Combining Truncated Binary Search Tree and Direct Search for Flexible Piecewise Function Evaluation for Explicit MPC in Embedded Microcontrollers

By:

* Farhad Bayat

** Tor Arne Johansen

*** Ali Akbar Jalali

* University of Zanjan (ZNU), Zanjan, Iran.

** Norwegian University of Science & Technology (NTNU)

*** Iran University of Science & Technology (IUST)

Main Topics

- **Piecewise Function Evaluation Problem.**
- **Main Motivation:** *Explicit Model Predictive Control (eMPC) leads to piecewise affine state feedback solutions which may consist of 10000's affine function pieces. With fast evaluation, eMPC can be applied to mechatronic control systems.*
- **Approach 1:** *Orthogonal Truncated Binary Search Tree combined with direct search.*
- **Approach 2:** *Orthogonal Truncated Binary Search Tree combined with a simple approximation method to replace the direct search phase.*
- **Examples & Conclusions.**

Piecewise Function Evaluation Problem

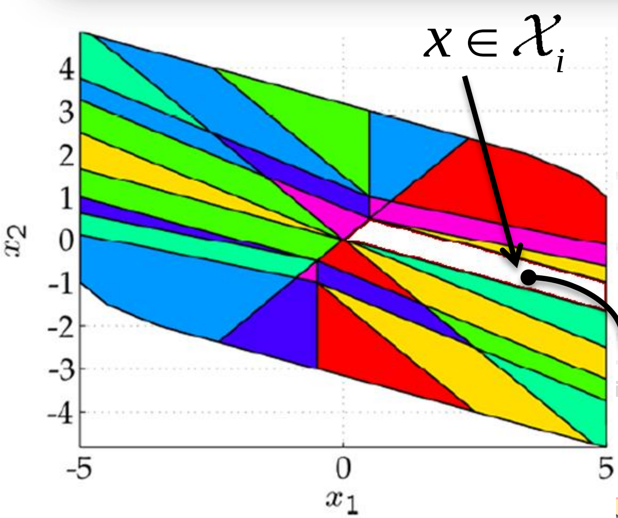
Definition:

Let a compact set $X \subset R^n$ *be partitioned into a set of* N_r *convex polyhedral regions* $\mathcal{X}_i,\ i = 1,\ldots,N_r$ *so that* $X = \bigcup_{i=1}^{N_r} \mathcal{X}_i$.

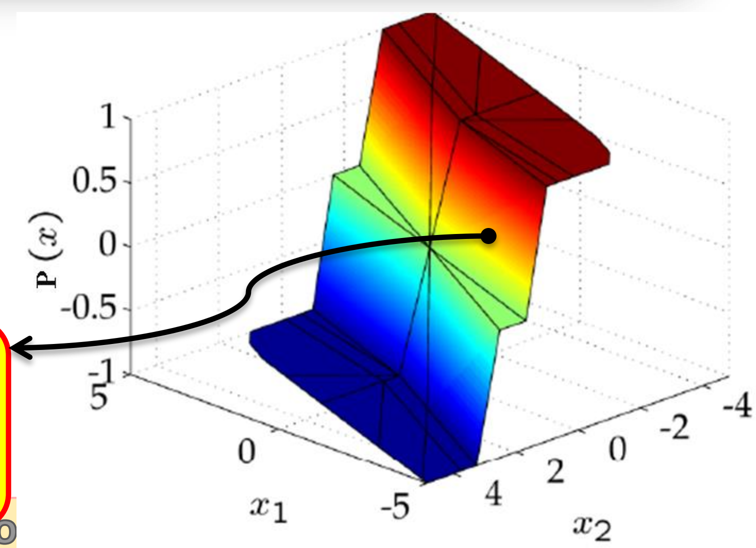
(i) *A piecewise function* $p(x): X \to R$ *is defined as* $p(x) = f(x \mid \phi_i) = f_i(x), \forall x \in \mathcal{X}_i$.

(ii) $p(x)$ *is said to be a Piecewise Affine (PWA) function if* $f_i(x) = [x^T\ 1]\phi_i$ *and continuous if* $f_i(x) = f_j(x), \forall x \in \mathcal{X}_i \cap \mathcal{X}_j$.

