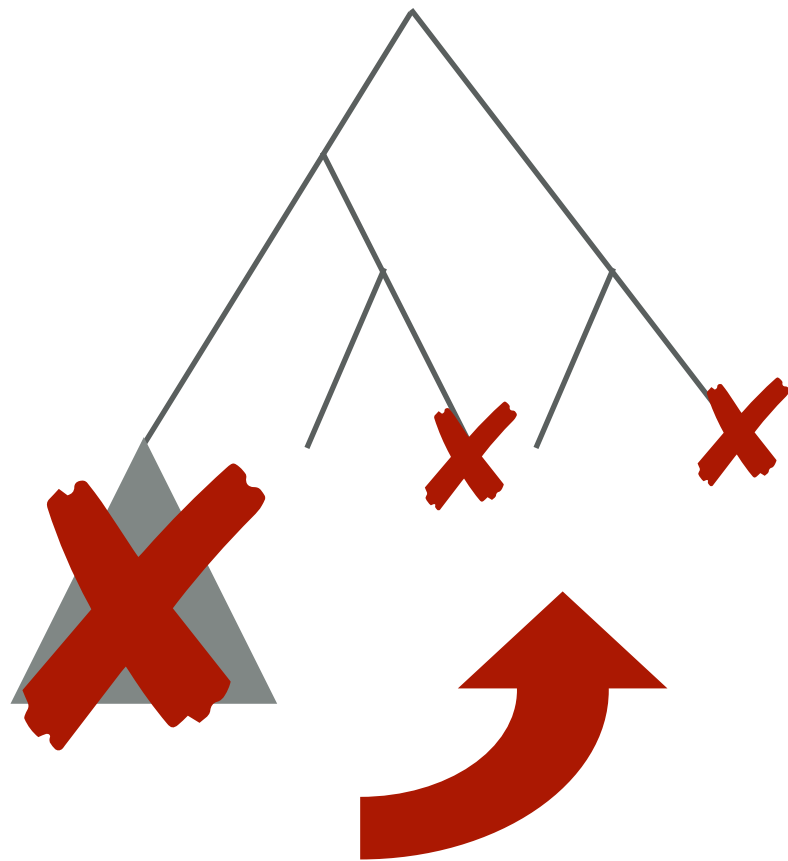
- LP relaxation: huge number of variables, most being 0 in any optimal solution (non-basic)
- simplex: iteratively find the largest negative reduced cost  $x \in \operatorname{argmin} cx + u_k(d - Dx) - w_k$
- **column generation**: the same but the non-basic variables are not all made explicit
- **LP bound = LR dual**, column generation/primal = Kelley cutting plane algorithm/dual

