Mixed Integer Programming for Change-Point detection

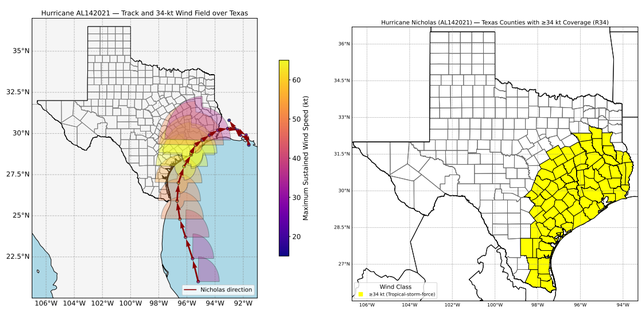
Apoorva Narula **Santanu S. Dey** **Yao Xie**
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Tech

1. Introduction

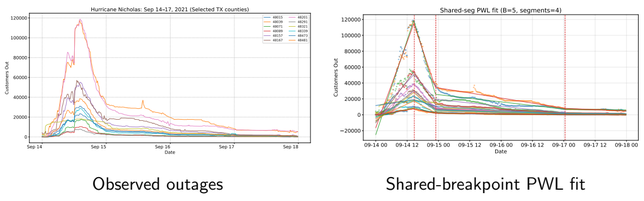
What is Change-point detection?
Goal: Locate domain points where the function shifts.

Mean change (Abrupt) Slope Change (Gradual)

Applications: Manufacturing Quality Control (detecting mean shifts), Aircraft Engine Monitoring, Power Systems Transformer Monitoring, and Power Outage Analysis (detecting slope shifts).
Illustrative example of extreme-weather outage analysis: Using hurricane trajectory (Hurricane Nicholas, September 2021) and county-level outage data (Texas).



Hurricane track 34-kt wind footprint



Observed outages Shared-breakpoint PWL fit

Interpretation: Detected change-points can indicate storm impact onset, rapid outage escalation, peak disruption, and recovery initiation.

2. Change-point Detection as Contiguous Domain Partitioning

Given observations

$$(x_t, y_t)_{t=1}^{T}, \quad x_t, y_t \in \mathbb{R}$$

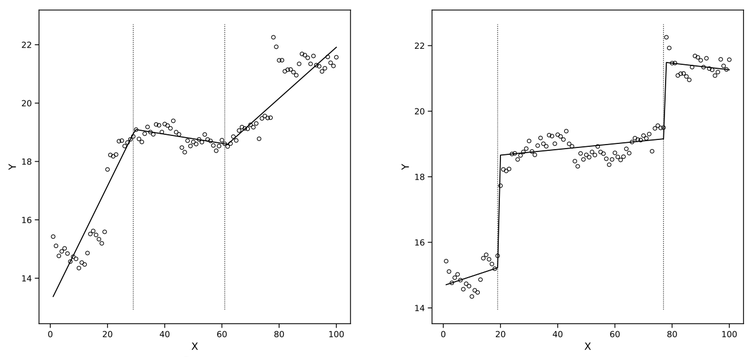
Partition domain $[r_0, r_K]$ into K contiguous segments such that a function piece is only active at a unique partition.

$$f(x) = \begin{cases} f_1(x|\theta_1), & x \in [r_0, \mathbf{r_1}], \\ f_2(x|\theta_2), & x \in (\mathbf{r_1}, \mathbf{r_2}], \\ \vdots \\ f_K(x|\theta_K), & x \in (\mathbf{r_{K-1}}, r_K], \end{cases}$$

Decision variables highlighted in red.

3. With or Without Continuity

With continuity **Without continuity**



Contiguous Segmentation Without Continuity: Polynomial Time - Solvable using Dynamic Programming ✓
With Continuity: NP-Hard - Solvable using MIP - Scalable?

4. MIP for Piecewise Linear Fitting

Objective:

$$\min \sum_{t=1}^{T} |y_t - \hat{y}_t|^q, \quad q \in \{1, 2\}$$

Subject to:

- **Contiguous Segmentation:** Binary variables $\delta_{j,t} \in \{0, 1\}$ indicate if point t is assigned to segment j.
- **Value fitting:**

$$\hat{y}_t = m_j x_t + c_j \quad \text{if } \delta_{j,t} = 1$$

- **Continuity:** Non-linear, non-convex constraints.

$$m_i r_i + c_i = m_{i+1} r_i + c_{i+1}, \quad \forall i$$

Linearized for piecewise linear fitting by Rebennack and Krasko (2020), *INFORMS Journal on Computing*.