(iii) For a given query point $x \in X$, the PWA function evaluation problem is referred to find $p(x) = f_i(x),\ i \in \{1,\ldots,N_r\}$ for which $x \in \mathcal{X}_i$.



$$p(x) = \begin{bmatrix} x_1 & x_2 & 1 \end{bmatrix} \begin{bmatrix} -0.5 \\ -1.5 \\ 0 \end{bmatrix}$$



Main Motivation: Model Predictive Control

On-Line Optimization

$$J_N^*(x(0)) = \min_{u_0,\dots,u_{N-1}} \left\{ \sum_{k=0}^{N-1} \left(u_k^T R\, u_k + x_k^T Q\, x_k\right) + x_N^T P\, x_N \right\}$$

subj. to

$$x_k \in \mathbb{X} \subseteq \mathbb{R}^n, \quad \forall k \in \{1,\dots,N\}$$

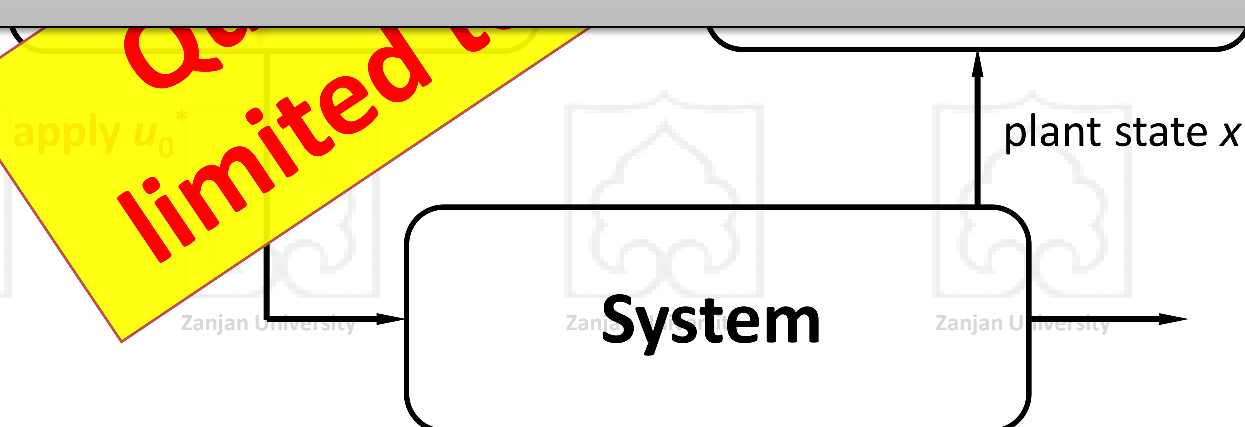
$$u_k \in \mathbb{U} \subseteq \mathbb{R}^m, \quad \forall k \in \{0,\dots,N-1\}$$

...mming is ...ses!!!

Qu... limited to...

Key Solution:

Multi-Parametric Quadratic Programming



Explicit Solution of MPC

$$
\begin{aligned}
J_N^*(x(\mathrm{t})) \;=\; & \min_{u_t,\ldots,u_{t+N-1}} \left\{ \sum_{k=0}^{N-1} (u_{t+k}^T R u_{t+k} + x_{t+k}^T Q x_{t+k}) + x_{t+N}^T P x_{t+N} \right\} \\
\text{subj. to} \quad & x_{t+k} \in \mathbb{X} \subseteq \mathbb{R}^n, \qquad \forall k \in \{1, \ldots, N\}, \\
& u_{t+k} \in \mathbb{U} \subseteq \mathbb{R}^m, \qquad \forall k \in \{0, \ldots, N-1\}, \\
& x_{t+k+1} = A x_{t+k} + B u_{t+k}, \;\; x_t = x(\mathrm{t}), \\
& Q = Q' \succeq 0, \;\; R = R' \succ 0, \;\; P \succeq 0
\end{aligned}
\qquad (1)
$$

substituting $x_{t+k} = A^k x(t) + \sum_{j=0}^{k-1} A^j B u_{t+k-1-j},$

$$
\begin{aligned}
J_N^*(x_t) = x_t' Y x_t + & \min_{U_N} \{ \frac{1}{2} U_N' H U_N + x_t' F x_t \} \\
\textit{subject to:} \quad & \\
& G U_N \leq W + E x_t \,, \\
& H \succ 0,
\end{aligned}
\qquad (2)
$$

(Bemporad et. al, 2002a):

***(H, F, Y , G, W, E)* are easily obtained from:**
***Q, R, P*, and (1).**

Optimization Variable: $U_N = [u_t^T, u_{t+1}^T, \ldots u_{t+N-1}^T]^T$

Multi-Parametric Quadratic Programming: Explicit Solution of MPC

(Bemporad et. al, 2002a):

