New Results for the GEO-CAPE Observation Scheduling Problem

Philippe Laborie, Bilal Messaoudi
IBM, IBM Analytics
laborie@fr.ibm.com

2017

Overview

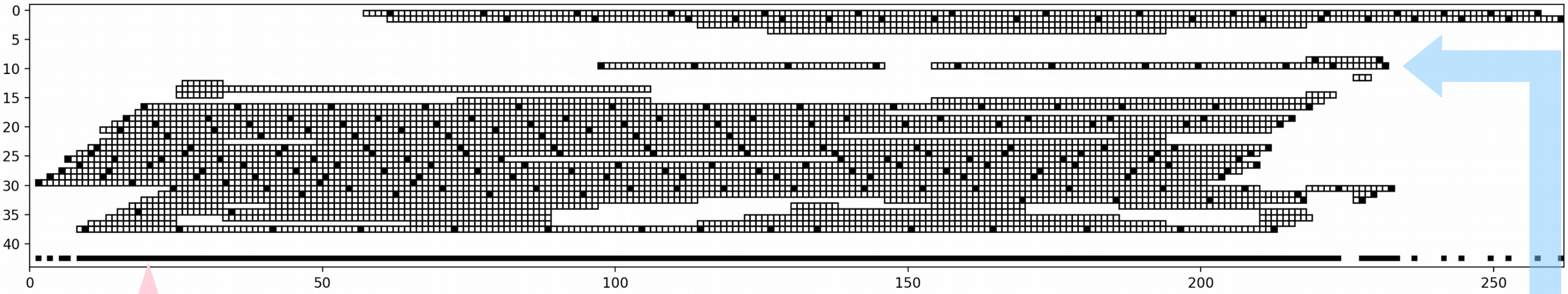
- Problem description
- Previous results
- Our contributions:
  - Providing optimality bounds (MILP model)
  - Finding better quality solutions (CP Optimizer model)

Problem description

- NASA satellite observation scheduling problem proposed in J. Frank, M. Do and T. T. Tran. *Scheduling Ocean Color Observations for a GEO-Stationary Satellite*. In Proc. ICAPS-2016.
- A set of *n* scenes to be observed by a satellite instrument during one day
- Each scene is observed multiple times during the day
- All observations have the same duration (1 time-unit)
- Scenes are not always observable (due to sunrise time, cloud coverage, ...): each scene is defined as a set of observability time slots
- The satellite instrument can only observe one scene at a time