# COLUMN GENERATION & BRANCH-AND-PRICE

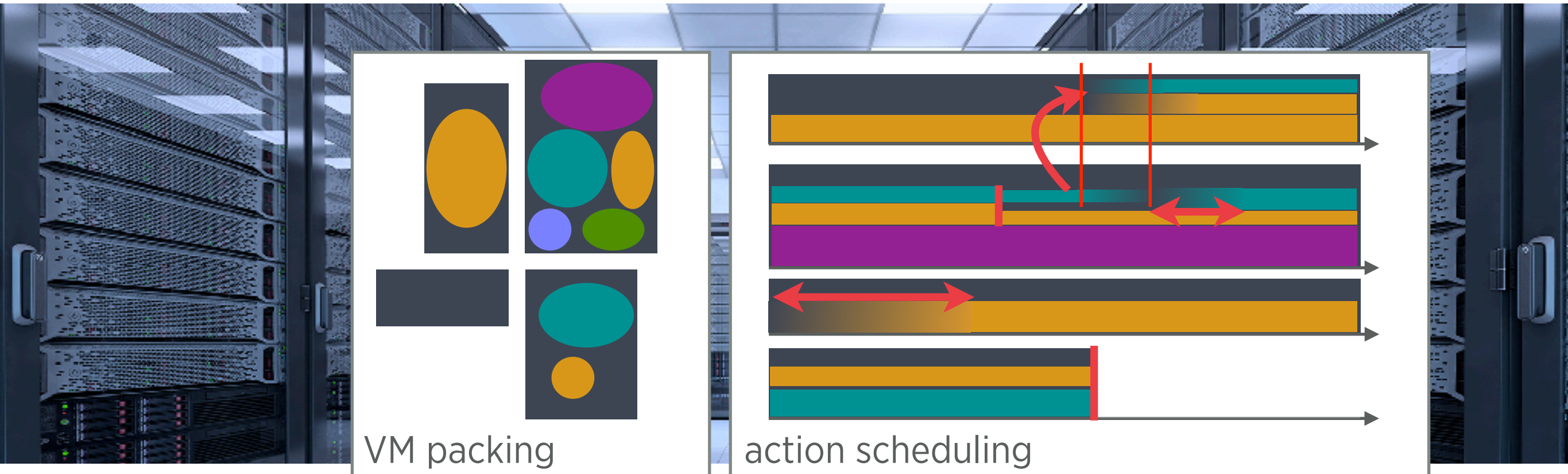
- ◆ the basic algorithm may be **slow to converge** and suffers from **instability**
- ◆ **improvement** techniques:
  - do not solve the subproblem at optimality: find a negative reduced cost
  - add several columns at each iteration / remove inactive columns
  - stabilisation techniques (similar to bundle methods)
- ◆ provides a way to **break symmetries**
- ◆ how to compute feasible (integral) solutions:
  - try rounding
  - branch-and-bound on the restricted LP
  - branch-and-price: generate columns at each node
  - like with LR, branch on the  $x$  decisions not on the master variables
  - unlike with LR, the primal (fractional) solution can be used for cut generation

$$\begin{aligned} z = \min f(x) &= \sum_{s,a,i} \lambda_s c_{ai} x_{ai}^s \\ \sum_s \lambda_s &= |E| \\ \sum_s \lambda_s x_{ai}^s &\geq d_{ai} \quad \forall a, i \\ \lambda_s &\in \mathbb{N} \quad \forall s \in S \end{aligned}$$

# BENDERS DECOMPOSITION & BRANCH-AND-CHECK

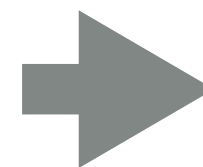


# Example 1 (continuation) REAL-TIME DATACENTER RESOURCE MANAGEMENT



challenge: *integrated packing/scheduling problem*

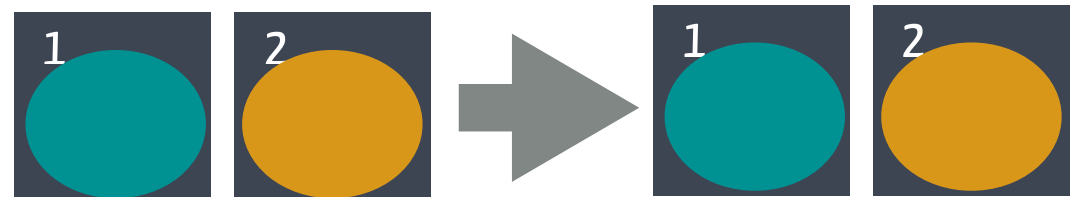
- the scheduling decisions depend on the final packing



VM migrates live  
or migrates off

- but the final packing depends also on the scheduling decisions:

packing can be proved unfeasible or suboptimal



# INTEGRATE PACKING/SCHEDULING

- ◆ current approach: branch first on the packing variables and deactivate the heavy scheduling constraints in this phase, then solve scheduling. If unfeasible or suboptimal, then backtrack:  
 $assign(VMs) \neq pack(VMs)$
- ◆ idea: identify a (minimal) partial assignment that causes unfeasibility/suboptimality, and generate a **nogood** constraint to discard it:  $assign(V) \neq pack(V), V \subseteq VMs$   
*ex* :  $duration(migration(VM1)) > incumbent \Rightarrow action(VM1) \in \{stay, stop\}$   
 $\Rightarrow assign(VM1) \in \{initialpack(VM1), noPM\}$
- ◆ **branch-and-check** [Hooker00, Thorsteinsson01]. Remark that activating scheduling propagation allows to infer inconsistencies from partial assignments (but may be too costly)
- ◆ **logic-based Benders decomposition** [Hooker00]: solve packing at optimality before calling the scheduling subproblem (usually more efficient if the subproblem is hard)
- ◆ nogoods are often problem-specific but can be generated from conflict analysis