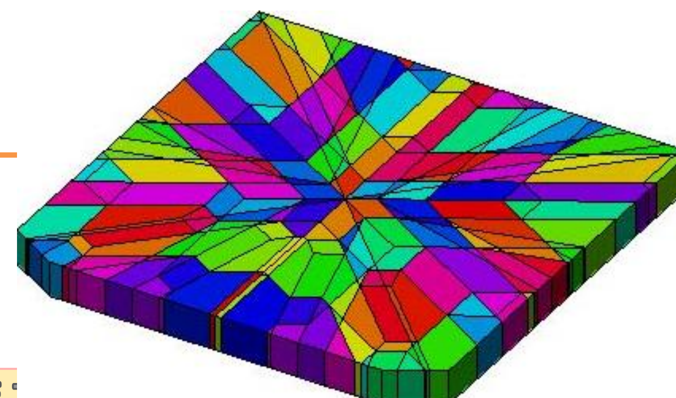
Theorem. *Consider the multi-parametric quadratic program* (2) *and let* **H**>0 *and* **X** *convex. Then the set of feasible parameters* $\mathbf{X}_f \subseteq \mathbf{X}$ *is convex, the optimizer* $z(x):\mathbf{X} \rightarrow \mathbf{R}^s$ *is continuous and piecewise affine.*

Corollary: *The control law* $u(t) = [I \;\; 0 \ldots 0]U_N^*$ *defined by the optimization problem (1) is continuous and piece-wise affine:*

$$u\big(x(t)\big) = F_i x\big(t\big) + G_i, \quad \forall x \in \mathcal{X}_i$$

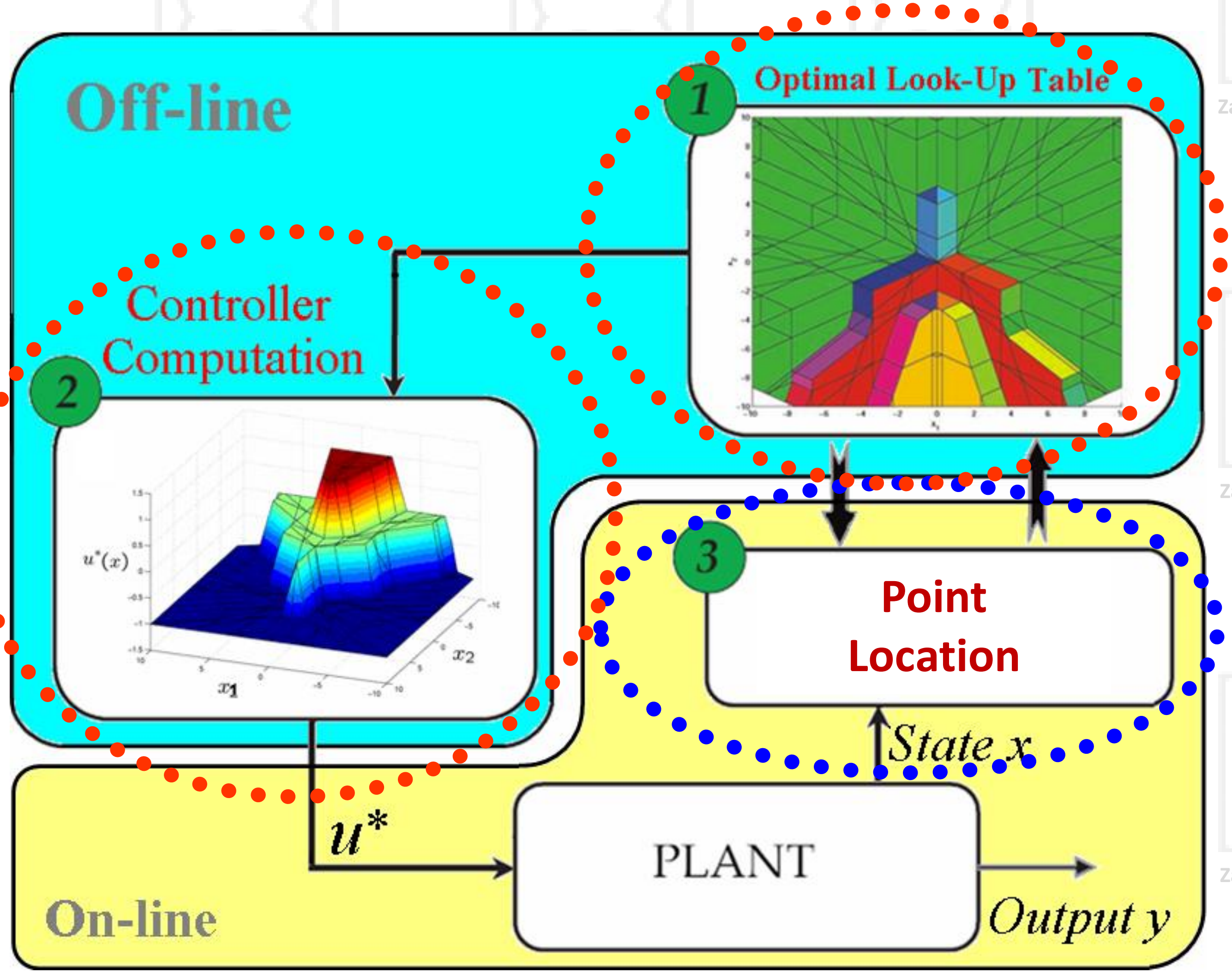
$$\mathcal{X}_i = \left\{x \in R^n \,|\, H_i x \le K_i\right\}$$