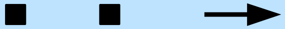
Problem description

- Example of a problem instance and a feasible solution



A scene with its possible observability time-slots

Observability time-slots

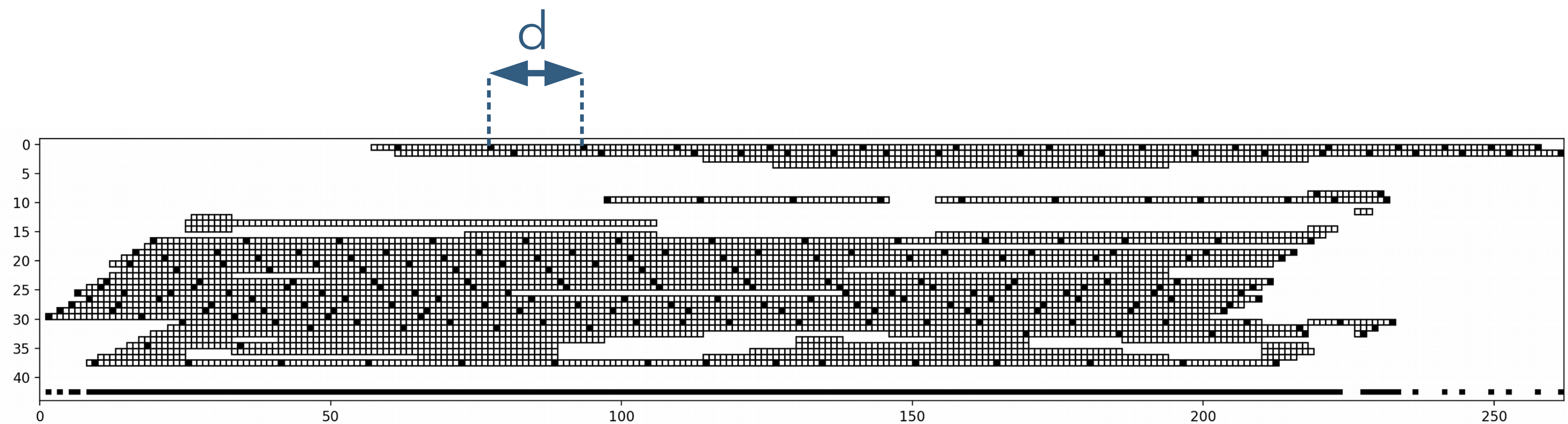


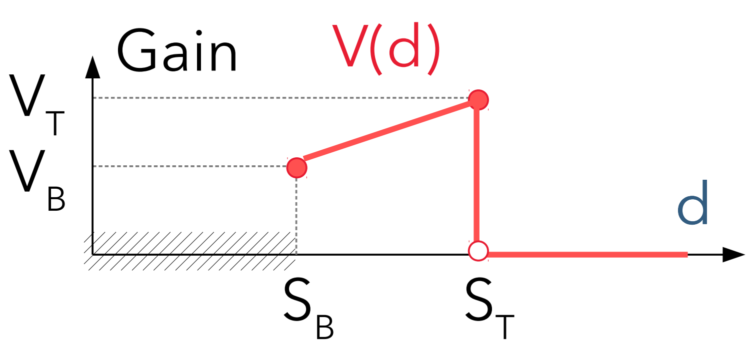
Scheduled observations

All scheduled observations

Problem description

- Schedule quality depends on the separation time *d* between consecutive observations of each scene



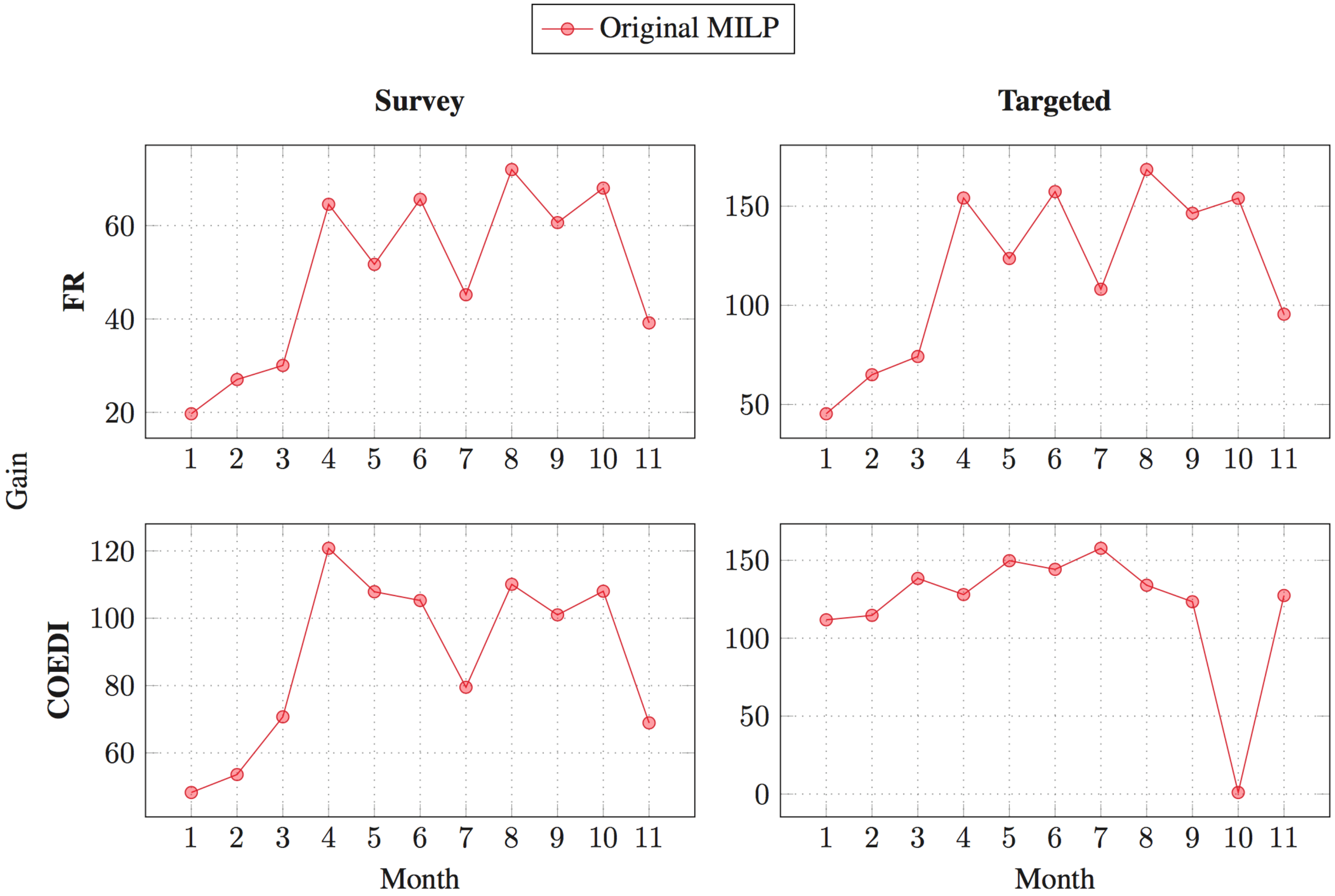


- Objective: **maximize** total gain due to separation times
- Number of scheduled observations is unknown