# BENDERS DECOMPOSITION IN MP

- ◆ **variable decomposition**: delay the evaluation of the slave subproblem then tighten the master
- ◆ decomposition scheme from integer linear programming [Benders63] and convex programming [Geoffrion72] when **strong duality** applies to the subproblem (ex LP or convex NLP): use duality information to generate nogoods
- ◆ different implementations: can evaluate the slave at optimal master solutions (traditional), at integer solutions (if the subproblem is easy to check), at fractional solutions (if the subproblem can be formulated from partial master solutions)
- ◆ easy to implement in modern branch-and-cut solvers with callback constraints
- ◆ successful on stochastic programming (called the L-shaped method)



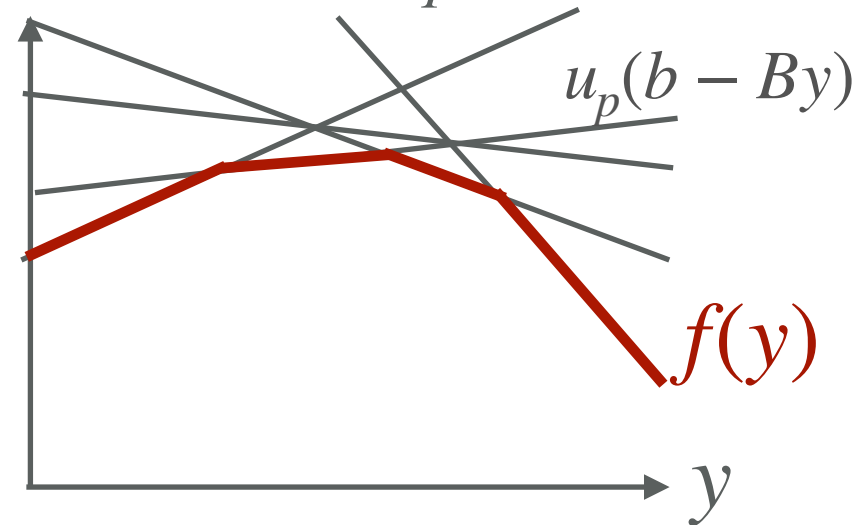
# BENDERS DECOMPOSITION FOR MILP

$$\begin{aligned}
 M &= \max \{cx + dy \mid Ax + By \leq b, x \geq 0, y \in Y\} \\
 &= \max_{y \in Y} \{dy + f(y)\}, \quad f(y) = \max \{cx \mid Ax \leq b - By, x \geq 0\} \\
 & \quad \quad \quad = \min \{u(b - By) \mid uA \geq c, u \geq 0\}
 \end{aligned}$$

$$\{uA \geq c, u \geq 0\} = \text{conv}((u_p)_p) + \text{cone}((r_q)_q)$$

$$M = \max_{y \in Y} \{dy + z \mid z \leq u_p(b - By) \forall p, 0 \leq r_q(b - By) \forall q\}$$

- remember the lagrangian dual?  
exercise: dualise the coupling constraints  
difference: here Y can be integral



- solve restricted master  $M_p$ : get  $y_p$ . Solve slave  $f(y_p)$ : get dual  $u_{p+1}$  if feasible ( $r_{q+1}$  otherwise). Tighten master:  $M_{p+1}$  then iterate if

$$\max_p (dy_p + f(y_p)) = LB < UB = \text{opt}(M_p)$$

# Example 3

## PUMP DESIGN IN WATER DISTRIBUTION NETWORKS



choose pumps, satisfy the demand, minimise investment+operation costs

challenges:

- two integrated decision levels:
- investment/long-term: investment, maintenance, ageing, uncertain demand/price/availability
- operation/short-term: pump scheduling, non-convex hydraulic constraints
- various types of networks: looped/branched, fixed/variable-speed pumps,

# GENERALISED BENDERS WITH STABILISATION

## APPLICATION TO PUMP DESIGN [BONVIN19]

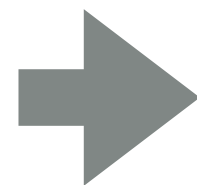
- ◆ master problem: select pumps to install:  $y_{pn} = 1$  if at least  $n$  pumps of type  $p$  installed
- ◆ slave problem: compute the expected minimum operational cost over the next 20 years for configuration  $y^*, n_p^* = \sum y_{pn}^*, \forall p$
- ◆ evaluate over  $D$  representative days and one critical day with high demand and one pump outage : slave decomposes in  $D+1$  daily pump scheduling problems (non-convex MINLPs)
- ◆ solve first the the critical day as a satisfaction problem: if unfeasible, at least one more pump of any type is required:  $\sum_p y_{pn_p^*} \geq 1$ . Using dominance, we can generate other unfeasible configurations.
- ◆ otherwise solve a convex NLP relaxation of each representative day (expected to be feasible)

$$f_d(y) = \min \sum_{p,n,t} \alpha_{pt} q_{pnt} + \beta_{pt} x_{pnt}$$

$$s.t. : x_{pnt} \leq y_{pn} \forall p, n, t$$

$$\sum_i q_{ijt} = D_{jt}^d + S_j(h_{t+1} - h_{jt}), \forall t, j \in Tanks$$

$$(x_t, q_t, h_t) \in F_t \subseteq \{0,1\}^{PN} \times \mathbb{R}_+^A \times \mathbb{R}_+^V$$



get the operation cost  $f_d(y)$   
 and one subgradient  $s_d(y) \in \delta f_d(y)$   
 (from duals of  $x_{pnt} \leq y_{pn}$  )

# BENDERS APPLIED TO PUMP DESIGN (CONTINUATION)

- ◆ Bender's master:

$$\begin{aligned} \min g(y) &= \sum_{p,n} C^p y_{pn} + \sum_d N_d z_d \\ \text{s.t.} : y_{pn} &\geq y_{pn+1} \quad \forall p, n \\ z_d &\geq f_d(y^k) + s_d(y^k)(y - y^k) \quad \forall d, k = 1..K \\ \sum y_{N_p^l} &\geq 1 \quad \forall l = 1..L \\ y &\in \{0,1\}^{PN}, z \in \mathbb{R}^D \end{aligned}$$

- ◆ with  $g_{lev} = \frac{\underline{g}_K + \bar{g}_K}{2}$ ,  $\underline{g}_0 = 0$ ,  $\bar{g}_0 = +\infty$

- ◆ if infeasible update LB  $\underline{g}_{K+1} = g_{lev}$

- ◆ otherwise solve slave and possibly update UB and stabilisation center:

$$\bar{g}_{K+1} := \min(\bar{g}_K, \sum_{p,n} C^p y_{pn}^{K+1} + \sum_d N_d f_d(y^{K+1})), \quad \bar{g}_{K+1} < \bar{g}_K \implies y_* := y^{K+1}$$

- ◆ stop when  $\frac{\bar{g} - \underline{g}}{\bar{g}} < \epsilon$

- ◆ with level stabilisation:

$$\min \|y - y_*\|_2^2 \quad (\text{linear because } y \text{ binary})$$

$$\text{s.t.} : y_{pn} \geq y_{pn+1} \quad \forall p, n$$

$$g_{lev} \geq g(y^k) + s(y^k)(y - y^k) \quad \forall k = 1..K$$

$$\sum y_{N_p^l} \geq 1 \quad \forall l = 1..L$$

$$y \in \{0,1\}^{PN}, z \in \mathbb{R}^D$$

# TAKE-AWAY

- ◆ decomposition methods are **flexible** solutions for practical **composite/large-scale** problems
- ◆ how deep to decompose: **trade-off** between performance and flexibility
- ◆ **CP/global constraints**: easy way to implement decomposition/hybridisation