Switching between polyhedral regions occur when the optimal active set changes



Explicit Solution of MPC

Piecewise Affine Function Evaluation: Point location

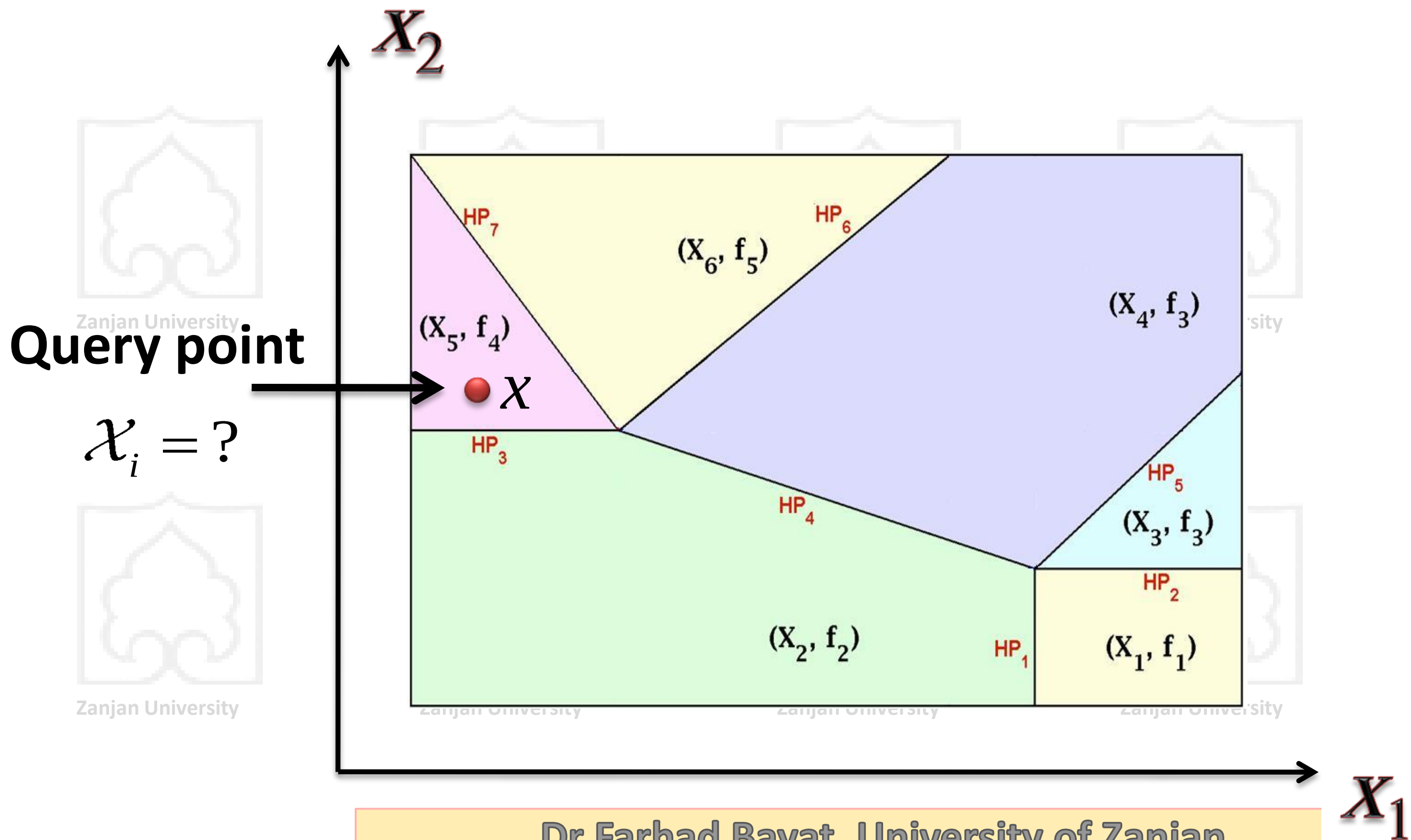


The point location problem

1- Binary Search Tree (BST)