Previous results

- Benchmark: 44 realistic instances
  - 2 types of instruments (COEDI/FR) x
  - 2 types of science experiments (Survey/Targeted) x
  - 11 observability scenarios corresponding to weather condition at different period of the year (months)
- Best solutions were obtained with a time-indexed MILP formulation
  - Main decision variables
    $o_{ij} \in \{0,1\}$ s.t. $o_{ij}=1$ iff scene *i* is observed at time-slot *j*
- Computation time: 1h
- Problems could not be solved to optimality, no optimality bound available

Previous results



Optimality bounds (MILP model)

- Disjunctive MILP model (details in the paper)
  - Main decision variables:
    $b_{i1,j1;i2,j2} \in \{0,1\}$ s.t. $b_{i1,j1;i2,j2}=1$ iff
    the $j1^{th}$ observation of scene i1 is performed before
    the $j2^{th}$ observation of scene i2

Optimality bounds (MILP model)

- Disjunctive MILP model (details in the paper)
  - Main decision variables:
    $b_{i1,j1;i2,j2} \in \{0,1\}$ s.t. $b_{i1,j1;i2,j2}=1$ iff the $j1^{th}$ observation of scene i1 is performed before the $j2^{th}$ observation of scene i2

$$\max \sum_{i\in\Psi}\sum_{j\in A_i^*} v_{i,j}$$

$$t_{i_1,j_1} \geq t_{i_2,j_2} + o_{i_2,j_2} - |H|\, b_{i_1,j_1;i_2,j_2} \quad \forall i_1,i_2\in\Psi, i_1\neq i_2, C^{i_1}\cap C^{i_2}\neq\emptyset, \forall j_1\in A_{i_1}, \forall j_2\in A_{i_2} \quad (1)$$

$$t_{i_2,j_2} \geq t_{i_1,j_1} + o_{i_1,j_1} - |H|\,(1-b_{i_1,j_1;i_2,j_2}) \quad \forall i_1,i_2\in\Psi, i_1\neq i_2, C^{i_1}\cap C^{i_2}\neq\emptyset, \forall j_1\in A_{i_1}, \forall j_2\in A_{i_2} \quad (2)$$

$$t_{i,j} \geq \min(C^i) \quad \forall i\in\Psi, \forall j\in A_i \quad (3)$$

$$t_{i,j} \leq \max(C^i) \quad \forall i\in\Psi, \forall j\in A_i \quad (4)$$

$$t_{i,j} > \max(\text{NoObs}^i\,[k])(1-b_{i,j;k}) \quad \forall i\in\Psi, \forall j\in A_i, \forall k\in \text{NoObs}^i \quad (5)$$

$$t_{i,j} < \min(\text{NoObs}^i\,[k])b_{i,j;k} + |H|\,(1-b_{i,j;k}) \quad \forall i\in\Psi, \forall j\in A_i, \forall k\in \text{NoObs}^i \quad (6)$$

$$t_{i,j+1} \geq t_{i,j} + S_B^i \cdot o_{i,j+1} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (7)$$

$$o_{i,j} \geq o_{i,j+1} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (8)$$

$$d_{i,j} \geq 0 \quad \forall i\in\Psi, \forall j\in A_i^* \quad (9)$$

$$d_{i,j} \leq |H|\, o_{i,j+1} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (10)$$

$$d_{i,j} = t_{i,j+1} - t_{i,j} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (11)$$

$$d_{i,j} \leq S_T^i \cdot z_{i,j} + |H|\,(1-z_{i,j}) \quad \forall i\in\Psi, \forall j\in A_i^* \quad (12)$$

$$d_{i,j} \geq S_T^i \cdot (1-z_{i,j}) \quad \forall i\in\Psi, \forall j\in A_i^* \quad (13)$$

$$v_{i,j} \leq V_T^i \cdot z_{i,j} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (14)$$

