Prescriptive Data Analytics

Christine Solnon

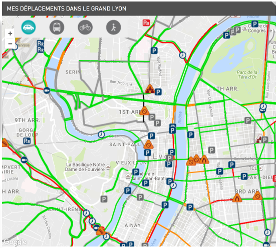
5IF - 2022-2023

Prescriptive Data Analytics

Four levels of Data Analytics

***Descriptive Analytics* :**
Extract Knowledge from Data

What are the traffic conditions right now?



***Diagnostic Analytics* :**
Explain why some events occur (XAI)

Why is there a traffic jam right now?

***Predictive Analytics* :**
Build models to predict future

What will be traffic conditions in 30 mn?

***Prescriptive Analytics* :**
Assist decision making / Make decisions

What is the best route if I leave at 8:25?

Four levels of Data Analytics

***Descriptive Analytics* :**

Extract Knowledge from Data

What are the traffic conditions right now?

***Diagnostic Analytics* :**

Explain why some events occur (XAI)

Why is there a traffic jam right now?

***Predictive Analytics* :**

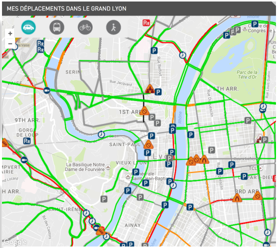
Build models to predict future

What will be traffic conditions in 30 mn?

***Prescriptive Analytics* :**

Assist decision making / Make decisions

What is the best route if I leave at 8:25?



Four levels of Data Analytics

Descriptive Analytics **:**

Extract Knowledge from Data

What are the traffic conditions right now?

Diagnostic Analytics **:**

Explain why some events occur (XAI)

Why is there a traffic jam right now?

Predictive Analytics **:**

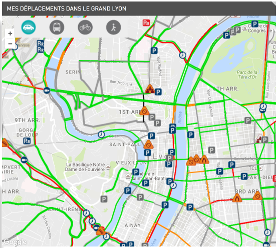
Build models to predict future

What will be traffic conditions in 30 mn?

Prescriptive Analytics **:**

Assist decision making / Make decisions

What is the best route if I leave at 8:25?



Four levels of Data Analytics

Descriptive Analytics **:**

Extract Knowledge from Data

What are the traffic conditions right now?

Diagnostic Analytics **:**

Explain why some events occur (XAI)

Why is there a traffic jam right now?

Predictive Analytics **:**

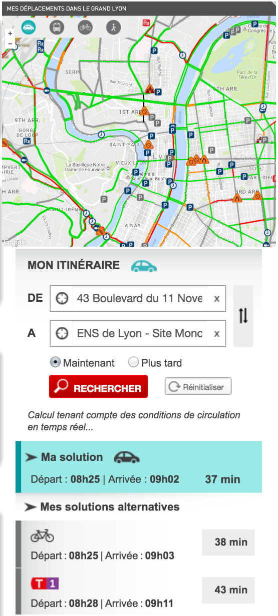
Build models to predict future

What will be traffic conditions in 30 mn?

Prescriptive Analytics **:**

Assist decision making / Make decisions

What is the best route if I leave at 8:25?



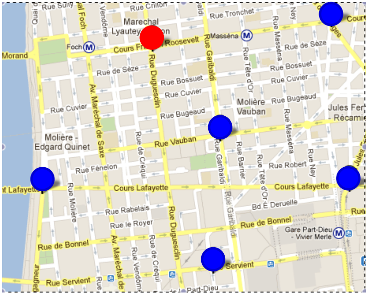
Basic urban delivery problem

Problem:

Given a city map and delivery addresses, compute the shortest tour

Solution process in two steps:

- Compute the shortest path graph
- Solve the Asymetric Traveling Salesman Problem (ATSP) in this graph



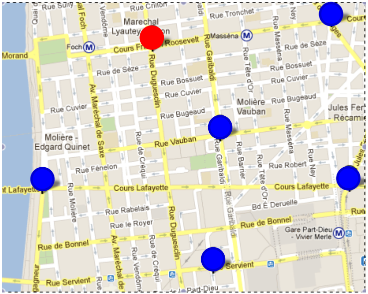
Basic urban delivery problem

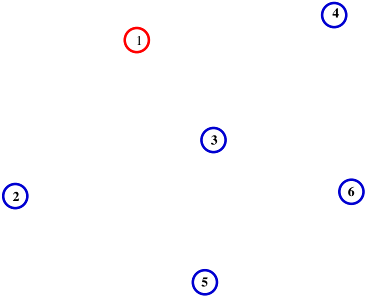
Problem:

Given a city map and delivery addresses, compute the shortest tour

Solution process in two steps:

- Compute the shortest path graph
- Solve the Asymetric Traveling Salesman Problem (ATSP) in this graph





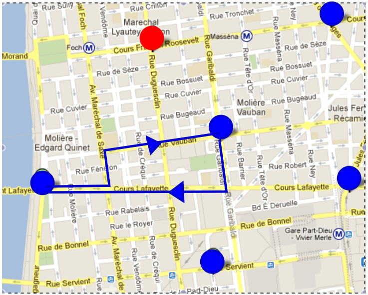
Basic urban delivery problem

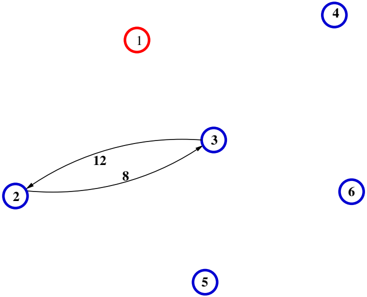
Problem:

Given a city map and delivery addresses, compute the shortest tour

Solution process in two steps:

- Compute the shortest path graph
- Solve the Asymetric Traveling Salesman Problem (ATSP) in this graph





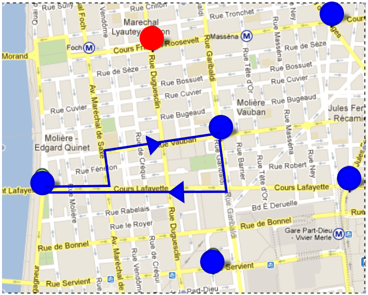
Basic urban delivery problem

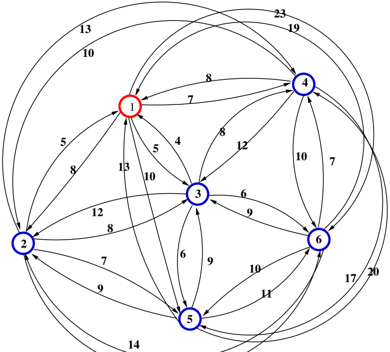
Problem:

Given a city map and delivery addresses, compute the shortest tour

Solution process in two steps:

- Compute the shortest path graph
- Solve the Asymetric Traveling Salesman Problem (ATSP) in this graph





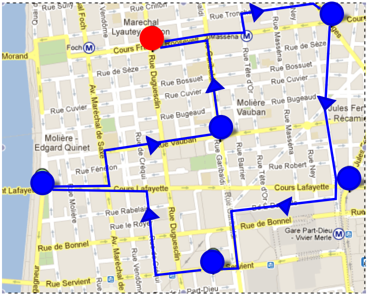
Basic urban delivery problem

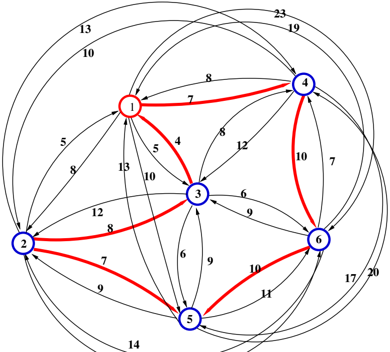
Problem:

Given a city map and delivery addresses, compute the shortest tour

Solution process in two steps:

- Compute the shortest path graph
- Solve the Asymetric Traveling Salesman Problem (ATSP) in this graph





Some Classical Variants

Vehicle Routing Problem (VRP)

- Deliveries are associated with demands
- There are several vehicles with limited capacities

Pickup and Delivery Problem (PDP)

Each request is composed of a pickup point and a delivery point
$\leadsto$ A delivery point cannot be visited before its corresponding pickup point

Dial A Ride Problem (DARP)

PDP with a limited capacity vehicle

Addition of Time Windows (TSP-TW, VRP-TW, PDP-TW, DARP-TW)

Each point i must be visited within a time-window

Various objective functions:

Travel duration, Arrival time, Number of vehicles, ...

What's difficult in classical problems?

Compute shortest paths between two points?

Easy and well solved problem:

- Efficient algorithms (polynomial time)
  $\rightsquigarrow$ For example, Dijkstra or A*

Compute optimal tours in the shortest path graph?

$\mathcal{NP}$-hard problems

- Theory: No algorithm can compute the optimal solution in polynomial time (unless $\mathcal{P} = \mathcal{NP}$)
- Practice: Use Artificial Intelligence!

What's difficult in classical problems?

Compute shortest paths between two points?

Easy and well solved problem:

- Efficient algorithms (polynomial time)
  $\rightsquigarrow$ For example, Dijkstra or A*

Compute optimal tours in the shortest path graph?

$\mathcal{NP}$-hard problems

- Theory: No algorithm can compute the optimal solution in polynomial time (unless $\mathcal{P} = \mathcal{NP}$)
- Practice: Use Artificial Intelligence!

New Data for New Problems

Classical problems:

- All Data are known when optimising tours $\rightsquigarrow$ Deterministic Problems
- Travel durations are constant $\rightsquigarrow$ Constant Problems

Ex: Average travel duration on section 18 = 42, ...

New Data $\rightsquigarrow$ New Problems:

- Probability distributions $\rightsquigarrow$ Stochastic Problems
  Ex: Probability(travel duration on section 18 = 42) = 0.4, ...
- Real-time Data revealed when realising tours $\rightsquigarrow$ Online Problems
  Ex: Actual travel duration on section 18 = 58
- Data which depends on time $\rightsquigarrow$ Time-Dependent Problems
  Ex: Travel duration at 8:00 = 42; Travel duration at 8:15 = 47; ...

Prescriptive Data Analytics

Context of this work

ASTRAL project [2014-2017]:

- Funded by IMU LabEx
- Partners: LICIT (IFSTTAR), LIRIS, and Métropole de Lyon
- Goal: Design predictive models for traffic forecasting in city centres

PhD thesis of Julien Salotti (defended in 2019):

- Co-supervised with R. Billot, N.-E. El Faouzzi, and S. Fenet
- Contributions:
  - Experimental evaluation of predictive models
  - Integration of causal information in predictive models

How to measure traffic conditions?

Spatio-temporal trajectories coming from the use of applications:

- Examples: GPS, Mobile phone communications with cellular networks
- Cons: Privacy issues, Spatial errors, Representativity issues, Property issues, ...

(Image from Romain Billot)

Electro-magnetic sensors:

- Physical detection of vehicles
- Cons: Incomplete spatial coverage

Data coming from electro-magnetic sensors

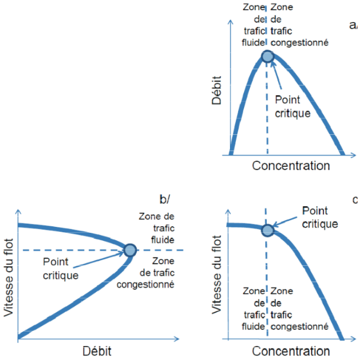
Dataset provided by Lyon Metropole:

- 634 sensors
- Two measures every 6 minutes:
  - Flow: Nb of vehicles per time period
  - Density: Nb of vehicles per road segment

Fundamental diagram:

Estimate speed given flow and density:

- Fluid traffic: flow increases when density increases
- Congested traffic: flow decreases when density increases

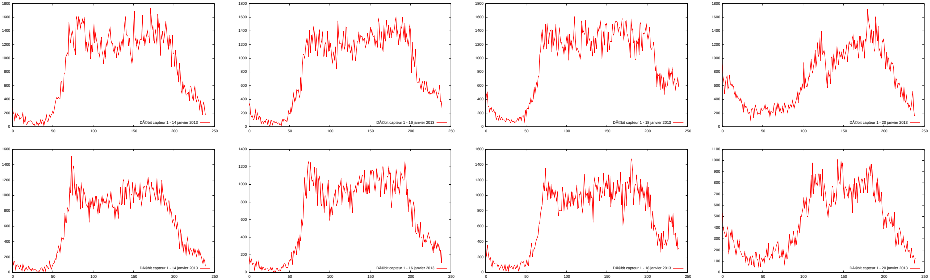


Reference:

Buisson and Lesort (2010): *Comprendre le trafic routier : Méthodes et calculs*

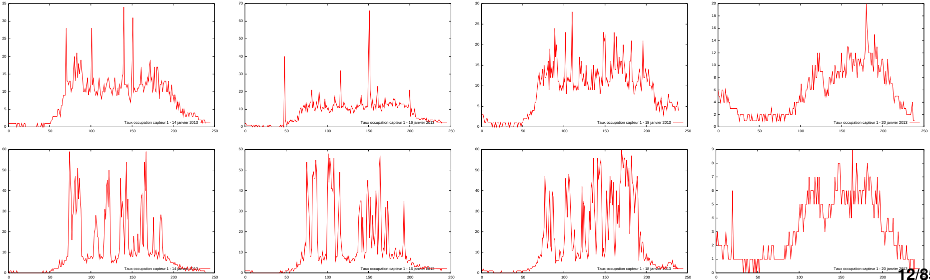
- Flow measured by sensors 1 (top) and 27 (bottom):

Monday 01/14/2013 | Wednesday 01/16/2013 | Friday 01/18/2013 | Sunday 01/20/2013



- Density measured by sensors 1 (top) and 27 (bottom):

Monday 01/14/2013 | Wednesday 01/16/2013 | Friday 01/18/2013 | Sunday 01/20/2013



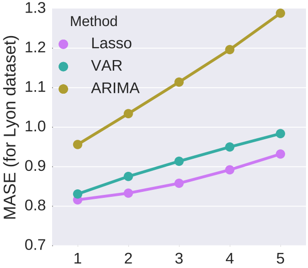
Predictive models: short term predictions (less than 1 hour)

Compared approaches:

- Univariate (U) vs Multivariate (M)
- Variable selection (S)
- Linear (L) vs Non Linear (NL)

Method	L	NL	U	M	S
ARIMA	x		x		
VAR	x			x	
LASSO	x				x

Mean Absolute Scaled Error (MASE) :



Reference:

Salotti, Fenet, Billot, El Faouzi, Solnon (2018): *Comparison of traffic forecasting methods in urban and suburban context*

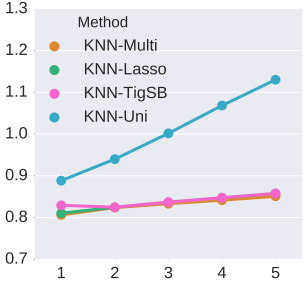
Predictive models: short term predictions (less than 1 hour)

Compared approaches:

- Univariate (U) vs Multivariate (M)
- Variable selection (S)
- Linear (L) vs Non Linear (NL)

Method	L	NL	U	M	S
ARIMA	x		x		
VAR	x			x	
LASSO	x				x
*k*NN-uni		x	x		
*k*NN-multi		x		x	
*k*NN-Lasso		x			x
*k*NN-TigSB		x			x

Mean Absolute Scaled Error (MASE) :



Reference:

Salotti, Fenet, Billot, El Faouzi, Solnon (2018): *Comparison of traffic forecasting methods in urban and suburban context*

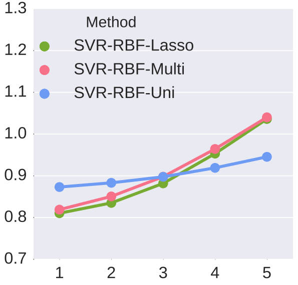
Predictive models: short term predictions (less than 1 hour)

Compared approaches:

- Univariate (U) vs Multivariate (M)
- Variable selection (S)
- Linear (L) vs Non Linear (NL)

Method	L	NL	U	M	S
ARIMA	x		x		
VAR	x			x	
LASSO	x				x
*k*NN-uni		x	x		
*k*NN-multi		x		x	
*k*NN-Lasso		x			x
*k*NN-TigSB		x			x
SVR-RBF-uni		x	x		
SVR-RBF-multi		x		x	
SVR-RBF-Lasso		x			x