5. Motivation and Contributions

Motivation: Making MIP-based contiguous segmentation more scalable.

Key Contributions: Tighter projection onto relaxed segment assignment variables $\delta_{j,t}$; Faster computation through experiments on stock price datasets from Yahoo Finance.

6. Benchmark Formulations

A. Segment Assignment (Basic): Goldberg et al. (2021), *Computational Optimization and Applications*. Break-point variables for contiguous segmentation:
A1. Assignment: Each observation assigned a segment.

$$\sum_{j=1}^{K} \delta_{j,t} = 1, \qquad \forall t = 1, \ldots, T \tag{1}$$

A2. Breakpoint Localization:

$$x_t \le r_j + M_t^{(1)}(1 - \delta_{j,t}) \tag{2}$$
$$x_t \ge r_{j-1} - M_t^{(2)}(1 - \delta_{j,t}) \tag{3}$$

If $\delta_{j,t}$=1, then $x_t \in [r_{j-1}, r_j]$; Otherwise, relaxed via big-M.

B. Segment Assignment (Alternate): Rebennack & Krasko (2020), *INFORMS Journal on Computing*: Contiguity **without breakpoint variables**.
B1. Assignment Constraints:

$$\sum_{j=1}^{K} \delta_{j,t} = 1, \qquad \forall t = 1, \ldots, T \tag{4}$$

B2. Contiguity Constraints:

$$\delta_{j+1,t+1} \le \delta_{j,t} + \delta_{j+1,t} \qquad \forall j, t \tag{5}$$
$$\delta_{1,t+1} \le \delta_{1,t} \qquad \forall t \tag{6}$$
$$\delta_{K,t+1} \ge \delta_{K,t} \qquad \forall t \tag{7}$$

Enforces contiguous segments, provably by strong induction.

7. Proposed Formulation

Narula, Dey & Xie (2026), *arXiv preprint*: Contiguous partitioning using **nested binary vectors**.
For each $j = 1, \ldots, K-1$, define

$$X_{j,t} \in \{0, 1\}, \qquad t = 1, \ldots, T.$$

- $X_{j,t} = 1$: point t belongs to one of the **first** j **segments**
- $X_{j,t} = 0$: point t belongs to segment $j+1$ or later

For example:

$$X_{j,\cdot} = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]$$

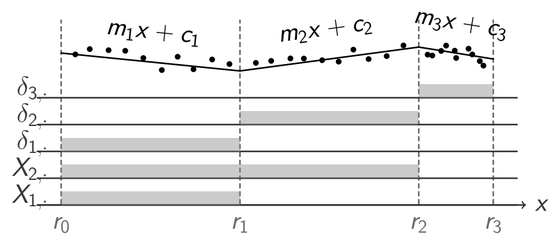
The above implies that the first 6 points lie in the first j segments, so the j-th breakpoint occurs between $t = 6$ and $t = 7$. To identify the points belonging to **segment** $j+1$, compare two consecutive nested vectors:

$$X_{j,\cdot} = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]$$
$$X_{j+1,\cdot} = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]$$

Their difference isolates the points that are in segment $j+1$:

$$\delta_{j+1,\cdot} = X_{j+1,\cdot} - X_{j,\cdot}$$
$$\delta_{j+1,\cdot} = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0]$$

Hence, the segment assignments can be recovered from differences of nested binary vectors, as discussed above.
This is also diagrammatically illustrated below:



The above diagram shows how nested binary vectors $X_{1,\cdot}$ and $X_{2,\cdot}$ determine the segment-assignment variables $\delta_{1,\cdot}, \delta_{2,\cdot}, \delta_{3,\cdot}$ in a continuous three-segment piecewise-linear fit.
Hence, the proposed extended formulation is:
C1. Monotonicity / Nesting Constraints:

$$\text{Monotonic in Time:} \quad X_{j,t} \ge X_{j,t+1} \qquad \forall j, t, \tag{8}$$
$$\text{Nested in Segments:} \quad X_{j+1,t} \ge X_{j,t} \qquad \forall j, t \tag{9}$$

C2. Recovering segment assignment:

$$\delta_{j,t} = \begin{cases} X_{1,t}, & j = 1, \\ X_{j,t} - X_{j-1,t}, & j = 2, \ldots, K-1, \\ 1 - X_{K-1,t}, & j = K. \end{cases}$$

8. Comparing LP Relaxations

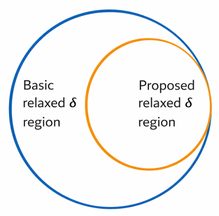
Prop. 1: Extended segment assignment polyhedron is integral.
Proof Sketch: Constraint matrix is **totally unimodular**:

- Identity block (from δ-definition), appended next to
- Difference constraints (at most two nonzeros: $+1, -1$)

Implication: Justifies lesser branch and bound effort.
Prop. 2: Alternate segment-assignment polyhedron is not integral. **Proof Sketch:** The formulation allows **fractional vertices** (as shown in the arXiv draft).