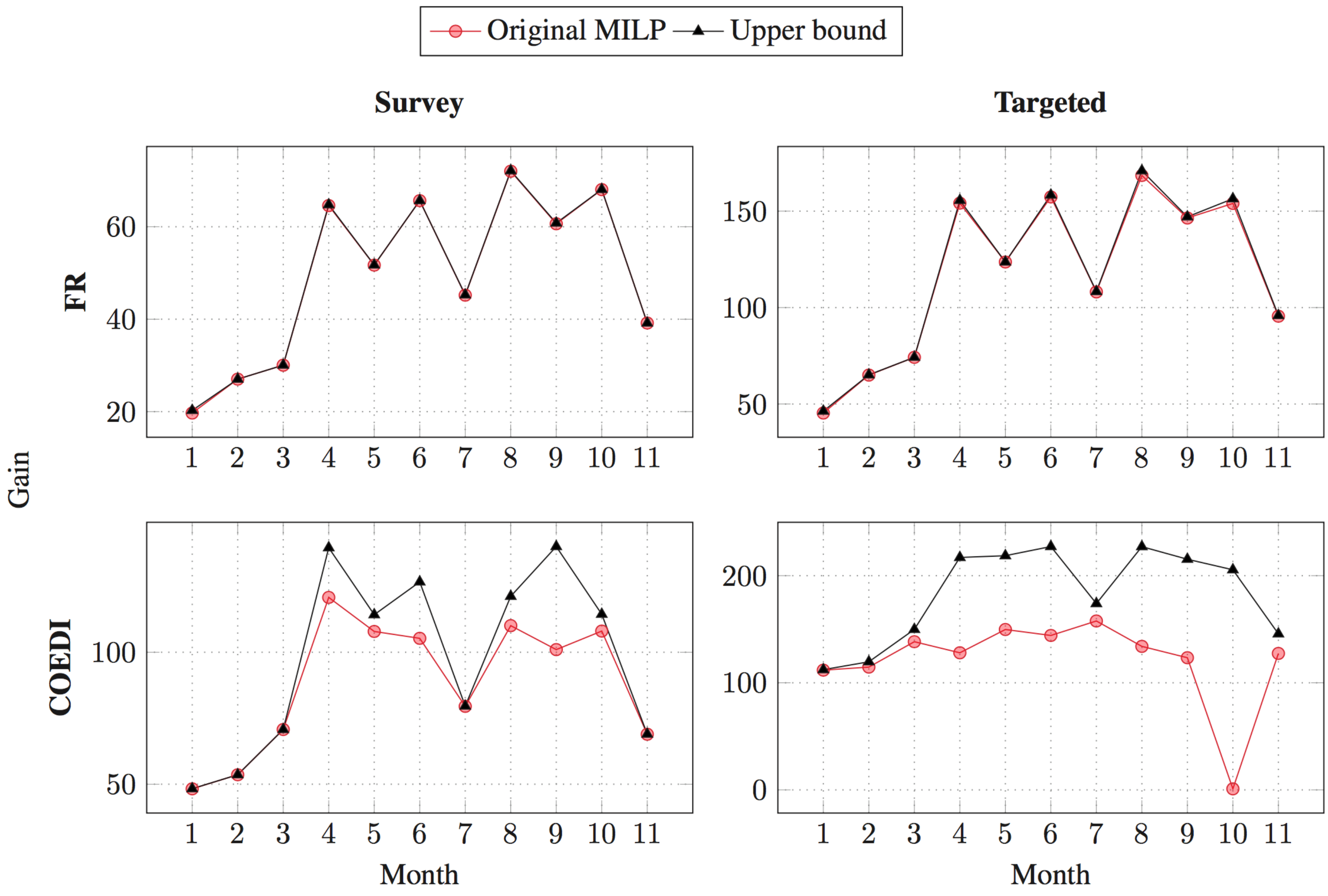
$$v_{i,j} \leq V_T^i \cdot o_{i,j+1} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (15)$$

$$v_{i,j} \leq V_B^i + |H|\,(1-z_{i,j}) + (d_{i,j} - S_B^i)\cdot\frac{V_T^i - V_B^i}{S_T^i - S_B^i} \quad \forall i\in\Psi, \forall j\in A_i^* \quad (16)$$

Optimality bounds (MILP model)

- Disjunctive MILP model (details in the paper)
  - Quickly solve easy instances (FR instrument) to optimality
  - Bad quality solutions / optimality bounds on the other ones
  - The model can be relaxed (energetic relaxation of observations no-overlap) to produce optimality bounds (upper-bounds)

Optimality bounds (MILP model)



Good quality solutions (CP Optimizer model)

What is CP Optimizer?

- An optimization engine for solving combinatorial problems (with a particular focus on scheduling problems)
- Available in IBM ILOG CPLEX Optimization Studio
- Implements a model & run paradigm (like CPLEX):
  - Problem is formulated as a **declarative model**
    - Extends classical combinatorial optimization framework with concepts like optional interval variables, functions of time, specialized constraints (e.g. noOverlap), …
    - Available in different programming languages (C++, Python, Java, .NET and OPL)
  - Powerful **automatic search** being continuously improved
    - Search is complete
    - Internally uses many techniques (CP, MP, meta-heuristics,...)

Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Data reading and constants

Decision variables

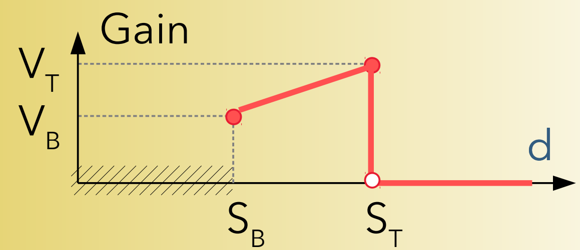
Objective function

Constraints

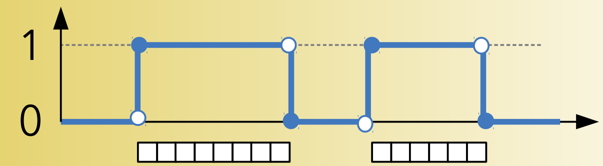
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

`TL`: origin of the schedule
`TU`: horizon of the schedule
`m` : upper-bounds on number of observations of a given scene
`V` : gain function



`NoObs[i]`: for observation *i*, step function equal to *0* on time-slots where scene *i* is not observable and *1* otherwise



Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Decision variables

Interval variable: decision variable *x* with a domain:

$$\mathrm{Dom}(x) \subseteq \{\perp\} \cup \{ [s,e) \mid s,e \in Z, s \leq e \}$$

Absent interval

Interval of integers

Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

`a[i,j]`: interval variable of size 1 representing the *j*th observation of scene *i* (if *i* is observed at least *j* times), otherwise a[i,j] is absent

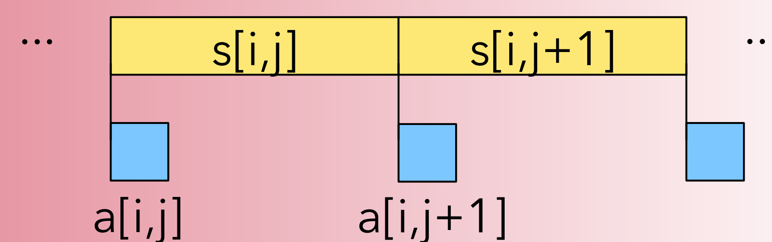
`s[i,j]`: interval variable representing the separation time between a[i,j] and a[i,j+1]

Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

`a[i,j]`: interval variable of size 1 representing the j^{th} observation of scene *i* (if *i* is observed at least *j* times), otherwise a[i,j] is absent

`s[i,j]`: interval variable representing the separation time between a[i,j] and a[i,j+1]

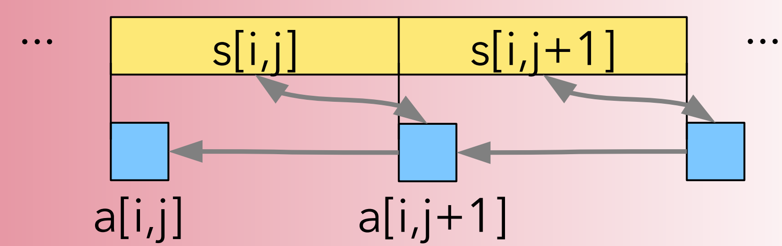


Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

`a[i,j]`: interval variable of size 1 representing the j^{th} observation of scene *i* (if *i* is observed at least *j* times), otherwise a[i,j] is absent

`s[i,j]`: interval variable representing the separation time between a[i,j] and a[i,j+1]



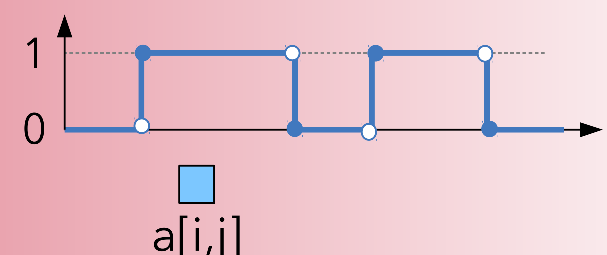
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

`a[i,j]`: interval variable of size 1 representing the j^{th} observation of scene *i* (if *i* is observed at least *j* times), otherwise a[i,j] is absent

`s[i,j]`: interval variable representing the separation time between a[i,j] and a[i,j+1]

Interval variable a[i,j] cannot overlap a time-slot that is not observable (NoObs[i](t)=0)



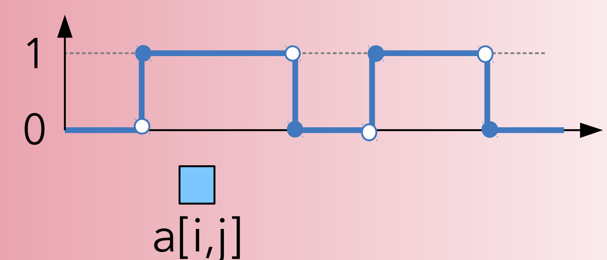
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

`a[i,j]`: interval variable of size 1 representing the j^{th} observation of scene *i* (if *i* is observed at least *j* times), otherwise a[i,j] is absent

`s[i,j]`: interval variable representing the separation time between a[i,j] and a[i,j+1]

Interval variable a[i,j] cannot overlap a time-slot that is not observable (NoObs[i](t)=0)



Intervals a[i,j] do not overlap as the the instrument can observe only one scene at a time

Good quality solutions (CP Optimizer model)