Mean Absolute Scaled Error (MASE) :



Reference:

Salotti, Fenet, Billot, El Faouzi, Solnon (2018): *Comparison of traffic forecasting methods in urban and suburban context*

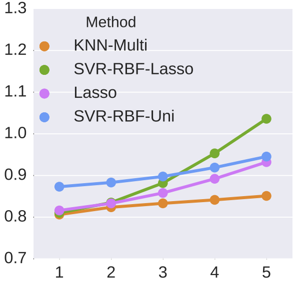
Predictive models: short term predictions (less than 1 hour)

Compared approaches:

- Univariate (U) vs Multivariate (M)
- Variable selection (S)
- Linear (L) vs Non Linear (NL)

Method	L	NL	U	M	S
ARIMA	x		x		
VAR	x			x	
LASSO	x				x
*k*NN-uni		x	x		
*k*NN-multi		x		x	
*k*NN-Lasso		x			x
*k*NN-TigSB		x			x
SVR-RBF-uni		x	x		
SVR-RBF-multi		x		x	
SVR-RBF-Lasso		x			x

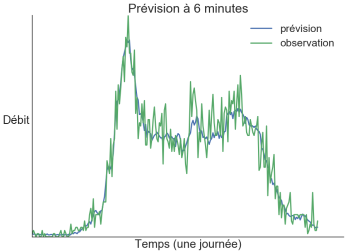
Mean Absolute Scaled Error (MASE) :

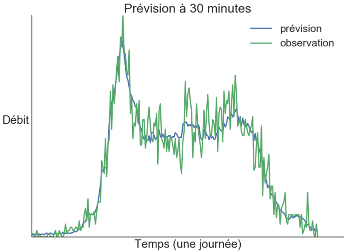


Reference:

Salotti, Fenet, Billot, El Faouzi, Solnon (2018): *Comparison of traffic forecasting methods in urban and suburban context*

Examples of forecasting (with kNN-multi):





Predictive models: long term predictions

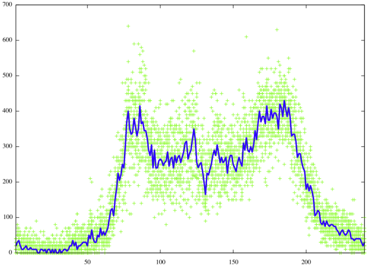
Clustering of days for each sensor:

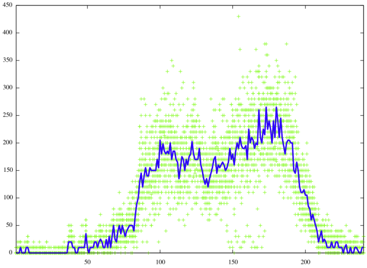
- Group days with similar time series
- May be done by exploiting knowledge or automatically

Build a representative time series for each cluster:

For each time step, search for a representative value
$\rightsquigarrow$ For example, the median

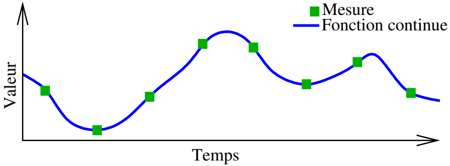
Example: Median flow (blue) over 20 days (green) for two sensors





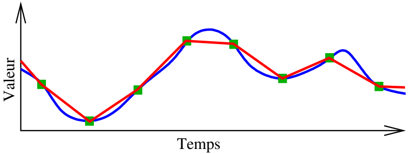
Representation of predictive models for temporal series

Temporal serie = one measure per time step (*e.g.*, 6 minutes):

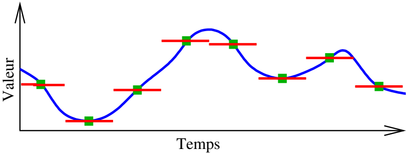


⇝ Models = one prevision per time step

Representation with piecewise linear functions:



Representation with piecewise constant functions:



How to exploit these predictions to optimise delivery tours?

When preparing the tour (the day before):

- Minimize tour durations by exploiting speed predictions
- Take into account the fact that speed is not constant through the day

$\rightsquigarrow$ Optimisation with **Time-Dependent (TD) Data**

While performing the delivery tour:

- Adapt the tour when observed events are different from predicted ones (unexpected events)
- Anticipate on likely events by exploiting statistics on past events

$\rightsquigarrow$ Optimisation with **Uncertain Data**

Context of this work

PhD thesis of Penélope Aguiar Melgarejo (defended in 2016):

- Funded by IBM and co-supervised with Philippe Laborie (IBM)
- Algorithms for optimizing with Time-Dependent Data
  - Computation of the duration of a path
  - Computation of shortest paths
  - Solve the TD-ATSP with Constraint Programming (CP)

Post-doc of Omar Rifki (2018/2019):

- Funded by IMU and co-supervised with Nicolas Chiabaut (LICIT)
- Experimental evaluation on realistic datasets
  $\rightsquigarrow$ Is it worth exploiting Time Dependent Data?

PhD thesis of romain Fontaine (started in 2020):

- Funded by “enjeu transport” and co-supervised with Jilles S. Dibangoye
- Algorithms for solving Time-Dependent VRPs

Adv.: New PhD starting in 2023 $\rightsquigarrow$ Tell me if you are interested!

Definition of the problem

Input Data:

- A set S of delivery points and a warehouse v_0
- **A start time t_0**
- For each road segment (i, j) **and each time t**:
  $d(i, j, t)$ = travel time from i to j **when leaving i at time t**

Output:

A sequence of road segments that:

- Starts from v_0, visits each point of S, and returns to v_0
- Minimises the arrival time **when leaving v_0 at t_0**

Solution process in two steps:

1. Compute quickest paths for each possible start time
   $\rightsquigarrow$ TD cost function for each couple of delivery points
2. Solve the TD-ATSP

Computation of quickest paths with TD data

Extension of Dijkstra to TD data:

Function *Dijkstra-TD*(g, d, t_0, s_0)
    for *each vertex* s_i **do** $h[s_i] \leftarrow +\infty$;
    $h[s_0] \leftarrow t_0$; Put all vertices of g in a priority queue F
    while F *is not empty* **do**
        remove from F the vertex s_i s.t. $h[s_i]$ is minimal
        for *each vertex* $s_j \in succ(s_i)$ **do**
            if $h[s_i] + d(s_i, s_j, h[s_i]) < h[s_j]$ **then**
                $h[s_j] \leftarrow h[s_i] + d(s_i, s_j, h[s_i])$
                update F
    return h

Computation of quickest paths with TD data

Extension of Dijkstra to TD data:

Function *Dijkstra-TD*(g, d, t_0, s_0)

  for *each vertex* s_i **do** $h[s_i] \leftarrow +\infty$;

  $h[s_0] \leftarrow t_0$; Put all vertices of g in a priority queue F

  while F *is not empty* **do**

    remove from F the vertex s_i s.t. $h[s_i]$ is minimal

    for *each vertex* $s_j \in succ(s_i)$ **do**

      if $h[s_i] + d(s_i, s_j, h[s_i]) < h[s_j]$ **then**

        $h[s_j] \leftarrow h[s_i] + d(s_i, s_j, h[s_i])$

        update F

  return h

Condition for Dijkstra to be correct?

Computation of quickest paths with TD data

Extension of Dijkstra to TD data:

Function *Dijkstra-TD*(g, d, t_0, s_0)
  for *each vertex* s_i **do** $h[s_i] \leftarrow +\infty$;
  $h[s_0] \leftarrow t_0$; Put all vertices of g in a priority queue F
  while F *is not empty* **do**
    remove from F the vertex s_i s.t. $h[s_i]$ is minimal
    for *each vertex* $s_j \in succ(s_i)$ **do**
      if $h[s_i] + d(s_i, s_j, h[s_i]) < h[s_j]$ **then**
        $h[s_j] \leftarrow h[s_i] + d(s_i, s_j, h[s_i])$
        update F
  return h

Condition for Dijkstra to be correct?

- Every subpath of an optimal path must be optimal

Is the condition satisfied when costs are time-dependent?

Computation of quickest paths with TD data

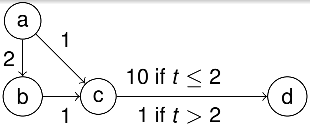
Extension of Dijkstra to TD data:

Function *Dijkstra-TD*(g, d, t_0, s_0)
    for *each vertex* s_i **do** $h[s_i] \leftarrow +\infty$;
    $h[s_0] \leftarrow t_0$; Put all vertices of g in a priority queue F
    while *F is not empty* **do**
        remove from F the vertex s_i s.t. $h[s_i]$ is minimal
        for *each vertex* $s_j \in succ(s_i)$ **do**
            if $h[s_i] + d(s_i, s_j, h[s_i]) < h[s_j]$ **then**
                $h[s_j] \leftarrow h[s_i] + d(s_i, s_j, h[s_i])$
                update F
    return h

Condition for Dijkstra to be correct?

- Every subpath of an optimal path must be optimal

Is the condition satisfied when costs are time-dependent?



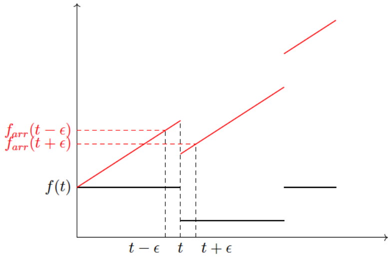
When leaving a at 0: cost of $\langle a, b, c, d \rangle = 4$
cost of $\langle a, c, d \rangle = 11$

Computation of quickest paths with TD data

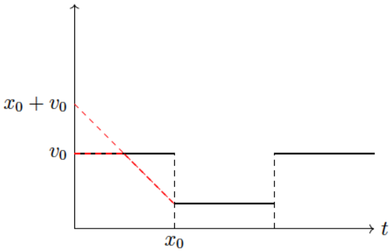
To ensure the correctness of Dijkstra, d must satisfy the FIFO property:

- If $t_1 < t_2$ then $t_1 + d(i, j, t_1) \leq t_2 + d(i, j, t_2)$
  $\rightsquigarrow$ We cannot arrive sooner if we leave later
- If d is not FIFO, then searching for shortest paths is $\mathcal{NP}$-hard

Example of non FIFO function:



How to make it FIFO?



References:

- Kaufman, Smith (1993): Fastest paths in time-dependent networks for intelligent vehicle-highway systems
- Ichoua, Gendreau, Potvin (2003): Vehicle dispatching with time-dependent travel times

Non Exhaustive Literature Review on the TD-TSP

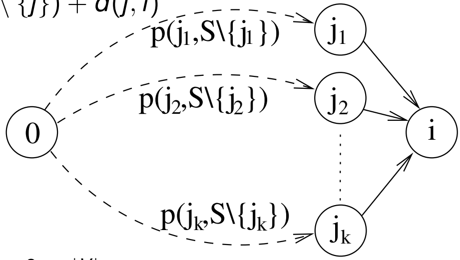
	Problem		Constraints		Solving approach	
Authors (Year)	TSP	VRP	TW	Q	Exact	Heuristic
Malandraki & Daskin (1992)	✓	✓	✓	✓	ILP	
Malandraki & Dial (1996)	✓				DP	RDP
Schneider (2002)	✓					LS
Ichoua et al (2003)		✓	✓			LS
Fleischmann et al (2004)		✓	✓			Greedy
Haghani & Jung (2005)		✓	✓	✓		GA
Eglese et al (2006)		✓	✓	✓		LS
Van Woensel et al (2008)		✓		✓		LS
Donati et al (2008)		✓	✓	✓		ACO
Ehmke et al (2012)	✓	✓				Greedy
Kanoh & Ochiai (2012)	✓					ACO
Figliozzi (2012)		✓	✓	✓		Greedy
Cordeau et al (2014)	✓				ILP	
Melgarejo et al (2015)	✓				CP	
Montero et al (2017)	✓		✓		ILP	
Vu et al (2018)	✓		✓		ILP	
Arigliano et al (2019)	✓		✓		ILP	
Sun et al (2020)		✓	✓	✓		LS
Vu et al (2020)	✓		✓		ILP	
Rifki et al (2020)	✓	✓	✓	✓	DP	
Fontaine et al (2022)	✓		✓		DP	

Dynamic Programming (DP) for the TSP

Bellman equations for a set V of points (with warehouse=0):

$\forall i \in V, \forall S \subseteq V \setminus \{0\}$: let $p(i, S)$ denote the length of the shortest path from 0 to i that visits each point of S exactly once

- If $S = \emptyset$, then $p(i, S) = d(0, i)$
- Otherwise $p(i, S) = \min_{j \in S} p(j, S \setminus \{j\}) + d(j, i)$

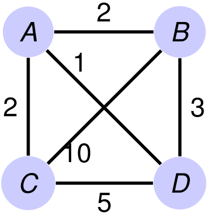


$\rightsquigarrow$ Computation of $p(0, V \setminus \{0\})$ in $\mathcal{O}(|V|^2 \cdot 2^{|V|})$

Reference:

Held, Karp (1962): *A dynamic programming approach to sequencing problems*

Illustration on a small example



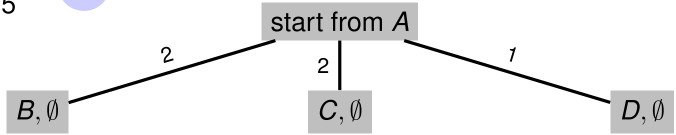


Illustration on a small example

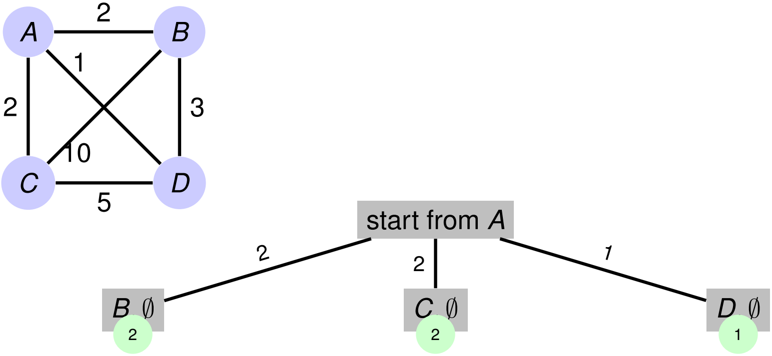


Illustration on a small example

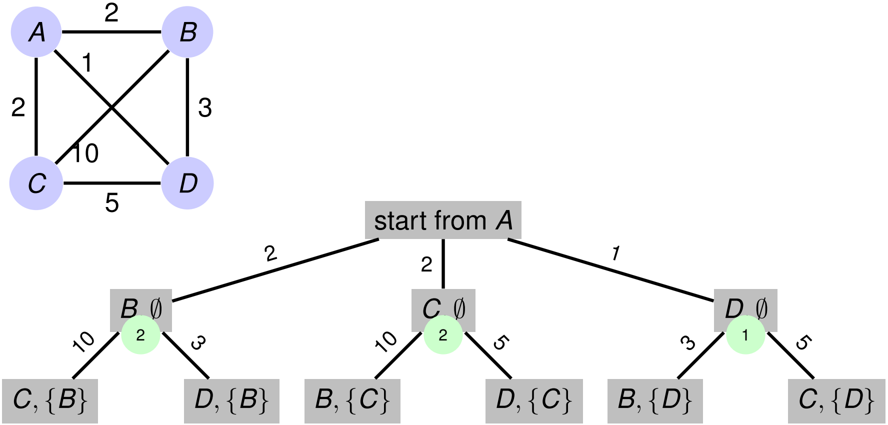


Illustration on a small example

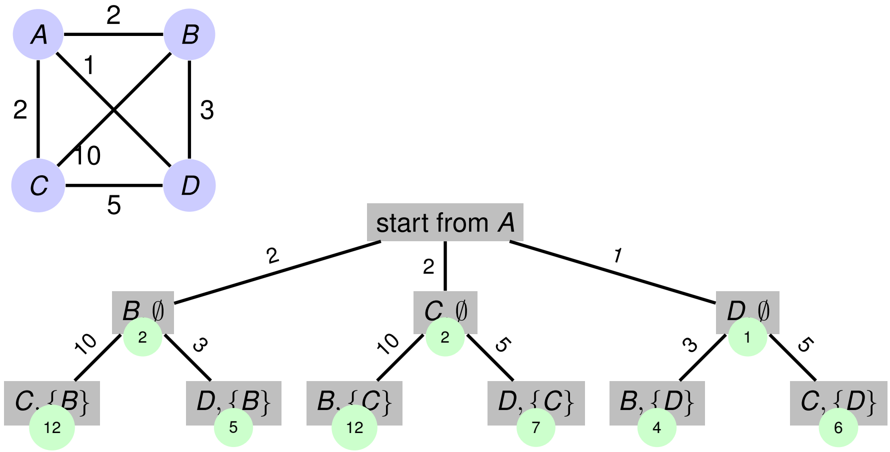


Illustration on a small example

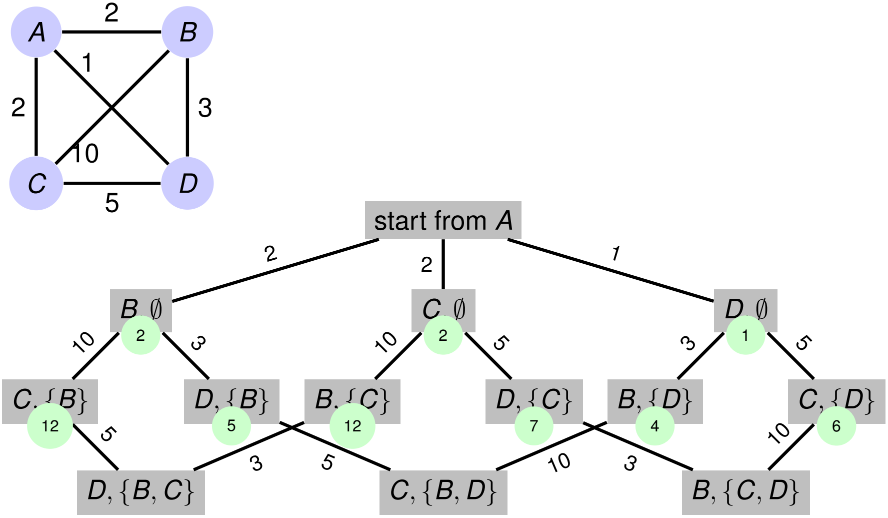


Illustration on a small example

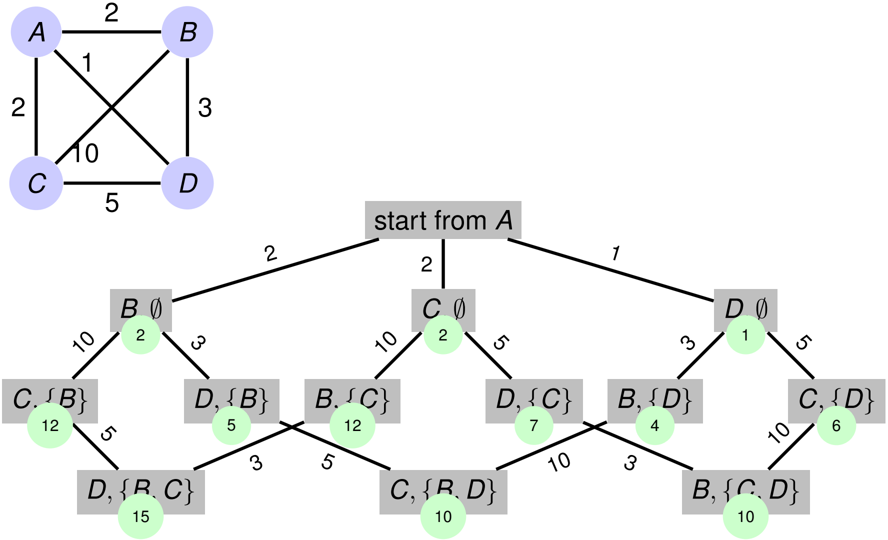


Illustration on a small example

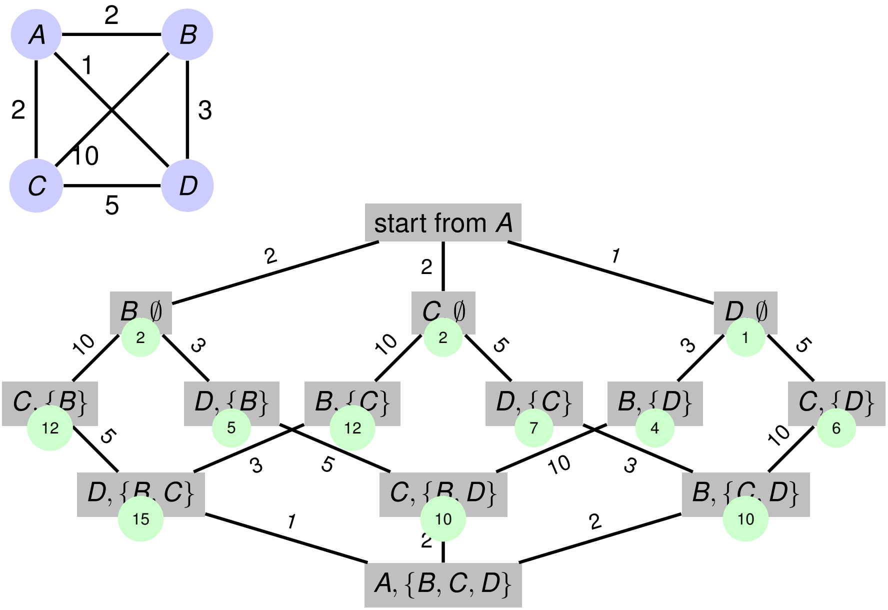
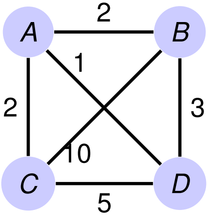
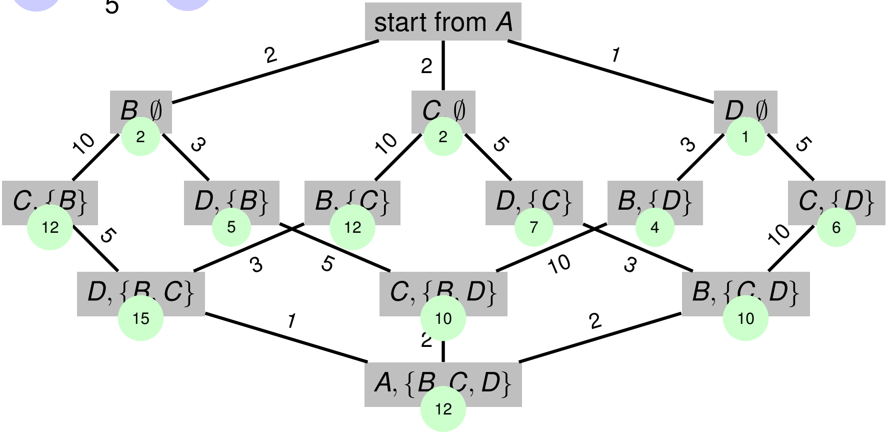


Illustration on a small example





DP for the TSP

Bellman equations for a set V of points (with warehouse=0):

$\forall i \in V, \forall S \subseteq V \setminus \{0\}$: let $p(i, S)$ denote the length of the shortest path from 0 to i that visits each point of S exactly once

- If $S = \emptyset$, then $p(i, S) = d(0, i)$
- Otherwise $p(i, S) = \min_{j \in S} p(j, S \setminus \{j\}) + d(j, i)$

Anytime Column Search (ACS):

Iterated Depth-First Search in the state-space
$\rightsquigarrow$ Anytime and exact approach

DP for the TD-TSP

Bellman equations for a set V of points (with warehouse=0):

$\forall i \in V, \forall S \subseteq V \setminus \{0\}$: let $p(i,S)$ denote the length of the shortest path from 0 to i that visits each point of S exactly once

- If $S = \emptyset$, then $p(i,S) = d(0,i,t_0)$
- Otherwise $p(i,S) = \min_{j \in S} p(j, S \setminus \{j\}) + d(j, i, p(j, S \setminus \{j\}))$

Anytime Column Search (ACS):

Iterated Depth-First Search in the state-space
$\rightsquigarrow$ Anytime and exact approach

References:

- Malandraki, Dial (1996): *A restricted dynamic programming heuristic algorithm for the TD-TSP*
- Vadlamudi, Gaurav, Aine, Chakrabarti (2012): *Anytime Column Search*

DP for the TD-TSP

Bellman equations for a set V of points (with warehouse=0):

$\forall i \in V, \forall S \subseteq V \setminus \{0\}$: let $p(i, S)$ denote the length of the shortest path from 0 to i that visits each point of S exactly once

- If $S = \emptyset$, then $p(i, S) = d(0, i, t_0)$
- Otherwise $p(i, S) = \min_{j \in S} p(j, S \setminus \{j\}) + d(j, i)$

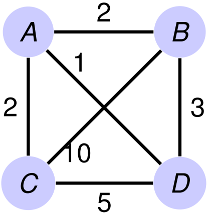
Anytime Column Search (ACS):

Iterated Depth-First Search in the state-space
$\rightsquigarrow$ Anytime and exact approach

References:

- Malandraki, Dial (1996): *A restricted dynamic programming heuristic algorithm for the TD-TSP*
- Vadlamudi, Gaurav, Aine, Chakrabarti (2012): *Anytime Column Search*

Illustration on a small example



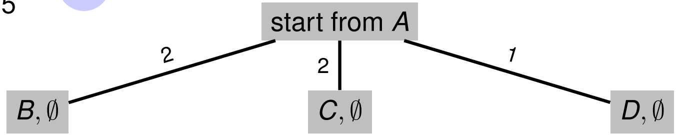
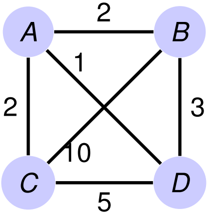


Illustration on a small example



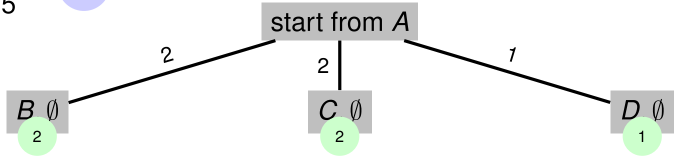
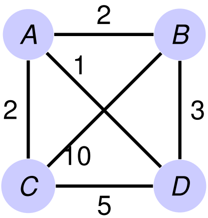


Illustration on a small example



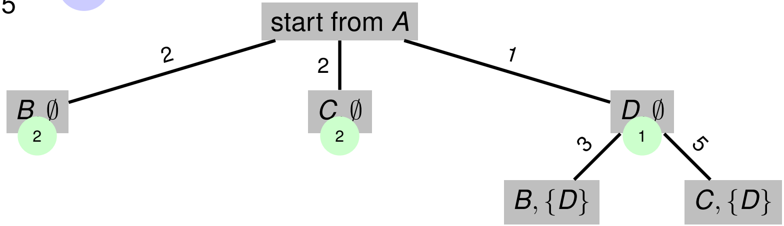
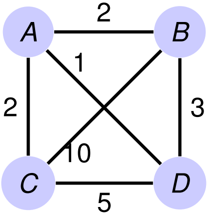


Illustration on a small example



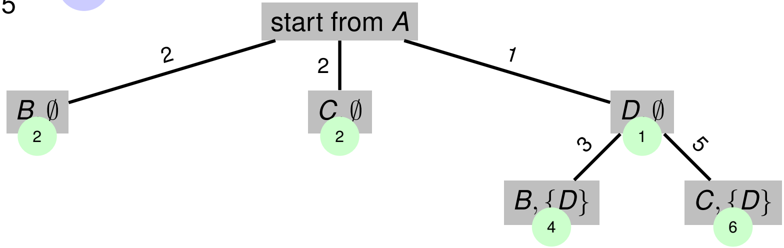


Illustration on a small example

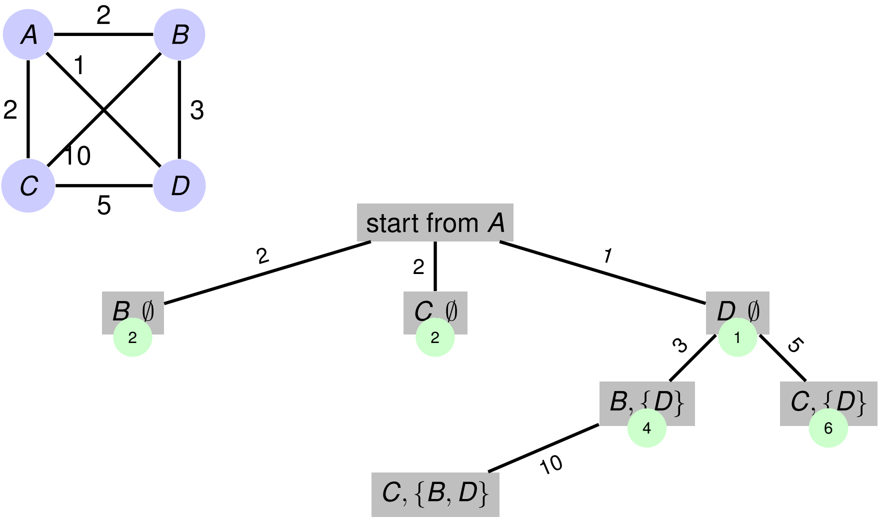


Illustration on a small example

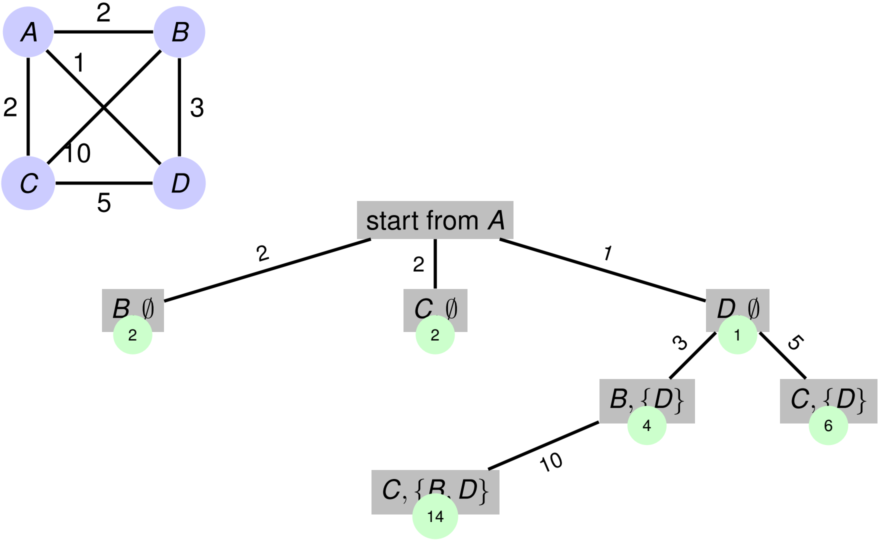


Illustration on a small example

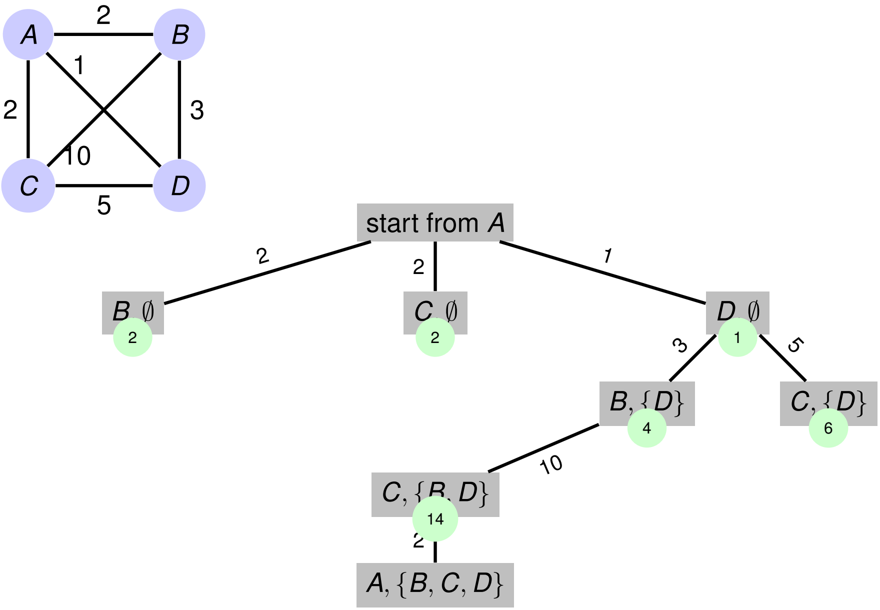
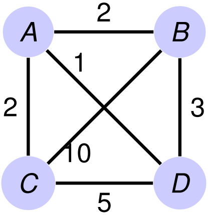