# TAKE-AWAY

- ◆ decomposition methods are **flexible** solutions for practical **composite/large-scale** problems
- ◆ how deep to decompose: **trade-off** between performance and flexibility
- ◆ **CP/global constraints**: easy way to implement decomposition/hybridisation
  - but not so easy to turn an optimisation algorithm to an **efficient filtering algorithm**
  - **partial decomposition** (propagation) to complement with a specialised search strategy (sequential branching, branch-and-check )

# TAKE-AWAY

- ◆ decomposition methods are **flexible** solutions for practical **composite/large-scale** problems
- ◆ how deep to decompose: **trade-off** between performance and flexibility
- ◆ **CP/global constraints**: easy way to implement decomposition/hybridisation
  - but not so easy to turn an optimisation algorithm to an **efficient filtering algorithm**
  - **partial decomposition** (propagation) to complement with a specialised search strategy (sequential branching, branch-and-check )
- ◆ **MP decomposition**: lagrangian relaxation, column generation, Benders decomposition
  - generic frameworks but their application is **problem-specific** (subproblem)
  - naive implementations may **not converge well**
- ◆ hybrid CP & MP: **complementary** orientations local/global, feasibility/optimality, logic/analytic
- ◆ stay curious: parts of your problem are perhaps well solved with other formalisms

# STAFF

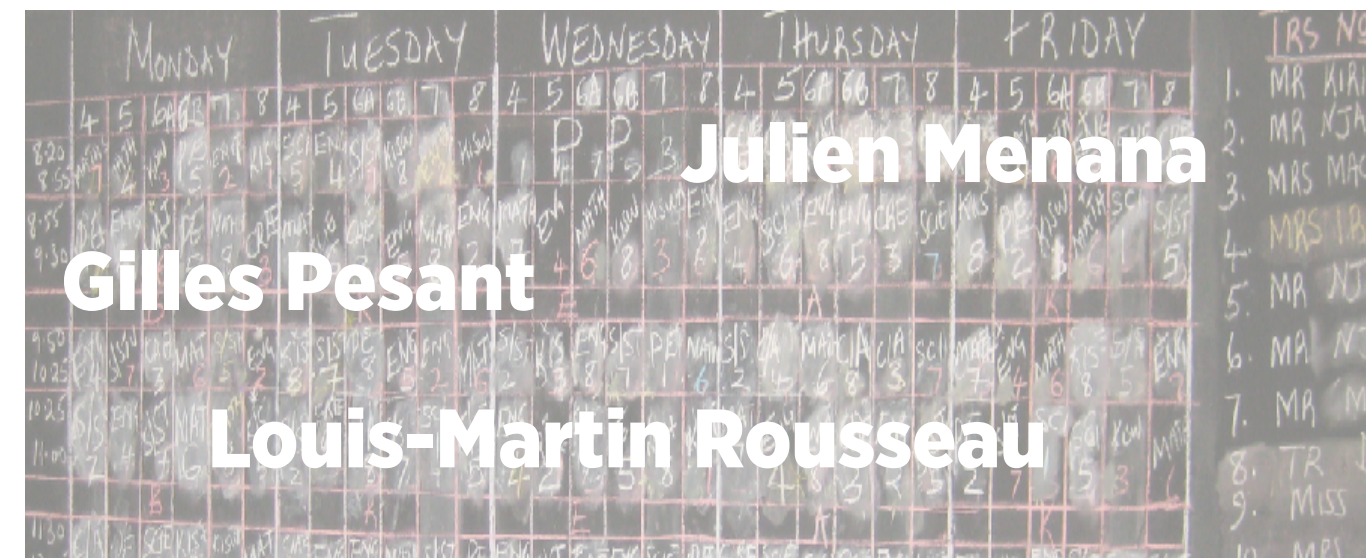


**Fabien Hermenier**



**Gratien Bonvin**

**Wellington de Oliveira**



**Gilles Pesant**

**Julien Menana**

**Louis-Martin Rousseau**

<https://sofdem.github.io/>