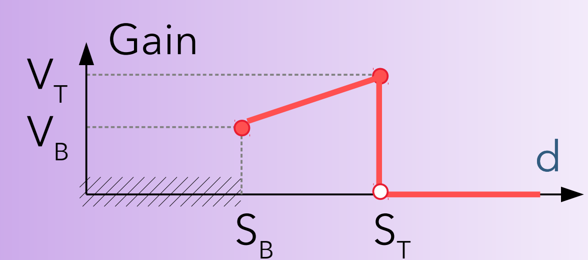
```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

a[i,j]: interval variable of size 1 representing the j^{th} observation of scene *i* (if *i* is observed at least *j* times), otherwise a[i,j] is absent

s[i,j]: interval variable representing the separation time between a[i,j] and a[i,j+1]

Objective is to maximize the total gain due to separation time between observations (length of interval variables s[i,j]):

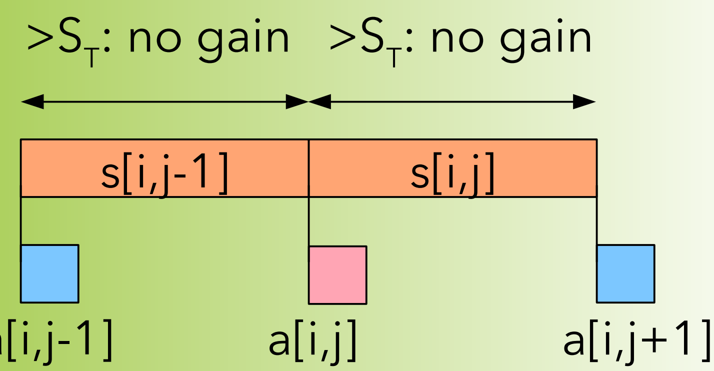
$$\sum_{i,j} V(\text{lengthOf}(s[i,j]))$$



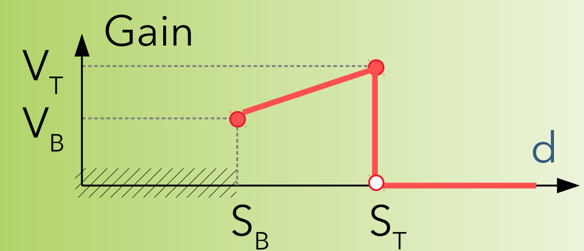
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Notion of an isolated observation a[i,j]:



Property: there exist an optimal solution that does not contain any isolated observation



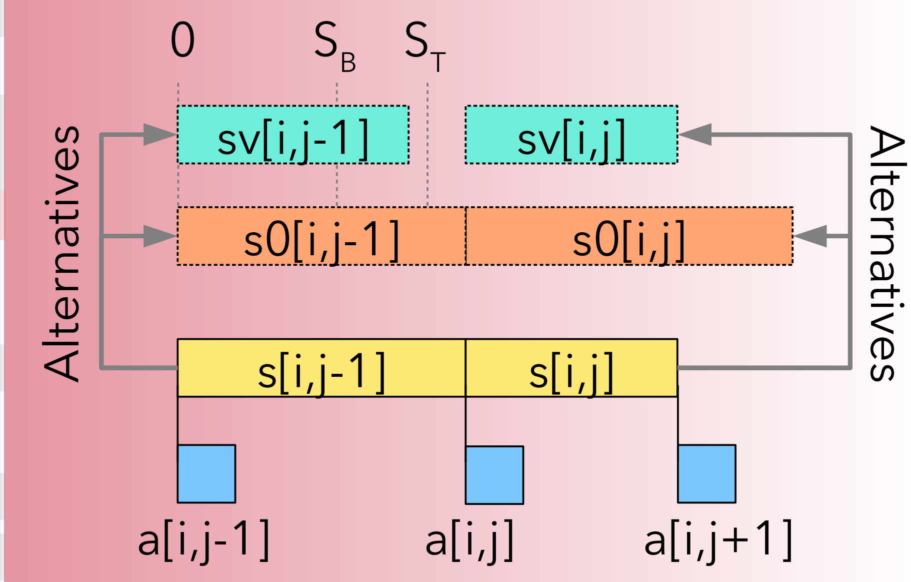
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Two possibilities are considered for a separation s[i,j]:

sv[i,j]: length is in $[S_B,S_T]$ and it brings some value

s0[i,j]: length is $>S_T$ and it does not bring any value



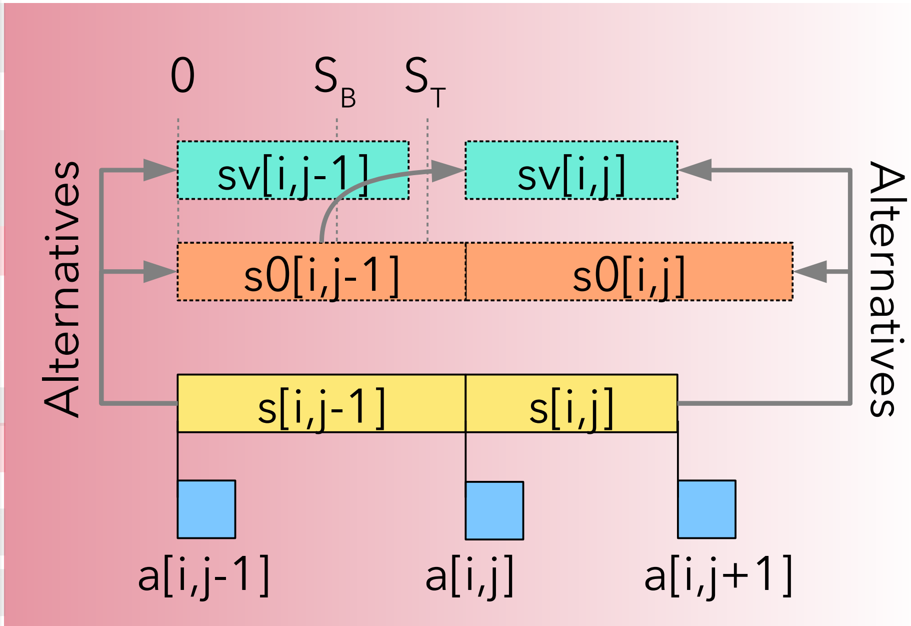
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Two possibilities are considered for a separation s[i,j]:

`sv[i,j]`: length is in $[S_B,S_T]$ and it brings some value

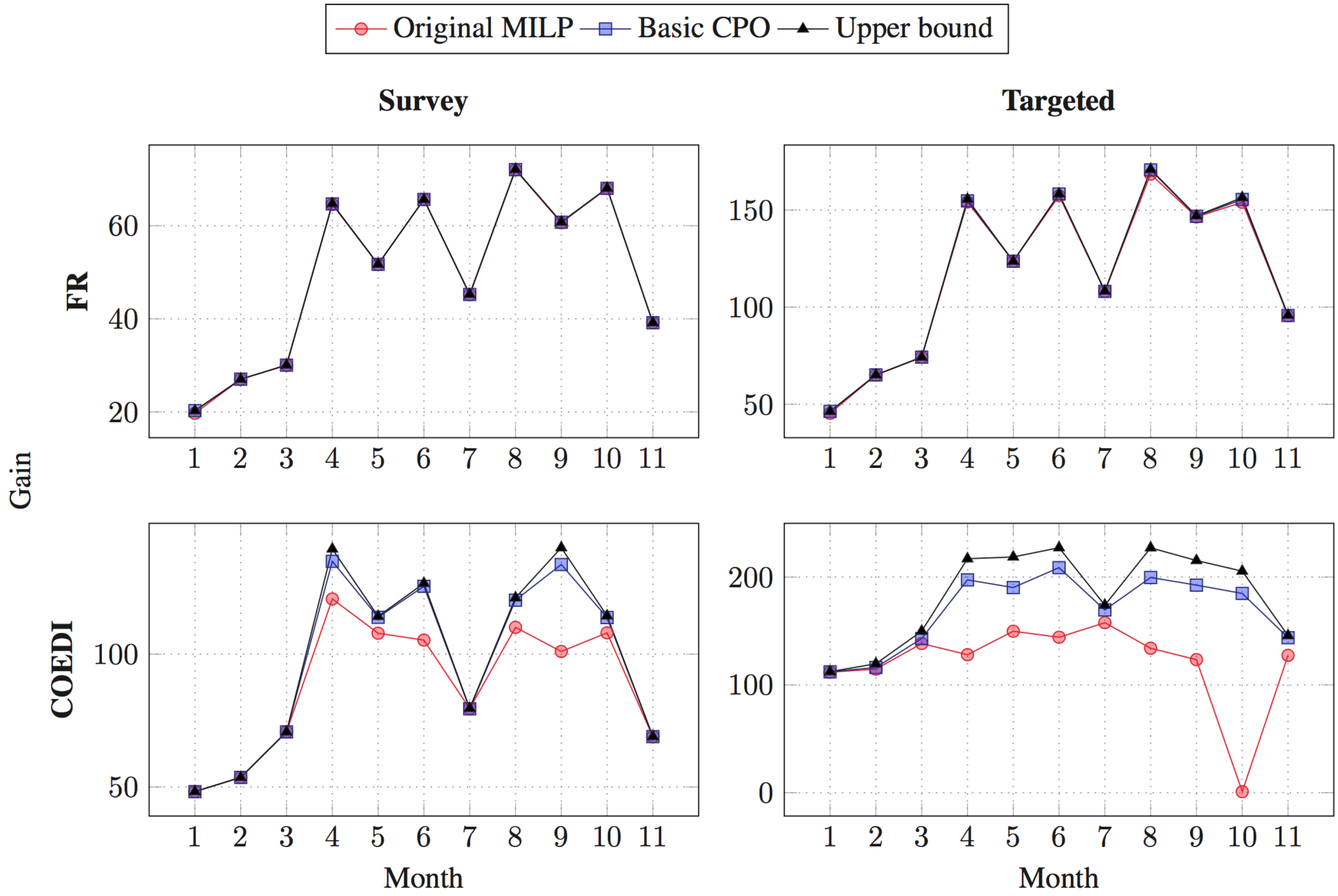
`s0[i,j]`: length is $>S_T$ and it does not bring any value



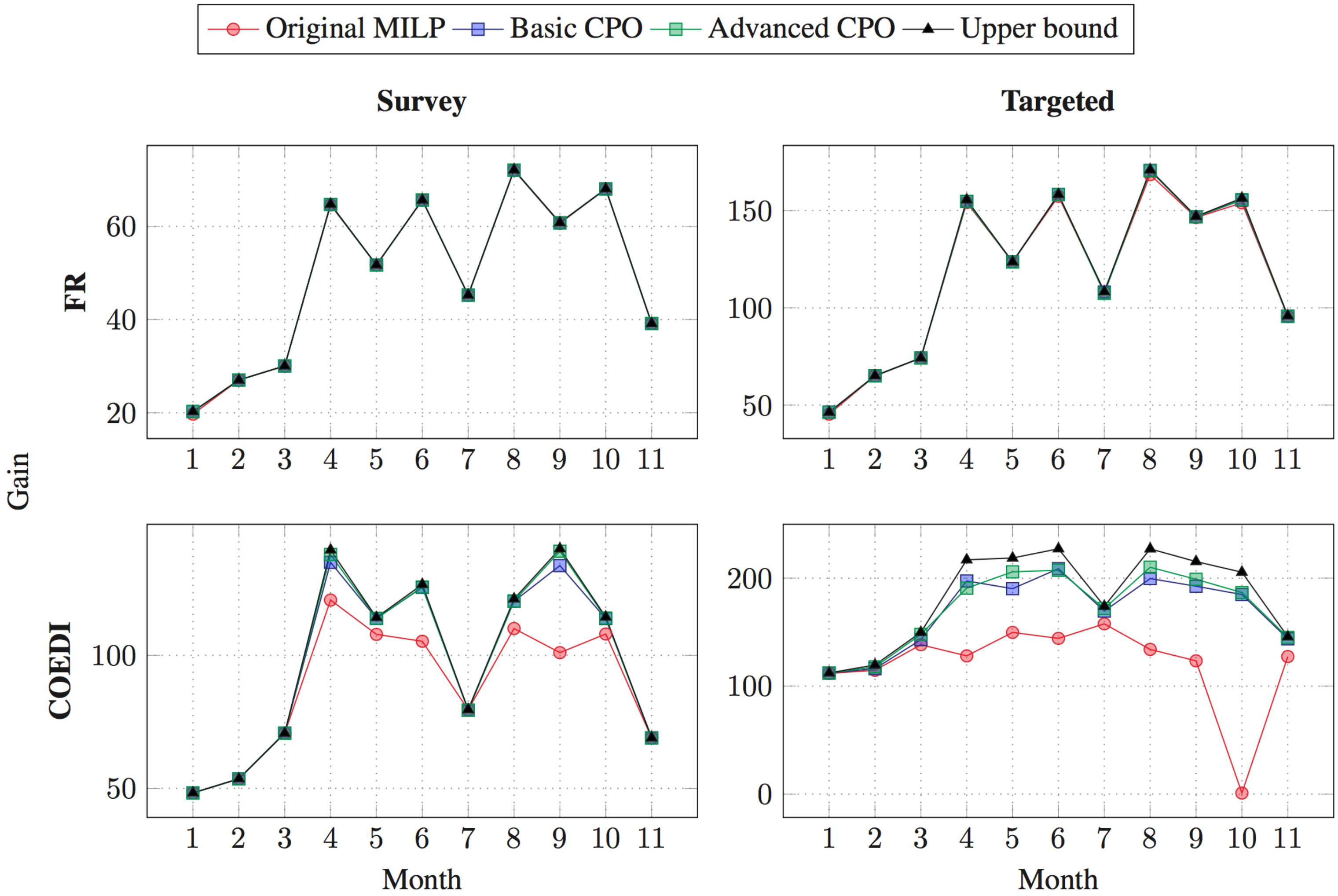
Good quality solutions (CP Optimizer model)

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Good quality solutions (CP Optimizer model)



Good quality solutions (CP Optimizer model)



Results and conclusion

- Optimal or near optimal solutions (gap≤1%) found for about 80% of the instances
- On COEDI instrument, average gap is 2.72% compared to 18.42% for the original MILP
- The model is easy to extend to additional features:
  - Different observation durations
  - Different observation modes
  - Sequence-dependent setup times between observations
  - Re-scheduling
- A good illustration of the versatility of CP Optimizer for solving challenging scheduling problems
  - More information in the ICAPS-2017 tutorial "Introduction to CP optimizer for Scheduling" material ...