Illustration on a small example



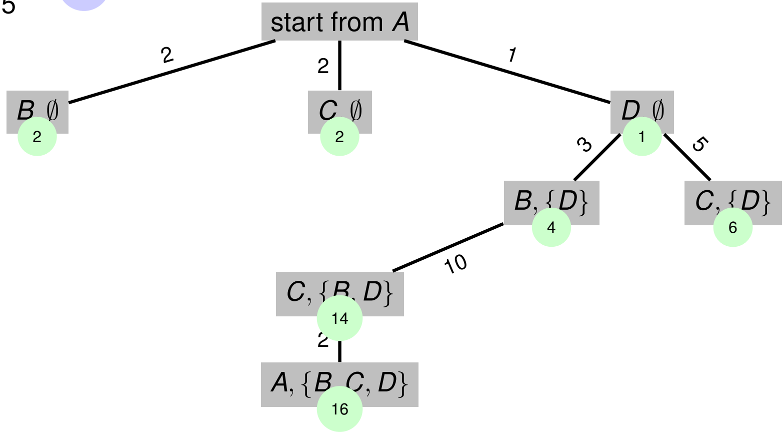
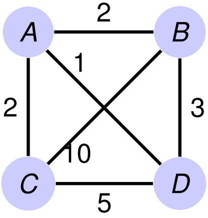