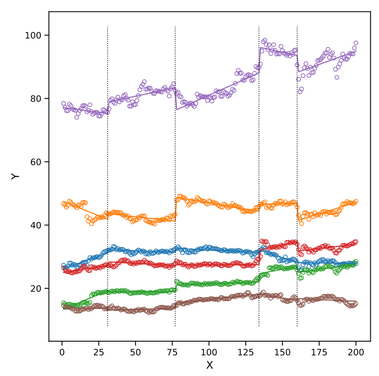
Prop. 3: Extended formulation yields **strictly tighter LP relaxation** than Basic formulation. **Proof Sketch:**

- Basic formulation allows highly fractional, oscillatory relaxed assignments. Proposed formulation enforces total variation bounds on these relaxed assignments.



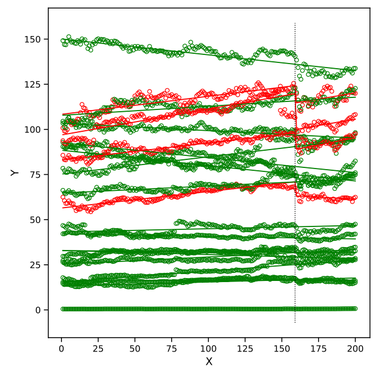
9. Multidimensional Extensions

Multidimensional Change-Point Detection



$T = 200$, $K = 5$, $D = 6$

Sparse Change-Point Detection



$D = 20$, fraction changed = 20%

Multidimensional: Fit D **piecewise functions** jointly with shared breakpoints, enabling detection of coordinated structural changes across multiple dimensions.

Sparse Change: Introduce binary indicators η_d to model whether dimension d changes, with sparsity enforced through

$$\sum_{d=1}^{D} \eta_d \le S.$$

10. Computational Comparison

Experimental Framework: We compare the **Basic**, **Alternative**, and proposed **Extended** MIP formulations on the following change-point detection tasks:

- Single-dimensional piecewise linear (PWL) fitting: with/without continuity.
- Multi-dimensional PWL fitting,
- Sparse change-point detection,

under both ℓ_1 and ℓ_2 loss functions. The datasets consist of **daily closing stock prices** downloaded from **Yahoo Finance** using Python-based API (`yfinance`). All formulations were solved using Gurobi 12.0.3 on Apple M3 processor. Performance was evaluated based on the number of instances where a formulation was fastest to converge to globally optimal solution, and total cumulative solver runtime across all instances.

Experiment	Setting	Norm	Basic	Alt.	Ext.	Total
Single-D PWL	No Cont.	ℓ_1	5	41	54	100
		ℓ_2	4	27	69	100
	Cont.	ℓ_1	25	8	67	100
		ℓ_2	4	36	60	100
Multi-D PWL		ℓ_1	8	46	46	100
		ℓ_2	17	40	43	100
Sparse CPD		ℓ_1	2	4	6	12
		ℓ_2	7	4	1	12
Overall			72	206	**346**	**624**

(Each entry = number of instances where the formulation was **fastest to reach optimality**.) **Key Takeaway:** Proposed formulations perform fastest overall.

Experiment	Setting	Norm	Basic	Alt.	Ext.
Single-D PWL	No Cont.	ℓ_1	3024	2633	3553
		ℓ_2	1046	661	566
	Cont.	ℓ_1	41452	35517	24352
		ℓ_2	48775	23840	21994
Multi-D PWL		ℓ_1	8112	8834	7760
		ℓ_2	3577	3436	3574
Sparse CPD		ℓ_1	3046	3883	3359
		ℓ_2	9561	10735	10380
Overall Total			118593	89538	**75538**

(Entries = **total solver time (seconds)** across all instances). **Key Takeaway:** Proposed formulation achieves the **lowest cumulative runtime**.