Illustration on a small example



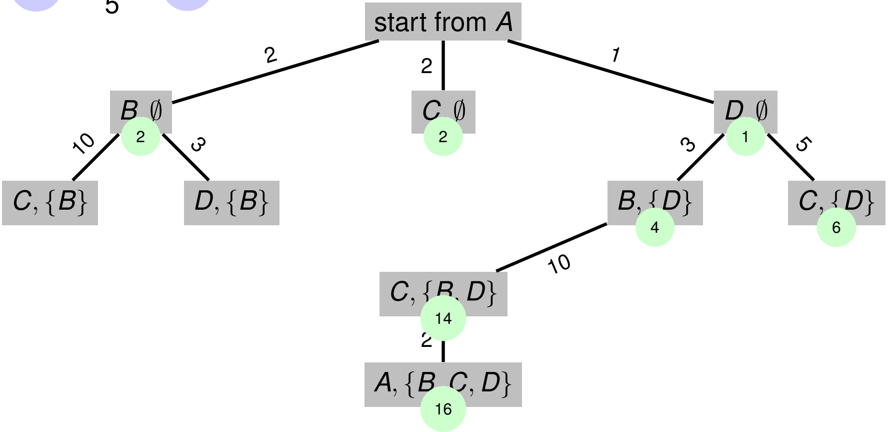


Illustration on a small example

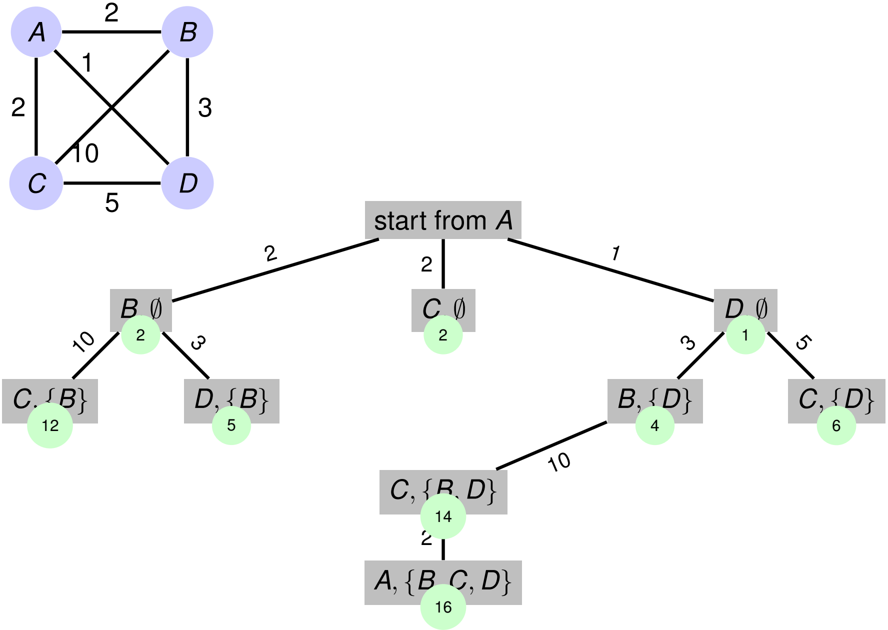


Illustration on a small example

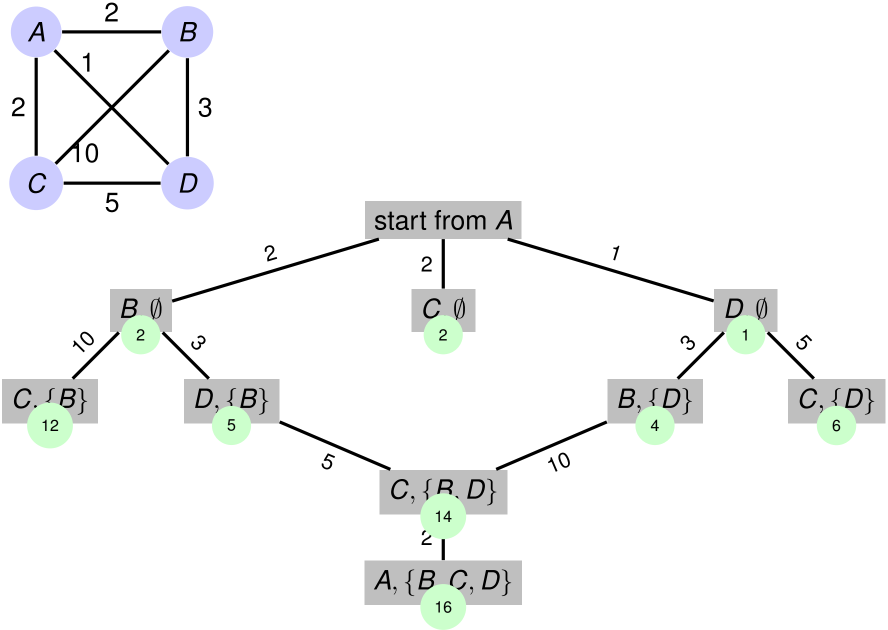


Illustration on a small example

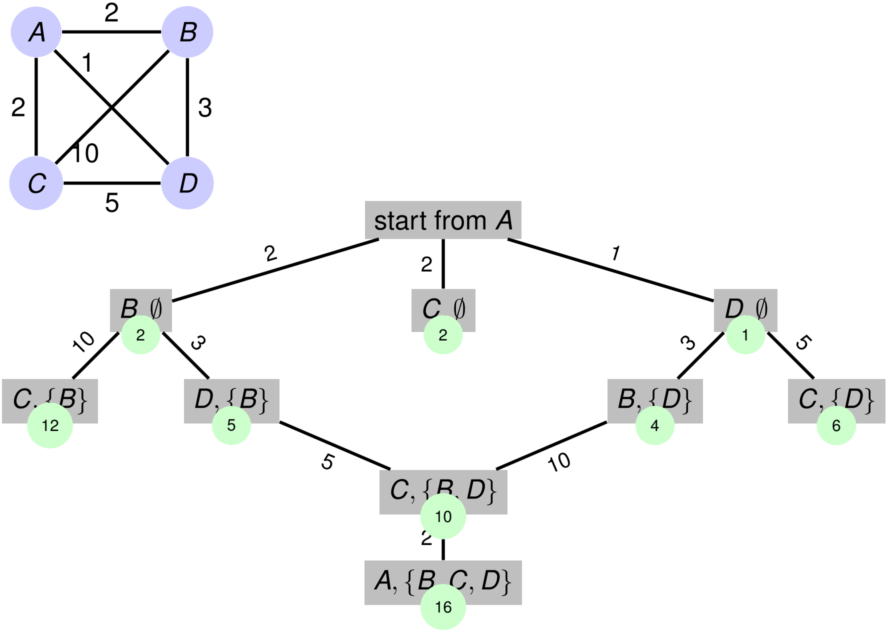


Illustration on a small example

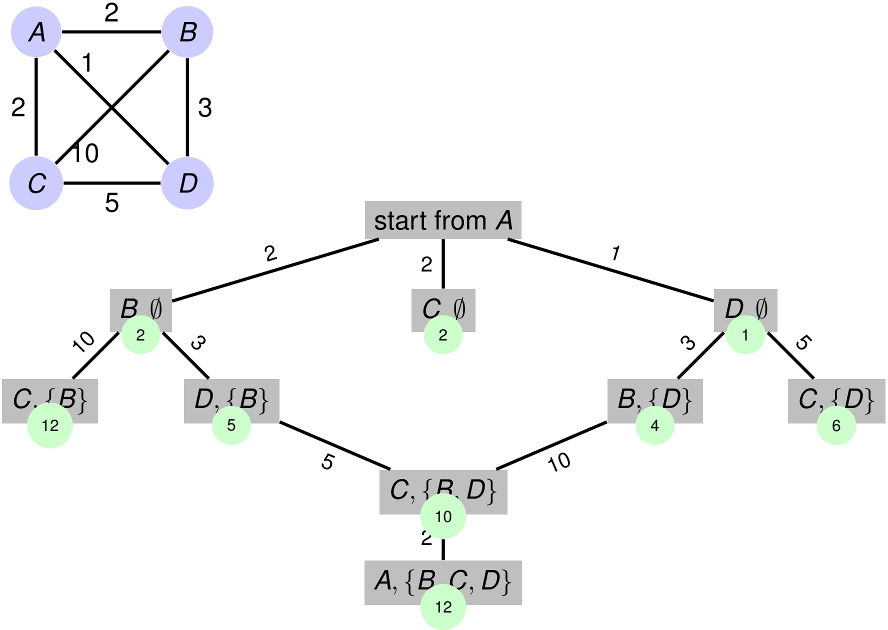


Illustration on a small example

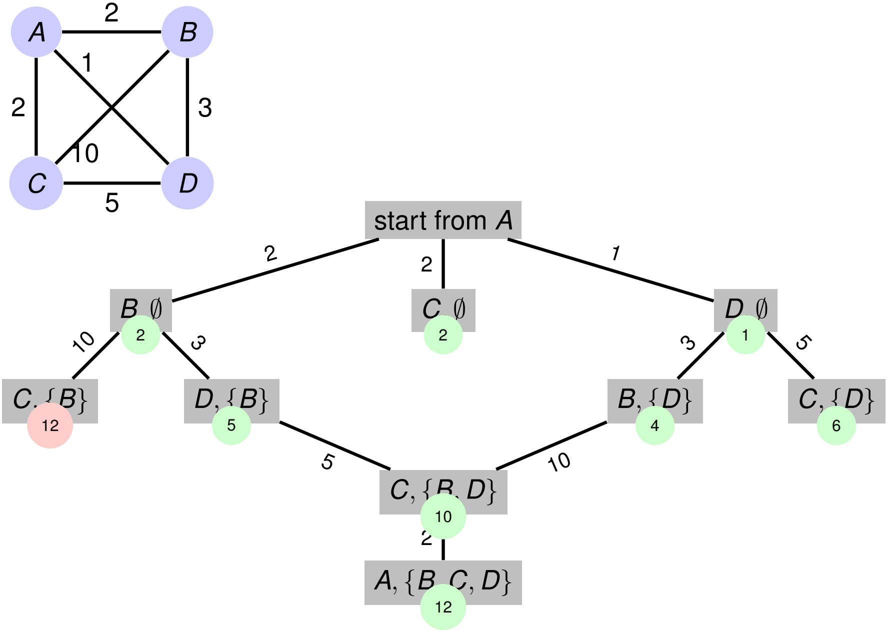


Illustration on a small example

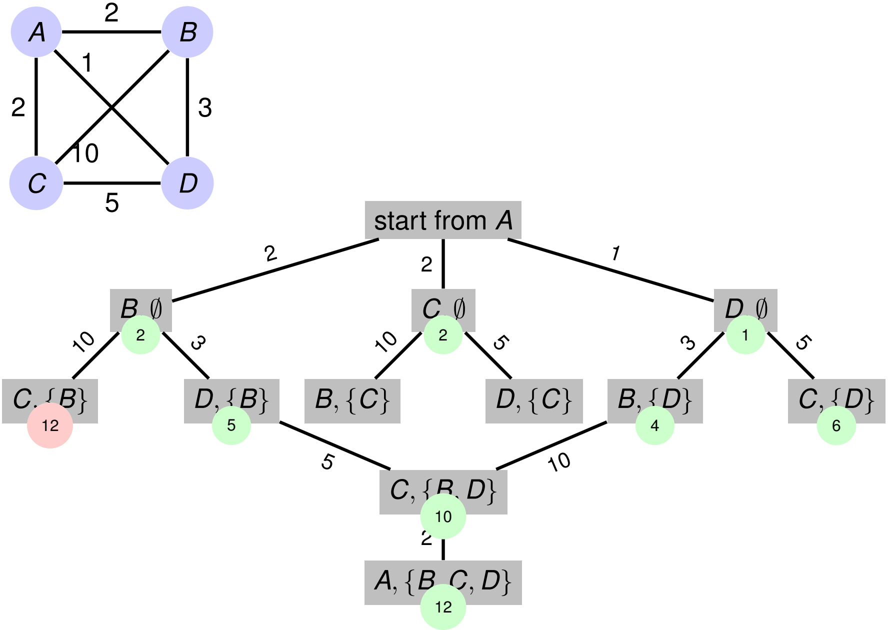


Illustration on a small example

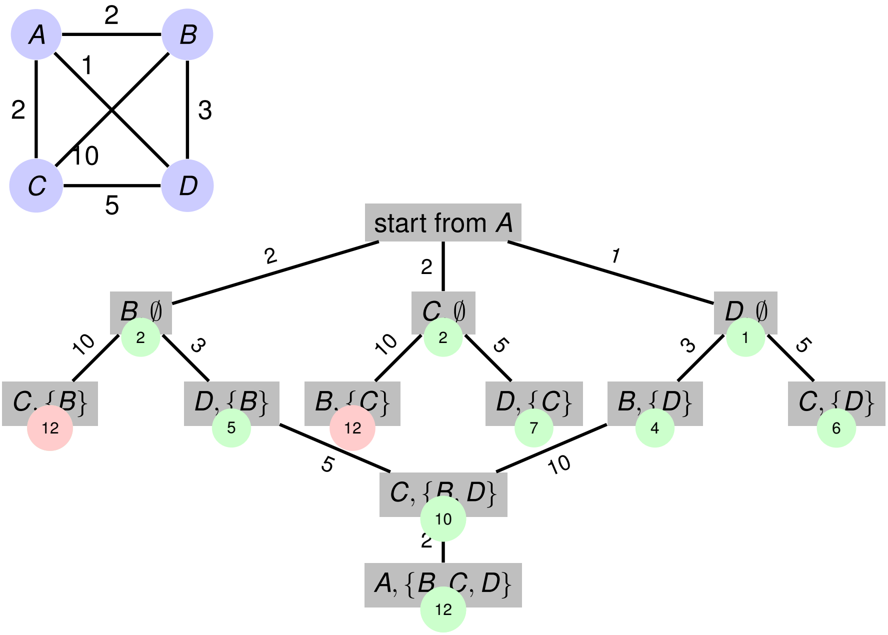


Illustration on a small example

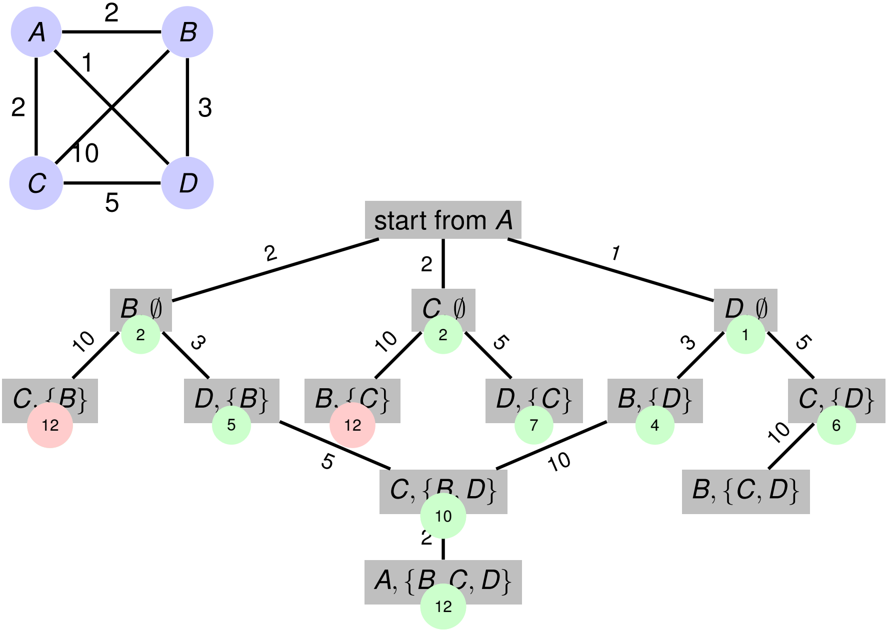


Illustration on a small example

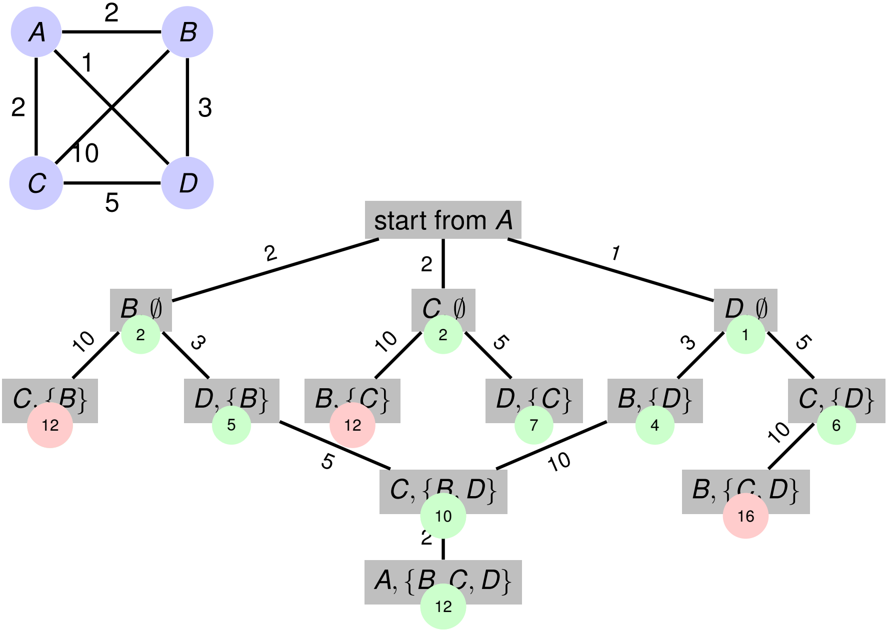
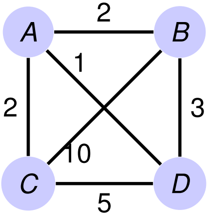


Illustration on a small example



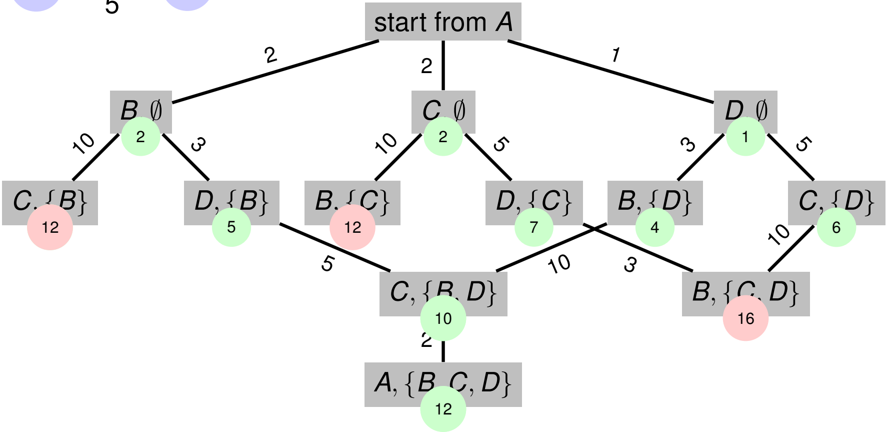


Illustration on a small example

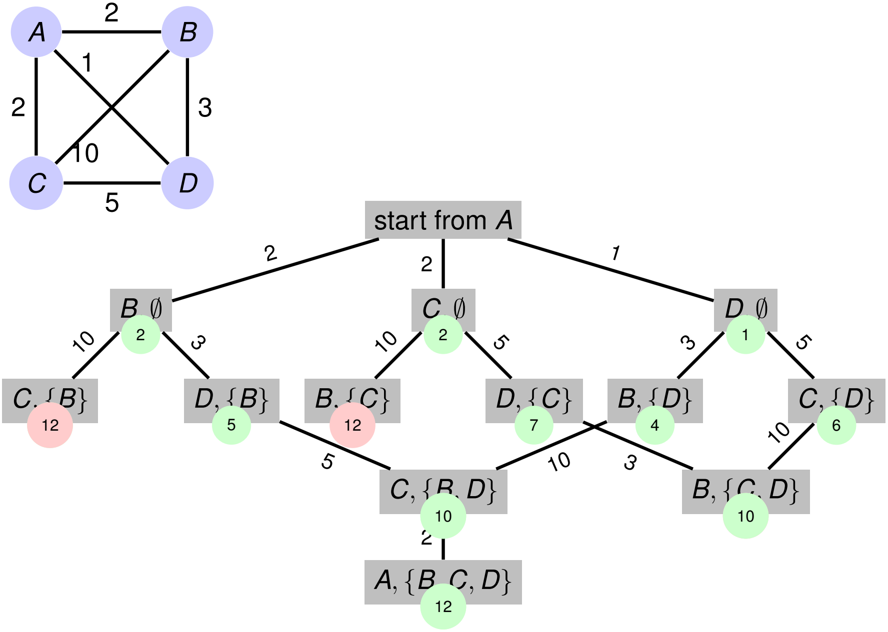
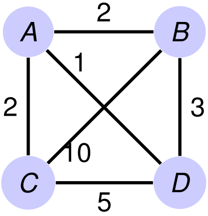
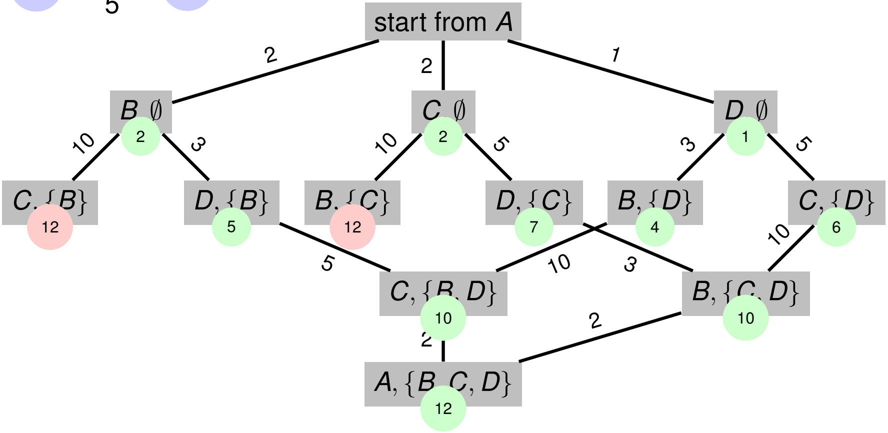


Illustration on a small example





Anytime and Exact DP-based approach for the TD-TSPTW (FON22)

Combine ACS with:

- Local search to converge faster towards better solutions
- Bounding and time window propagation to prune the state space

Reference:

Fontaine, Dibangoye, Solnon (2022): *Exact and Anytime approach for solving the TD-TSP-TW*

Recent ILP approaches for the TD-TSP with Time Windows

Exploitation of common congestion patterns (ARI19):

- Computation of bounds by considering suitable-defined constant costs
  $\rightsquigarrow$ the more arcs share a same congestion pattern, the tighter the bound
- State-of-the-art results on instances with common congestion patterns

Dynamic Discretization Discovery (VU20):

- Dynamic time step refinement to strengthen time-indexed ILP models
- State-of-the-art results on instances with very tight time windows

References:

- Arigliano, Ghiani, Grieco, Guerriero, Plana (2019): *Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm*
- Vu, Hewitt, Boland, Savelsbergh (2020): *Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows*

Comparison of ARI19 (ILP) with FON22 (DP)

Randomly generated instances of [Arigliano et al 2019] with $n = 31$:

- Δ is used to control congestion pattern similarity
  $\rightsquigarrow$ The closer Δ to 1, the more common congestion patterns
- β is used to control time window tightness
  $\rightsquigarrow$ The closer β to 1, the tighter the time window

Percentage of solved instances within 1h:

		Pattern B_1					Pattern B_2				
		Δ=.70	Δ=.80	Δ=.90	Δ=.95	Δ=.98	Δ=.70	Δ=.80	Δ=.90	Δ=.95	Δ=.98
ARI19	β=0	23	43	67	93	100	0	7	23	60	100
	β=.25	33	53	90	100	100	7	30	63	90	97
	β=.5	17	23	70	87	97	7	13	47	67	97
	β=1	80	73	83	87	100	73	60	67	77	77

- ARI19 is sensitive to β and to Δ
- FON22 is sensitive to β, but not to Δ

Comparison of ARI19 (ILP) with FON22 (DP)

Randomly generated instances of [Arigliano et al 2019] with $n = 31$:

- Δ is used to control congestion pattern similarity
  $\rightsquigarrow$ The closer Δ to 1, the more common congestion patterns
- β is used to control time window tightness
  $\rightsquigarrow$ The closer β to 1, the tighter the time window

Percentage of solved instances within 1h (on different computers...):

		Pattern B_1					Pattern B_2				
		Δ=.70	Δ=.80	Δ=.90	Δ=.95	Δ=.98	Δ=.70	Δ=.80	Δ=.90	Δ=.95	Δ=.98
ARI19	β=0	23	43	67	93	100	0	7	23	60	100
	β=.25	33	53	90	100	100	7	30	63	90	97
	β=.5	17	23	70	87	97	7	13	47	67	97
	β=1	80	73	83	87	100	73	60	67	77	77
FON22	β=0	67	67	67	67	67	83	80	70	70	70
	β=.25	100	100	100	100	100	100	100	100	100	100
	β=.5	100	100	100	100	100	100	100	100	100	100
	β=1	100	100	100	100	100	100	100	100	100	100

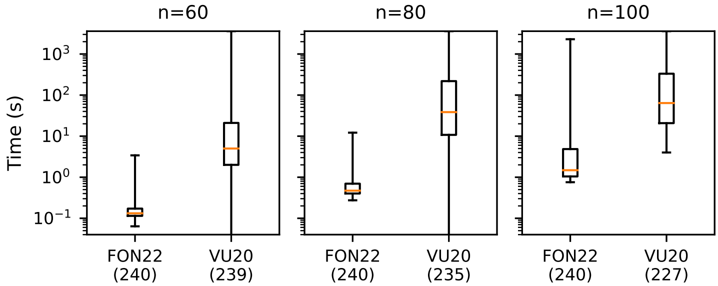
- ARI19 is sensitive to β and to Δ
- FON22 is sensitive to β, but not to Δ

Comparison of VU20 (ILP) with FON22 (DP)

Randomly generated instances of [Vu et al 2020]

- Same model as [Arigliano et al 2019]
- Instances with very tight time windows only ($\beta = 1$)
- $n \in \{60, 80, 100\}$ (240 instances per value of n)

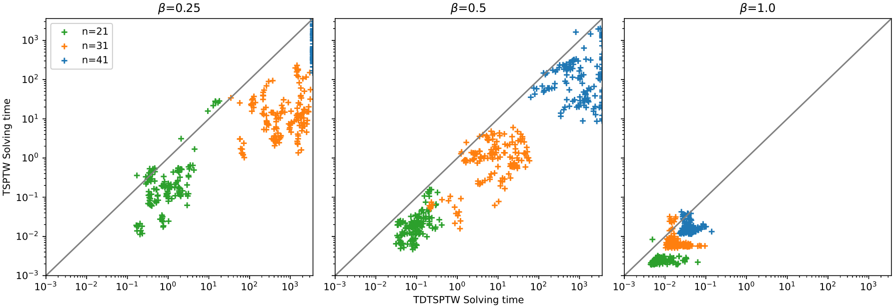
Solving times (on different computers...):



- FON22 solves all instances within 1h
- VU20 fails on 19 instances and is more than 1 order slower

Is the TD-TSP-TW harder than the TSP-TW?

Solving times of FON22 with TD costs (x) and constant costs (y):



⇝ Yes, much harder for most instances!

Is it worth spending much more time?

To answer this question, we must consider realistic TD cost functions!

Construction of a realistic benchmark

Utilisation of a simulator (SymuVia, LICIT) of the Lyon road network:

- Different levels of sensor coverage:
  - D_{Lyon}: real sensor positions (cover=7%)
  - D_σ with $\sigma \in \{10, 20, ..., 100\}$: cover=$\sigma$% (evenly distributed)

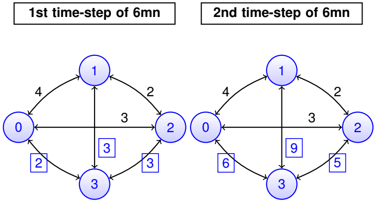
  $\rightsquigarrow$ Values for uncovered road segments are interpolated
- Different time-step length $l \in \{6, 12, 24, 60, 720\}$ ($l = 720 \Leftrightarrow$ static case)

Reference:

Rifki, Chiabaut, Solnon: *On the impact of spatio-temporal granularity of traffic conditions on the quality of pickup and delivery optimal tours*

How to compare tours optimised on different Data?

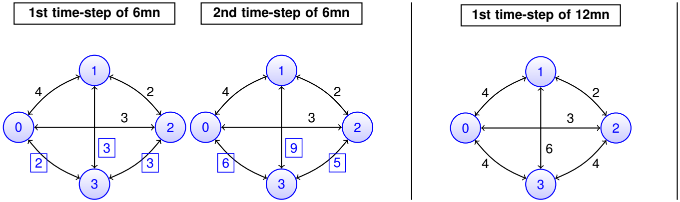
⇝ Illustration on an artificial example



- Best : $T^6 = \langle 0, 3, 1, 2, 0 \rangle$
- Arrival time = 10
- Realistic travel time $rtt = 10$

How to compare tours optimised on different Data?

⇝ Illustration on an artificial example

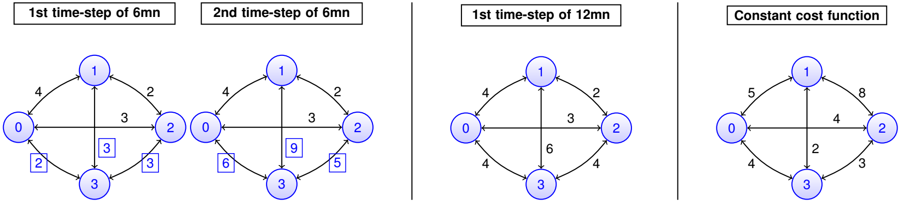


- Best : $T^6 = \langle 0, 3, 1, 2, 0 \rangle$
- Arrival time = 10
- Realistic travel time $rtt = 10$

- $T^{12} = \langle 0, 1, 2, 3, 0 \rangle$
- Arrival time = 14
- $rtt = 17$

How to compare tours optimised on different Data?

⇝ Illustration on an artificial example



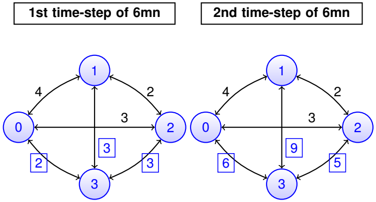
- Best : $T^6 = \langle 0, 3, 1, 2, 0 \rangle$
- Arrival time = 10
- Realistic travel time $rtt = 10$

- $T^{12} = \langle 0, 1, 2, 3, 0 \rangle$
- Arrival time = 14
- $rtt = 17$

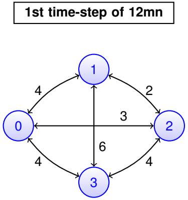
- $T^{720} = \langle 0, 2, 3, 1, 0 \rangle$
- Arrival time = 14
- $rtt = 19$

How to compare tours optimised on different Data?

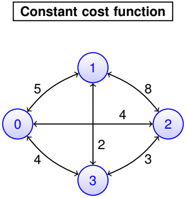
⇝ Illustration on an artificial example



- Best : $T^6 = \langle 0, 3, 1, 2, 0 \rangle$
- Arrival time = 10
- Realistic travel time $rtt = 10$



- $T^{12} = \langle 0, 1, 2, 3, 0 \rangle$
- Arrival time = 14
- $rtt = 17$



- $T^{720} = \langle 0, 2, 3, 1, 0 \rangle$
- Arrival time = 14
- $rtt = 19$

Evaluate all tours with a same cost function

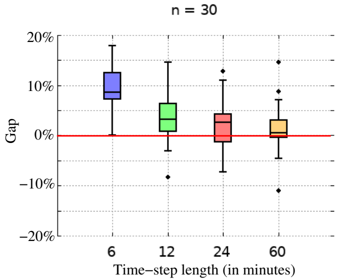
Use the cost function which is the closest to real conditions
⇝ **Realistic travel time (rtt)** computed with $l = 6$mn and $\sigma = 100\%$

Question 1: Can we find better tours when using TD Data?

Performance measure:

Gap between T^{720} and T^{l} with $l \in \{6, 12, 24, 60\}$ = $\frac{rtt(T^{720}) - rtt(T^{l})}{rtt(T^{l})} \times 100$

Results when $\sigma = 100\%$:



Answer to Question 1:

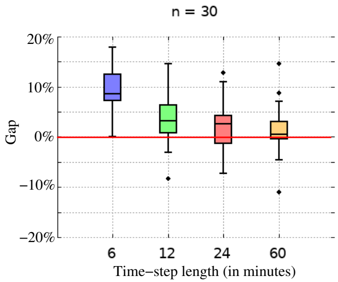
- When $\sigma = 100\%$: Yes, and the smaller the time step the larger the gain
- When $\sigma =$ Lyon: No

Question 1: Can we find better tours when using TD Data?

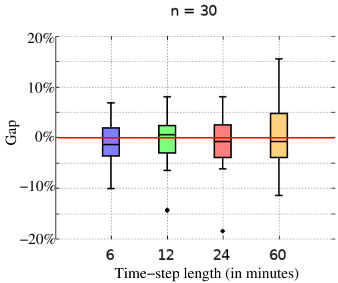
Performance measure:

Gap between T^{720} and T^{l} with $l \in \{6, 12, 24, 60\}$ = $\frac{rtt(T^{720}) - rtt(T^{l})}{rtt(T^{l})} \times 100$

Results when $\sigma = 100\%$:



Results when $\sigma =$ Lyon:



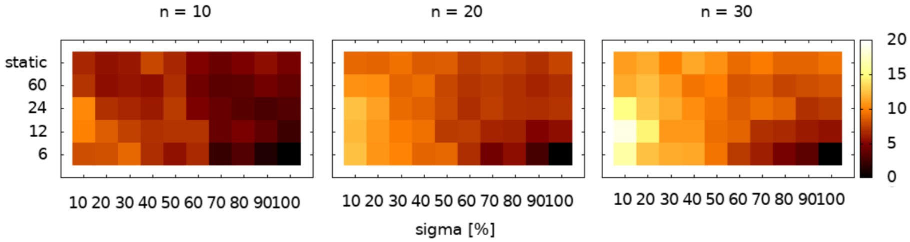
Answer to Question 1:

- When $\sigma = 100\%$: Yes, and the smaller the time step the larger the gain
- When $\sigma =$ Lyon: No

Question 2: What is the impact of l and σ on tour quality?

Performance measure:

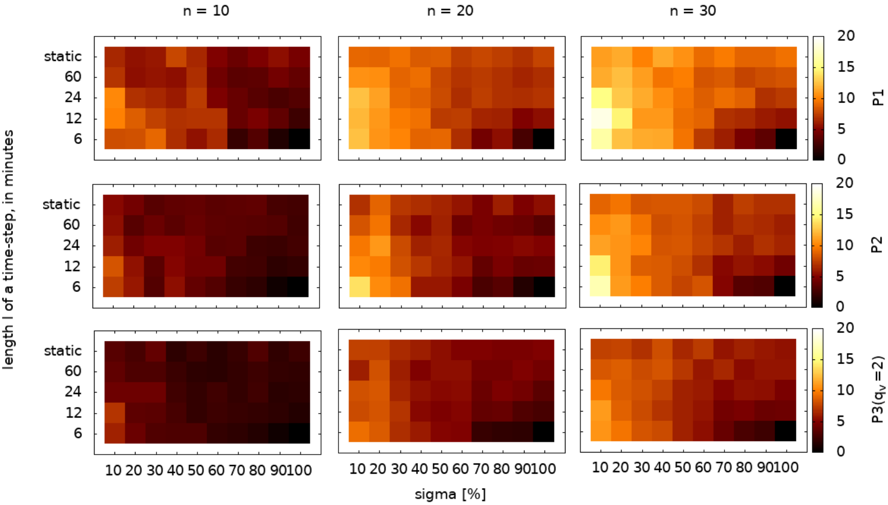
Gap between $T^{(l,\sigma)}$ and $T^{(6,100)} = \frac{rtt(T^{(l,\sigma)}) - rtt(T^{(6,100)})}{rtt(T^{(6,100)})} \times 100$



Answer to Question 2:

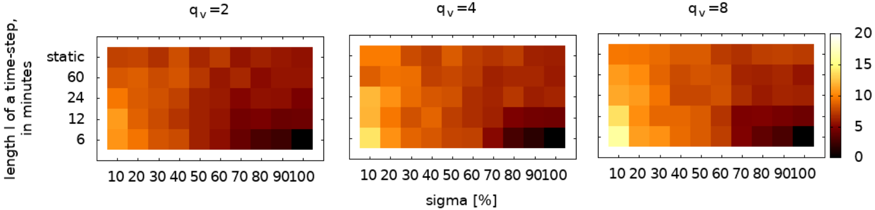
- The impact of l and σ increases when increasing n
- It is worth exploiting TD data when $\sigma = 100\%$: Tours computed with $l = 720$mn are 8% as long as those computed with $l = 6$mn when $n = 30$
- We'd better use constant data when $\sigma \leq 50\%$
  $\rightsquigarrow$ Interpolation doesn't allow to compute good approximations of speed

Question 3: Does this impact change when adding constraints?



- P1 = TD-ATSP
- P2 = TD-PDP (TD-ATSP + precedence constraints)
- P3 = TD-DARP (TD-PDP + capacity constraints with capacity $q_v = 2$)

Question 3 (continued): What if we change the capacity q_v?



Answer to question 3:

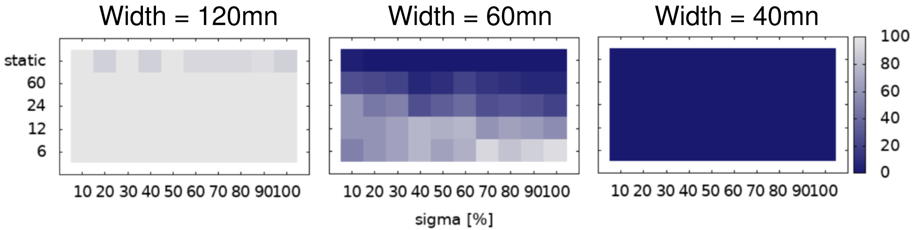
- When adding precedence or capacity constraints, the interest of exploiting TD Data decreases because these constraints decrease the number of valid tours
- The tighter the constraints, the less interesting TD Data are

Question 4: Impact of l and σ on time window satisfaction?

Performance measure:

Percentage of tours for which TW are still satisfied when evaluating them with $l = 6$ and $\sigma = 100\%$

Results for TD-TW-ATSP when $n = 40$ and the number of TW $\in \{2, 4, 6\}$:

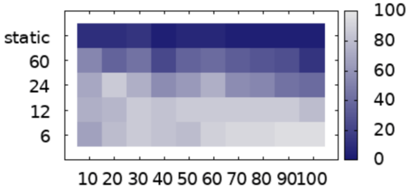


- When TWs are very large (120mn), all tours are still feasible except when using constant costs
- When TWs are very tight (40mn), all tours are infeasible
- Between these extreme cases, it is worth exploiting TD data even when $\sigma = 10\%$

What about the TD-DARP-TW?

Percentage of feasible tours:

($n = 60$, $q = 6$, and TW length $= 60$)

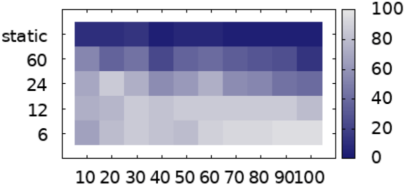


- Feasibility is decreased when increasing l, even when $\sigma < 100\%$
  $\rightsquigarrow$ Tours optimised with constant costs are nearly always infeasible
- When $l = 6$, decreasing σ decreases feasibility...
  ...But when $l \geq 12$, decreasing σ increases feasibility
- This may be explained by shortest path durations

What about the TD-DARP-TW?

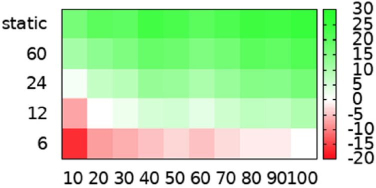
Percentage of feasible tours:

($n = 60$, $q = 6$, and TW length $= 60$)



Shortest paths durations:

(% wrt durations when $l = 6$ and $\sigma = 100$)



- Feasibility is decreased when increasing l, even when $\sigma < 100\%$
  $\rightsquigarrow$ Tours optimised with constant costs are nearly always infeasible
- When $l = 6$, decreasing σ decreases feasibility...
  ...But when $l \geq 12$, decreasing σ increases feasibility
- This may be explained by shortest path durations

Conclusion: Two main challenges for TD routing problems

Scalability:

TD problems are much more difficult than constant ones
$\rightsquigarrow$ Some instances with 40 points to visit are not solved within 1h

Reliability:

Getting reliable TD cost functions is not an easy task!

- The number of sensors has a strong impact on reliability
  $\rightsquigarrow$ What about their position?
- Interpolation is not a good predictor for missing values
  $\rightsquigarrow$ Can we find better predictors?
- So far, we have assumed that we have perfect predictive models
  $\rightsquigarrow$ Is it really the case?

Conclusion: Is it worth exploiting time-dependent data?

It depends on the goal!

- If the goal is to reduce tour durations:
  $\rightsquigarrow$ Not really if we don't have perfect TD Data
- If the goal is to better satisfy time window constraints:
  $\rightsquigarrow$ Yes, even when only 10% of the road segments have sensors

What about carbon footprint?

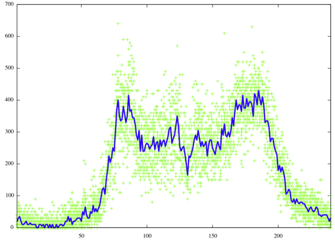
- What is the cost of getting reliable TD Data?
- Can TD problems ease shared and multi-modal mobility?
  $\rightsquigarrow$ Work with social scientists on this question!?

Prescriptive Data Analytics

Motivations

Optimisation with Data coming from predictive models:

- A prediction may be wrong
  What should we do in this case?
- Some predictions are more reliable than others
- Can we anticipate with respect to likely events?



Context of this work:

PhD thesis of Michael Saint Guillain defended in 2019 (co-tutelle with Louvain-la-neuve, Belgium, co-supervised with Yves Deville)

Classical optimisation problems

Problems are defined by means of:

- Input Data
- Decision variables (X) and their domains (D)
- Constraints to be satisfied (C)
- Objective function to optimise (F)

Solution:

Assign values to variables so that all constraints are satisfied and the objective function is optimal

What can we do when observed Data $\neq$ input Data?

- Recompute a new solution wrt new Data
- Drawbacks:
  - Re-computation is time consuming
  - The new solution may be much worse than the one computed by anticipating wrt likely events

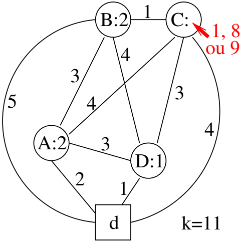
Stochastic Optimisation Problems

Stochastic problems are defined by means of:

- Input Data **known with certainty**
- **Uncertain input Data = Random variables**
- Decision variables (X) and their domains (D)
- Constraints to be satisfied (C)
- Objective function to optimise (F)

Example: VRP with stochastic demands

- Certain data: points to deliver, distances, vehicle capacity
- Uncertain data: demands
- Probability distributions of demands:
  - $p(r_A = 2) = p(r_B = 2) = p(r_D = 1) = 1$
  - $p(r_C = 1) = \frac{1}{3}, p(r_C = 8) = \frac{1}{3}, p(r_C = 9) = \frac{1}{3}$



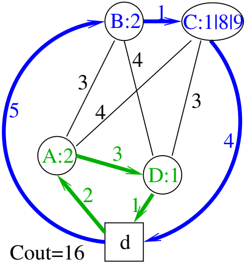
Robust solutions for stochastic optimisation problems

Ensure the feasibility of the solution wrt some given probability p:

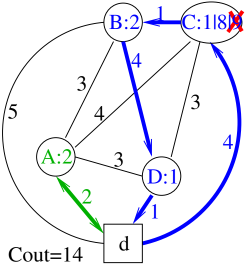
Goal = Assign values to variables so that the objective function is optimal and **the probability that constraints are satisfied is greater than p**

Example: Robust solutions for the VRP with stochastic demands

Robust solution wrt $p = 1$:



Robust solution wrt $p = \frac{2}{3}$:



Stochastic Constraint Programming [Walsh 2009, Piette 2016]

Used for General Game Playing: WoodStock winner of IGGPC 2016

Flexible solutions for stochastic optimisation problems

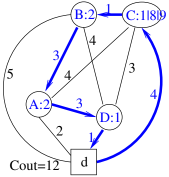
Optimise the expected cost of adapted solutions:

- Define an adaptation procedure to be applied when random variables are realised = Simple (and fast) procedure
- Before the beginning of random variable realisations (*offline*) : Compute an *a priori* solution = Assign values that optimise the **expectation** of the objective function **wrt the adaptation procedure**
- Each time a random variable is realised (*online*) : $\rightsquigarrow$ Apply the adaptation procedure

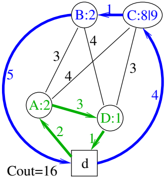
Ex.: Adaptation procedure for the VRP with stochastic demands

- Go back to the depot if current load + next demand $> k$

A priori Solution for $k = 11$:

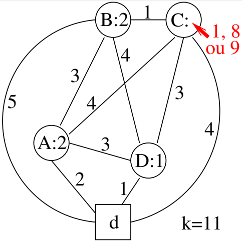


Adapted solution if $r_C = 8$ or 9 :



Expectation Optimisation vs Average Problem Optimisation

Problem (recall):

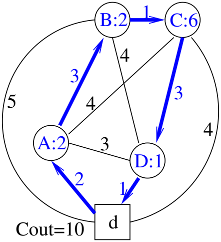


Request probability distributions:

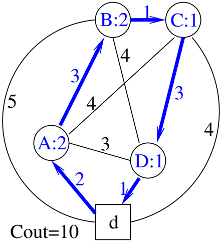
- $p(r_A = 2) = p(r_B = 2) = p(r_D = 1) = 1$
- $p(r_C = 1) = \frac{1}{3}, p(r_C = 8) = \frac{1}{3}, p(r_C = 9) = \frac{1}{3}$

What if we consider the "average" problem (i.e. $r_C = 6$)?

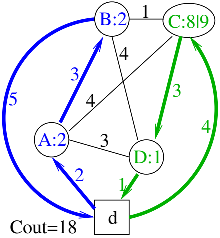
Optimal solution:



Adaptation if $r_C = 1$



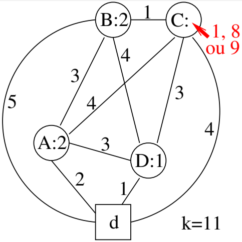
Adapt. if $r_C \in \{8, 9\}$



$\rightsquigarrow$ Expected cost $= \frac{10+18+18}{3} = \frac{46}{3}$. Can we do better?

Expectation Optimisation vs Average Problem Optimisation

Problem (recall):

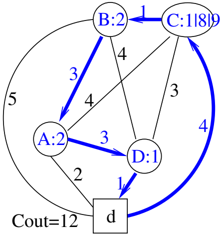


Request probability distributions:

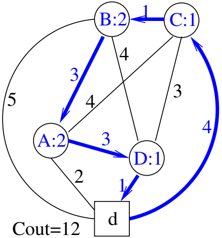
- $p(r_A = 2) = p(r_B = 2) = p(r_D = 1) = 1$
- $p(r_C = 1) = \frac{1}{3}, p(r_C = 8) = \frac{1}{3}, p(r_C = 9) = \frac{1}{3}$

Optimisation of the expected cost of adapted solutions:

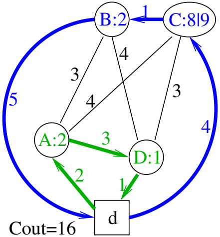
A priori Solution:



Adaptation if $r_C = 1$



Adapt. if $r_C \in \{8, 9\}$



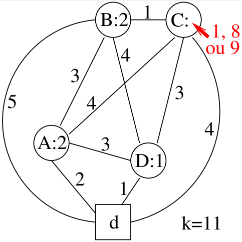
$\rightsquigarrow$ Expected cost $= \frac{12+16+16}{3} = \frac{44}{3}$. That's better!

Expectation Optimisation vs Average Problem Optimisation

Problem (recall):

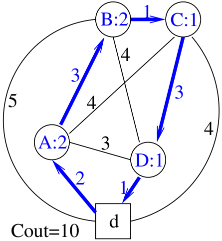
Request probability distributions:

- $p(r_A = 2) = p(r_B = 2) = p(r_D = 1) = 1$
- $p(r_C = 1) = \frac{1}{3}$, $p(r_C = 8) = \frac{1}{3}$, $p(r_C = 9) = \frac{1}{3}$

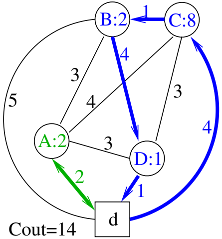


What if we have an oracle that knows the future?

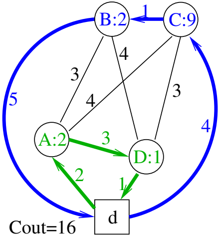
Optimal sol. if $r_C = 1$



Optimal sol. if $r_C = 8$



Optimal sol. if $r_C = 9$



$\rightsquigarrow$ Expected cost $= \frac{10+14+16}{3} = \frac{40}{3}$. That's even better, but oracles don't exist!

How to compute the expected cost of an *a priori* solution

$\rightsquigarrow$ **Use Dynamic Programming**

Example: Stochastic TSP

- Uncertain Data = Vertices to visit (some clients may be missing)
- Stochastic knowledge: Each vertex i is present with probability $p(i)$ and missing with probability $1 - p(i)$
- *A priori* solution: Hamiltonian cycle $(v_0, v_1, \ldots, v_n, v_0)$
- Adaptation: Skip vertices associated with missing clients

Bellman equations to compute the expected cost:

Let $e(v_i)$ = expected length of the adaptation of $(v_i, v_{i+1}, \ldots, v_n, v_0)$

- If $i = n$, then $e(v_i) = d_{v_i,v_0}$
- Otherwise, $e(v_i) = \sum_{j=i+1}^{n} Pr(v_i, v_j) * (d_{v_i,v_j} + e(v_j))$
  - $Pr(v_i, v_j) =$ proba that v_j is present and $v_{i+1}, \ldots, v_{j-1}$ are missing $\rightsquigarrow Pr(v_i, v_j) = (1 - p(v_{i+1})) * \ldots * (1 - p(v_{j-1})) * p(v_j)$

Expectation of an *a priori* solution = $e(v_0)$ computed in $\mathcal{O}(n^2)$

Problem: Not always possible to find Bellman's equations...

How to compute the expected cost of an *a priori* solution

$\rightsquigarrow$ **Use Monte Carlo sampling**

Expected cost of an *a priori* solution $A = \sum_{s \in S} Pr(s) \cdot cost_A(s)$

- S is the set of all possible scenarios
- $cost_A(s)$ = cost of A adapted to s

Example: Expectation of the length of a tour for the Stochastic TSP

- Scenario = subset of vertices (corresponding to present clients)
- For each subset $s \subseteq V$:
  - Probability of s = $\Pi_{i \in S} p(i) * \Pi_{i \in V \setminus S}(1 - p(i))$
  - $cost_A(s)$ = length of the subcycle of A that only contains nodes of s

Problem: The number of scenarios is exponential

Approximation with Monte-Carlo Sampling:

- Generate a representative subset of scenarios using probabilities
- For each sampled scenario, compute the cost of the adapted solution
- Return the average cost

Computation of an *a priori* solution with optimal expected cost

Exact approach: Branch & Cut (Integer L-shaped method)

- Drop some constraints (integrality, subtour elimination, etc)
- Replace the non-linear obj. function by a lower bounding variable z
- Iterate:
  - Solve the current problem
  - Add feasibility cuts if dropped constraints are violated
  - Add optimality cuts if $z <$ actual expected cost

Meta-Heuristic approaches (most often, local search-based):

- Generate an initial *a priori* solution
- While termination conditions not reached:
  - Change the values of some decision var. wrt some heuristics ($\neq$ possible heuristics: greedy, simulated annealing, tabu, etc)
  - Evaluate the impact on the expected cost
  - Accept or not the changes wrt some meta-heuristics
- Return the best *a priori* solution

Application to Police Patrol Management in Brussel

Description of the problem:

- Requests are revealed during the day, and must all be accepted
- Goal: Minimise service time expectation

Historical Data from 2013 to 2017:

⇝ Evolution of request localisation wrt time

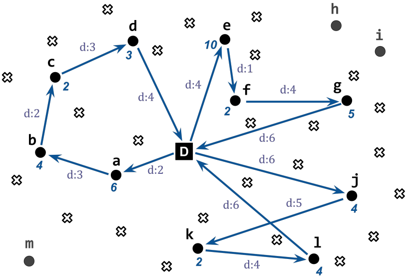
4 - 6am *6 - 8am* *8 - 10am*

Reference:

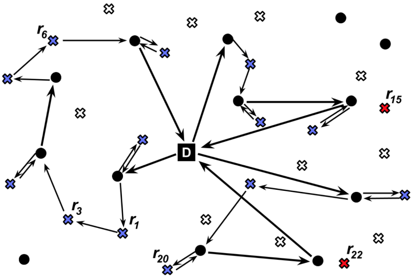
Saint-Guillain, Paquay, Limbourg: *Time-dependent stochastic vehicle routing problem with random requests: Application to online police patrol management in Brussels*

Introduction of waiting vertices and waiting times

Example of *a priori* solution:



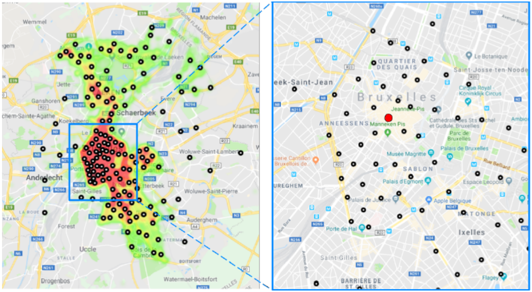
Example of adapted solution:



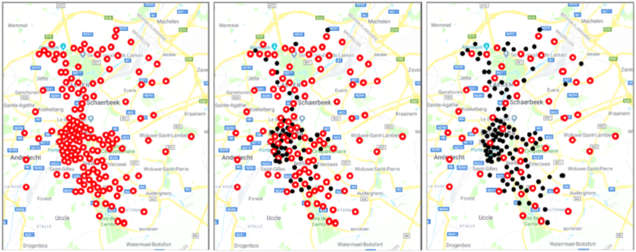
New algorithms:

- Dyn. prog. to compute exp. cost in $\mathcal{O}(n^2h^3q)$ time and $\mathcal{O}(nh^2q)$ space (n = nb of vertices, h = nb of time steps, q = max. capacity)
- Local Search approach to compute an approximate *a priori* solution

Choice of locations by spatial clustering



Waiting locations ($\in \{150, 100, 50\}$):



Results

Average relative gain wrt Wait & Serve strategy:

	Nb of waiting locations 50	100	150	Wait & Serve
3 vehicles	18.6%	17.0%	19.1%	11.6 mn
4 vehicles	25.4%	26.2%	28.1%	10.3 mn
6 vehicles	38.6%	39.3%	38.1%	10.0 mn

Conclusions:

- Exploiting stochastic knowledge allows to reduce service time
- Gain increases when increasing the number of vehicles
- Gain doesn't increase with the number of waiting locations
  $\leadsto$ Increasing the nb of waiting locations increases the search space size

Conclusion

How to exploit huge amounts of sensed Data?

- Descriptive analytics to understand
- Diagnostic analytics to explain
- Predictive analytics to forecast
- Prescriptive analytics to optimise
  - Time-Dependent optimisation for temporal Data
  - Stochastic optimisation for uncertain Data

Where are the challenges?

- $\mathcal{NP}$-hard problems for which complete approaches hardly scale
- Need accurate, reliable, and available Data
  $\rightsquigarrow$ Privacy and sustainability issues
- Citizens must be ready to use these smart services
  ... or smart services should adapt themselves to citizens!

Hot multidisciplinary research field!

Constrained Optimization

Model of a Constrained Optimization Problem (COP):

$\rightsquigarrow$ Define (X, D, C, F) with:

- X = Set of variables (unknowns)
- D = function which defines the domain $D(x_i)$ of every variable $x_i \in X$
  $\rightsquigarrow$ $D(x_i)$ = Set of values that may be assigned to x_i
- C = Constraints (relations between variables of X)
- $F : X \rightarrow \mathbb{R}$ = objective function to optimize

Solution of a problem (X, D, C, F):

Assignment of a value to every variable of X such that:

- Each variable $x_i \in X$ is assigned to a value that belongs to $D(x_i)$
- Every constraint of C is satisfied
- F is maximized (or minimized)

Remark: A problem may have several different models...

Example: Model for the TSP

Variables: $X = \{x_{i,j} \mid i,j \in V \times V, i \neq j\}$ **with** $D(x_{i,j}) = \{0,1\}$

$\rightsquigarrow x_{i,j} = 1$ if we travel to j just after i

Constraints:

- $\forall i \in V$, we must visit i once:

$$\forall i \in V, \sum_{j \in V} x_{i,j} = \sum_{j \in V} x_{j,i} = 1$$

- $\forall S \subset V$, no subtour:

$$\forall S \subset V, \sum_{(i,j) \in S \times S} x_{i,j} < |S|$$

Objective function: Minimize $\sum_{(i,j) \in S \times S} d_{i,j} * x_{i,j}$

Example for the tour $0 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 0$

$x_{0,3} = x_{3,1} = x_{1,2} = x_{2,4} = x_{4,0} = 1$

Example: Other model for the TSP

Variables: $X = \{next_i, visit_i \mid i \in V\}$

- $D(next_i) = V \setminus \{i\}$
  $\rightsquigarrow$ $next_i = j$ if the vertex visited after i is j
- $D(visit_i) = V$
  $\rightsquigarrow$ $visit_i = j$ if the ith visited vertex is j

Constraints:

- We start from and return back to 0: $visit_0 = 0$ and $next_{visit_{n-1}} = 0$
- The next of $visit_{i-1}$ is $visit_i$: $\forall i \in V \setminus \{0\}, visit_i = next_{visit_{i-1}}$
- Each vertex is visited once: $allDifferent(visit)$
- Each vertex follows a different vertex: $allDifferent(next)$

Objective function: Minimize $\sum_{i \in V} d_{i,next_i}$

Example for the tour $0 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 0$

next

0	1	2	3	4
3	2	4	1	0

visit

0	1	2	3	4
0	3	1	2	4

Some particular cases

- No constraint:
  $\rightsquigarrow$ Optimization problem
- No objective function:
  $\rightsquigarrow$ Constraint Satisfaction Problem (CSP)
- Domains are discrete (enumerable)
  $\rightsquigarrow$ Combinatorial problem
- *F* linear, $D = \mathbb{R}$ and *C* = linear inequalities
  $\rightsquigarrow$ Linear Programming (LP)
- *F* linear, $D = \mathbb{Z}$ and *C* = linear inequalities
  $\rightsquigarrow$ Integer Linear Programming (ILP)
- *F* linear, $D = \{0, 1\}$ and *C* = linear inequalities
  $\rightsquigarrow$ Knapsack problem
- *F* quadratic, $D = \mathbb{R}$ and *C* = linear inequalities
  $\rightsquigarrow$ Quadratic Problem
- ...

Complexity

Some particular cases have polynomial complexities:

- Linear programming with continuous domains
- 2-SAT
- Assignment problems
- Shortest path problems
- ...

They are most often NP-hard:

- ILP, Knapsack
- SAT, 3-SAT, Planar-3-SAT, ...
- Many graph problems:
  Coloring, TSP, max Clique, ...
- CSP with finite domains
- ...

In some cases they are undecidable:

- Diophantine equations
- CSP with non finite domains
- ...

How to solve NP-hard problems?

- Some instances of NP-hard problems may be easy to solve
  $\rightsquigarrow$ far from phase transition, landscapes with few local optima, ...
- Some NP-hard problems become polynomial when adding constraints
- Some NP-hard problems may be approximated in polynomial time (with bounds on errors)
- Otherwise, we have to be intelligent when exploring the search space
  - Heuristic approaches:
    $\rightsquigarrow$ Avoid explosion by ignoring some parts of the search space
  - Complete approaches:
    $\rightsquigarrow$ Prevent explosion by structuring and filtering search space

Heuristic approaches

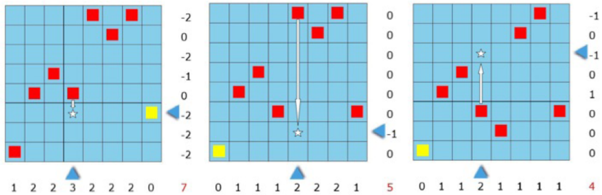
Exploration guided by (meta-)heuristics

- Intensify search around the most promising areas
- Diversify search to discover new areas

Two kinds of heuristic approaches

- Perturbative: Modify existing combinations
  - Ex: Local search (LS), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), ...
- Constructive: Build new combinations from scratch
  - Ex: Ant Colony Optimization (ACO), Estimation of Distribution Algorithms (EDA), ...

Example of local search for the 8-queen problem:



Complete approaches

Ad hoc approaches

- *Branch & Bound*, *Branch & Cut*, *Branch & Price*, ...
- Dynamic programming
- ...

Generic approaches: Problem $\leadsto$ Model $\leadsto$ Generic solver

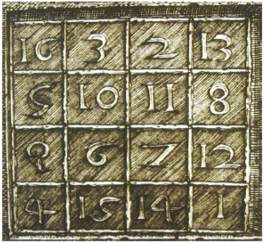
- MILP (*Mixed Integer Linear Programming*)
  $\leadsto$ Numerical variables; Constraints = Linear inequalities
- SAT (satisfiability of Boolean formulae)
  $\leadsto$ Boolean variables; Constraints = Logical clauses
- CP (*Constraint Programming*)
  $\leadsto$ Any kind of variables and constraints

Why using CP?

- Ease of modelling
- Efficiency

Why using CP instead of SAT?

Magic Square



- SAT: 256 variables and ~65000 clauses
- CP:

```
% All different on cells
constraint
alldifferent(i,j in 1..4)(magic[i,j]);

% sum in rows.
constraint forall (i in 1..4) (
  sum(j in 1..4)(magic[i,j]) = 34 );

% sum in columns.
constraint forall (j in 1..4) (
  sum(i in 1..4)(magic[i,j]) = 34 );

% sum in diagonals.
constraint
  sum(i in 1..4)(magic[i,i]) = 34;
  sum(i in 1..4)(magic[i,4-i+1]) = 34;

solve satisfy;
```

Slide from Christian Bessière

Why using CP instead of SAT?

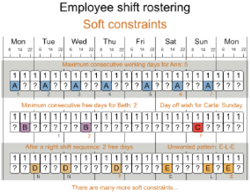
Pigeon Hole

- In 2010, SAT was not able to solve it with more than 15 pigeons (no polynomial-size proof)
- CP solves it in milliseconds

Slide from Christian Bessière

Why using CP instead of ILP?

Nurse Rostering



- Linear Programming: more than 10.000 lines
- CP:

```
int: Q = 6; int: q0 = 1; set of int: STATES = 1..Q;
array[STATES,SHIFTS] of int: t =
     [| 2, 3, 1    % state 1
      | 4, 4, 1    % state 2
      | 4, 5, 1    % state 3
      | 6, 6, 1    % state 4
      | 6, 0, 1    % state 5
      | 0, 0, 1|]; % state 6

array[NURSES,DAYS] of var SHIFTS: roster;

constraint forall(j in DAYS)(
             sum(i in NURSES)(bool2int(roster[i,j] == d)) == req_day /\
             sum(i in NURSES)(bool2int(roster[i,j] == n)) == req_night
           );
constraint forall(i in NURSES)(
             regular([roster[i,j] | j in DAYS], Q, S, t, q0, STATES) /\
             sum(j in DAYS)(bool2int(roster[i,j] == n)) >= min_night
           );

solve satisfy;
```

Slide from Christian Bessière

Why using CP instead of ILP?

Sports League Scheduling

- In 1995, ILP was not able to solve the NHL problem with more than 12 teams (NHL involves 30 teams)
- CP solved it up to 60 teams

Slide from Christian Bessière

Why using CP instead of a dedicated approach?

⇝ Differential cryptanalysis of AES

Basic model in Picat:

```
basicModel(R, ObjStep1, DX, DY, DK) =>
    DX = new_array(R,4,4),      DX :: 0..1,
    DY = new_array(R-1,4,4),    DY :: 0..1,
    DK = new_array(R,4,4),      DK :: 0..1,
    foreach (I in 1..R-1, J in 1..4, K in 1..4)     %%%%%% ARK constraint
        sum([DY[I,J,K],DK[I+1,J,K],DX[I+1,J,K]]) #!= 1
    end,
    foreach(I in 1..R-1, K in 1..4)                 %%%%%% MC constraint
        DX[I,1,K] + DX[I,2,(K mod 4)+1] + DX[I,3,((1+K) mod 4)+1] + DX[I,4,((2+K) mod 4)+1] + DY[I,1,K] + DY[I,2,K] + DY[I,3,K] + DY[I,4,K] #= S,
        S notin 1..4
    end,
    foreach(I in 2..R, J in 1..4)                   %%%%%% KS constraint
        sum([DK[I-1,J,1],DK[I-1,(J mod 4)+1,4],DK[I,J,1]]) #!= 1,
        foreach(K in 2..4)
            sum([DK[I-1,J,K],DK[I,J,K-1],DK[I,J,K]]) #!= 1
        end
    end,
    sum([[DX[I,J,K] : I in 1..R, J in 1..4, K in 1..4]) + sum([DK[I,J,4] : I in 1..R, J in 1..4]) #= ObjStep1.
```

Advanced model in Picat:

- Less than 200 lines of code
- Solve all instances in less than 4h (for keys of 128, 192, and 256 bits)
    - Branch & Bound [Biryukov et al 2010] :
      ⇝ Several days (weeks) for keys of 128 (192) bits
    - Graph traversal [Fouque et al 2013] :
      ⇝ 30mn/12 cores for 128 bits, but 60GB of RAM

CP Langages and Libraries

- ALICE [Jean-Louis Laurière, 1976]
  $\rightsquigarrow$ First CP approach
- CHIP, Prolog V, Gnu-Prolog, Picat
  $\rightsquigarrow$ Extensions of Prolog
- CHOCO (Java), Gecode (C++), OR-Tools (C++)
  $\rightsquigarrow$ Open source libraries
- OPL Development Studio (IBM)
  $\rightsquigarrow$ Modelling language + CP + MIP
- ...

Example: Choco code for the TSP

```
public void solveTSP(Graph g) {
    Model model = new Model("TSP");

    // Create variables
    IntVar[] next = new IntVar[g.getNbVertices()];
    for (int i = 0; i < g.getNbVertices(); i++)
        next[i] = model.intVar(g.getSucc(i));
    IntVar[] cost = model.intVarArray(g.getNbVertices(), g.getMinCost(), g.getMaxCost());
    IntVar totalCost = model.intVar(g.getNbVertices()*g.getMinCost(), g.getNbVertices()*g.getMaxCost(

    // Add constraints
    for (int i = 0; i < g.getNbVertices(); i++)
        model.element(cost[i], g.getCost(i), next[i]).post();
    model.circuit(next,0).post();
    model.sum(cost, "=", totalCost).post();

    // Solve
    model.setObjective(Model.MINIMIZE, totalCost);
    while (model.getSolver().solve());
}
```

Example: OPL model for the ATSP

Input Data:

```
range Points = 1..nbPoints;
int duree[Points][Points] =...
```

Variables :

```
dvar int visit[Points] in Points;        /* visit[i] = ième point visité */
dvar int h[Points] in T;                 /* h[i] = heure d'arrivée sur i */
```

Objective function to optimize and constraints:

```
minimize h[v_final];
subject to {
    visit[1] == v_init;                          /* On part de v_init */
    h[v_init] == t_0;                            /* à l'heure t_0 */
    visit[nbPoints] == v_final;                  /* et on termine sur v_final */
    allDifferent(visit);                         /* Chaque point est visité une fois */
    forall(i ∈ 1..nbPoints − 1){
        h[visit[i+1]] == h[visit[i]] + duree[visit[i]][visit[i+1]]
    }
}
```

Generic Solving Algorithm

Function *solve*(X, D, C)

- **Input** : A CSP (X, D, C)
- **Precondition** : (X, D, C) is locally consistent
- **Postcondition** : Return a solution of (X, D, C) or $\emptyset$ if no solution

if *for each variable* $x_i \in X, |D(x_i)| = 1$ **then return** D;

Choose a variable $x_i \in X$ such that $|D(x_i)| > 1$

for *each value* $v \in D(x_i)$ **do**

    Save D and reduce $D(x_i)$ to $\{v\}$

    if (X, D, C) *is locally consistent* **then**

        $Sol \leftarrow$ solve(X, D, C)

        if $Sol \neq \emptyset$ **then return** Sol;

    Restore D

return $\emptyset$

Generic Solving Algorithm

```
Function solve(X, D, C)
    Input          : A CSP (X, D, C)
    Precondition   : (X, D, C) is locally consistent
    Postcondition  : Return a solution of (X, D, C) or ∅ if no solution
    if for each variable x_i ∈ X, |D(x_i)| = 1 then return D;
    Choose a variable x_i ∈ X such that |D(x_i)| > 1
    for each value v ∈ D(x_i) do
        Save D and reduce D(x_i) to {v}
        if (X, D, C) is locally consistent then
            Sol ← solve(X, D, C)
            if Sol ≠ ∅ then return Sol;
        Restore D
    return ∅
```

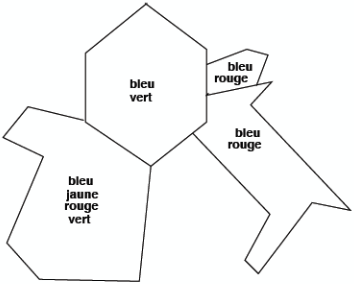
Local consistency of a CSP (X, D, C)

Check that each constraint can be satisfied

- Different levels of consistency may be considered
- Simplest consistency: check constraints whose variables have singleton domains

Exercise

Map coloring:



CSP:

- $X = \{F, E, S, I\}$
- $D(F) = \{b, v\}$,
  $D(E) = \{b, j, r, v\}$,
  $D(S) = D(I) = \{b, r\}$
- $C = \{F \neq E,$
  $F \neq S,$
  $F \neq I,$
  $S \neq I\}$

Build the search tree (tree of the recursive calls to *solve*)
$\rightsquigarrow$ Choose variables according to the order: F, E, S, I

Improvements of *solve*: Constraint propagation

Why propagating constraints?

- Avoid encoutering several times a same inconsistency
  $\leadsto$ Filter domains to ensure some given consistency
  $\leadsto$ No recursive call if a domain is empty
- Different consistency levels may be considered: Arc Consistency (AC), k-consistency, Singleton AC, etc
  $\leadsto$ Different strengthes and different complexities

Most popular consistency: AC

- Let c be a constraint defined over a set X_c of k variables. c is AC if: $\forall x_i \in X_c, \forall v_i \in D(x_i), \forall x_j \in X_c \setminus \{x_i\}, \exists v_j \in D(x_j)$ such that c is satisfied by the assignment $x_1 = v_1, ..., x_k = v_k$
- Different algorithms for ensuring AC
  - AC3 (binary constraints): $\mathcal{O}(ed^3)$ in time; $\mathcal{O}(e)$ in space
  - AC2001 (binary constraints): $\mathcal{O}(ed^2)$ in time; $\mathcal{O}(ed)$ in space
  - STR: $\mathcal{O}(t)$ in time and space (t = number of allowed tuples)

Exercise

CSP associated with map coloring (recall):

- $X = \{F, E, S, I\}$
- $D(F) = \{b, v\}$, $D(E) = \{v\}$, $D(S) = D(I) = \{b, r\}$
- $C = \{F \neq E, F \neq S, F \neq I, S \neq I\}$

Filter domains to ensure AC

Improvements of *solve*: Learning from failures

What can we do when a domain becomes empty?

- *backtrack*: Go back to the last call
- *backjump*: Go back to the call that assigned the last variable involved in the failure
- Learn *nogoods*: Add the failure cause to constraints

Example

- $X = \{x_1, x_2, x_3, x_4, x_5\}$
- $D(x_1) = \{5, 6\}$, $D(x_2) = \{2, 3\}$, $D(x_3) = \{3, 4\}$, $D(x_4) = D(x_5) = \{4, 5\}$
- All variables must be assigned to different values

Improvements of *solve*: Ordering heuristics

Function *solve*(X, D, C)
- **if** *for each variable* $x_i \in X, |D(x_i)| = 1$ **then return** D;
- Choose a variable $x_i \in X$ such that $|D(x_i)| > 1$
- **for** *each value* $v \in D(x_i)$ **do**
  - Save D and reduce $D(x_i)$ to $\{v\}$
  - **if** (X, D, C) *locally consistent* **then**
    - $Sol \leftarrow$ solve(X, D, C)
    - **if** $Sol \neq \emptyset$ **then return** Sol;
  - Restore D
- **return** $\emptyset$

Variable ordering heuristics

- *deg*: Variable involved in the largest number of constraints
  $\rightsquigarrow$ Reduce tree depth
- *dom*: Variable with the smallest domain
  $\rightsquigarrow$ Reduces tree width
- $\frac{dom}{deg}$: Compromise between *dom* and *deg*
- $\frac{dom}{wdeg}$: Each constraint has a weight (incremented on failures)
  $\rightsquigarrow$ divide $|D(x_i)|$ by sum of weights of constraints associated with x_i

Improvements of *solve*: Ordering heuristics

Function *solve*(X, D, C)
  if *for each variable* $x_i \in X, |D(x_i)| = 1$ **then return** D;
  Choose a variable $x_i \in X$ such that $|D(x_i)| > 1$
  for *each value* $v \in D(x_i)$ **do**
    Save D and reduce $D(x_i)$ to $\{v\}$
    if (X, D, C) *locally consistent* **then**
      $Sol \leftarrow \text{solve}(X, D, C)$
      if $Sol \neq \emptyset$ **then return** Sol;
    Restore D
  return $\emptyset$

Value ordering heuristics

Choose values that are more likely to belong to solutions
$\rightsquigarrow$ No universal heuristic

- May be learned... but this may be expensive
- Useless for proving inconsistency of infeasible instances
- 95% of the solving time is spent on inconsistent sub-trees

Global constraints

What is a global constraint?

Constraint defined over a set of constraints (the cardinality of which is not fixed)

Examples of global constraints:

- allDifferent($x_1, \ldots, x_n$)
- sum($x_1, \ldots, x_n, s$)
- atLeast($x_1, \ldots, x_n, k, v$)

Why global constraints?

- Ease modelling
- Improve propagation:
  $\rightsquigarrow$ filter more values and/or reduce time complexity

Decomposition of global constraints

How to decompose a global constraint?

Replace the constraint with an equivalent set of non global constraints

Example 1: allDifferent$(x_1, \ldots, x_n)$

- $\forall i, j \in [1, n], i \neq j : x_i \neq x_j$

Example 2: atLeast$(x_1, \ldots, x_n, k, v)$

Introduction of n new variables $s_1, \ldots, s_n$ such that $D(x_i) = [0, i]$

- $s_1 = (x_1 == v)$
- $\forall i \in [2, n] : s_i = s_{i-1} + (x_i == v)$
- $s_n \geq k$

Example 3: sum$(x_1, \ldots, x_n, s)$

Introduction of n new variables $s_1, \ldots, s_n$

- $s_1 = x_1$
- $\forall i \in [2, n] : s_i = s_{i-1} + x_i$
- $s = s_n$

Propagation of global constraints

AC-decomposable constraint

There exists a polynomial size decomposition such that the decomposition is AC iff the global constraint is AC

Example: atLeast$(x_1, \ldots, x_n, k, v)$ is AC-decomposable

Propagating the constraints $s_i = s_{i-1} + (x_i == v)$ filters the same values as propagating atLeast$(x_1, \ldots, x_n, k, v)$, but more efficiently

Example: sum$(x_1, \ldots, x_n, s)$ is not AC-decomposable

Proof: deciding if an instance of sum is AC is an $\mathcal{NP}$-hard problem

What about *allDifferent*?

- The binary decomposition filters less values
- Deciding if an instance of *allDifferent* is AC is in $\mathcal{P}$
- Can we find a decomposition that preserves AC?
  $\rightsquigarrow$ No!

Dedicated filtering algorithms

What can we do when a constraint is neither $\mathcal{NP}$-hard nor AC-decomposable?

$\leadsto$ Implement a propagation algorithm that ensures AC in polynomial time!

Propagation of allDifferent

$\leadsto$ Matching algorithm of Hopcroft and Karp (1973)

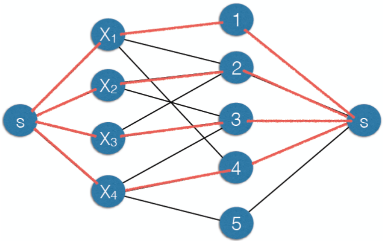


Figure from Christian Bessière

allDifferent(x_1, x_2, x_3, x_4)

- $D(x_1) = \{1, 2, 4\}$
- $D(x_2) = \{2, 3\}$
- $D(x_3) = \{2, 3\}$
- $D(x_4) = \{3, 4, 5\}$

Dedicated filtering algorithms

What can we do when a constraint is neither $\mathcal{NP}$-hard nor AC-decomposable?

$\rightsquigarrow$ Implement a propagation algorithm that ensures AC in polynomial time!

Propagation of allDifferent

$\rightsquigarrow$ Matching algorithm of Hopcroft and Karp (1973)

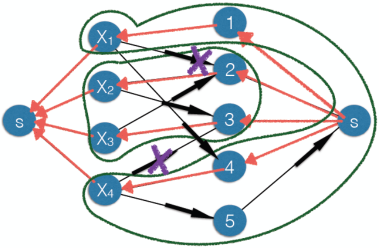


Figure from Christian Bessière

allDifferent(x_1, x_2, x_3, x_4)

- $D(x_1) = \{1, 2, 4\}$
  $\rightsquigarrow$ remove 2 from $D(x_1)$
- $D(x_2) = \{2, 3\}$
- $D(x_3) = \{2, 3\}$
- $D(x_4) = \{3, 4, 5\}$
  $\rightsquigarrow$ remove 3 from $D(x_4)$