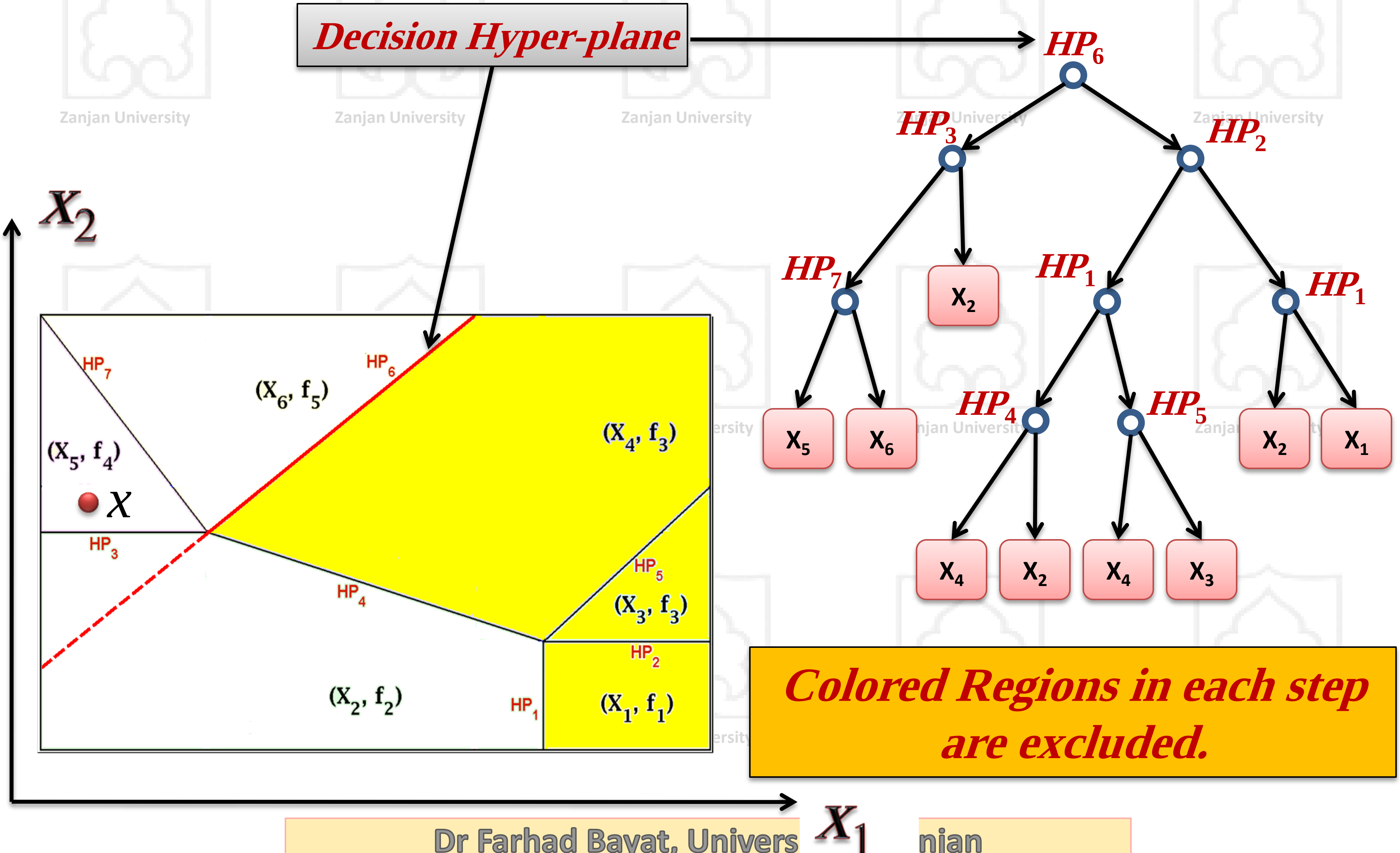
(Tøndel, Johansen and Bemporad, Automatica, 2003**)**

2- Orthogonal Truncated Binary Search Tree (OTBST)



Binary Search Tree

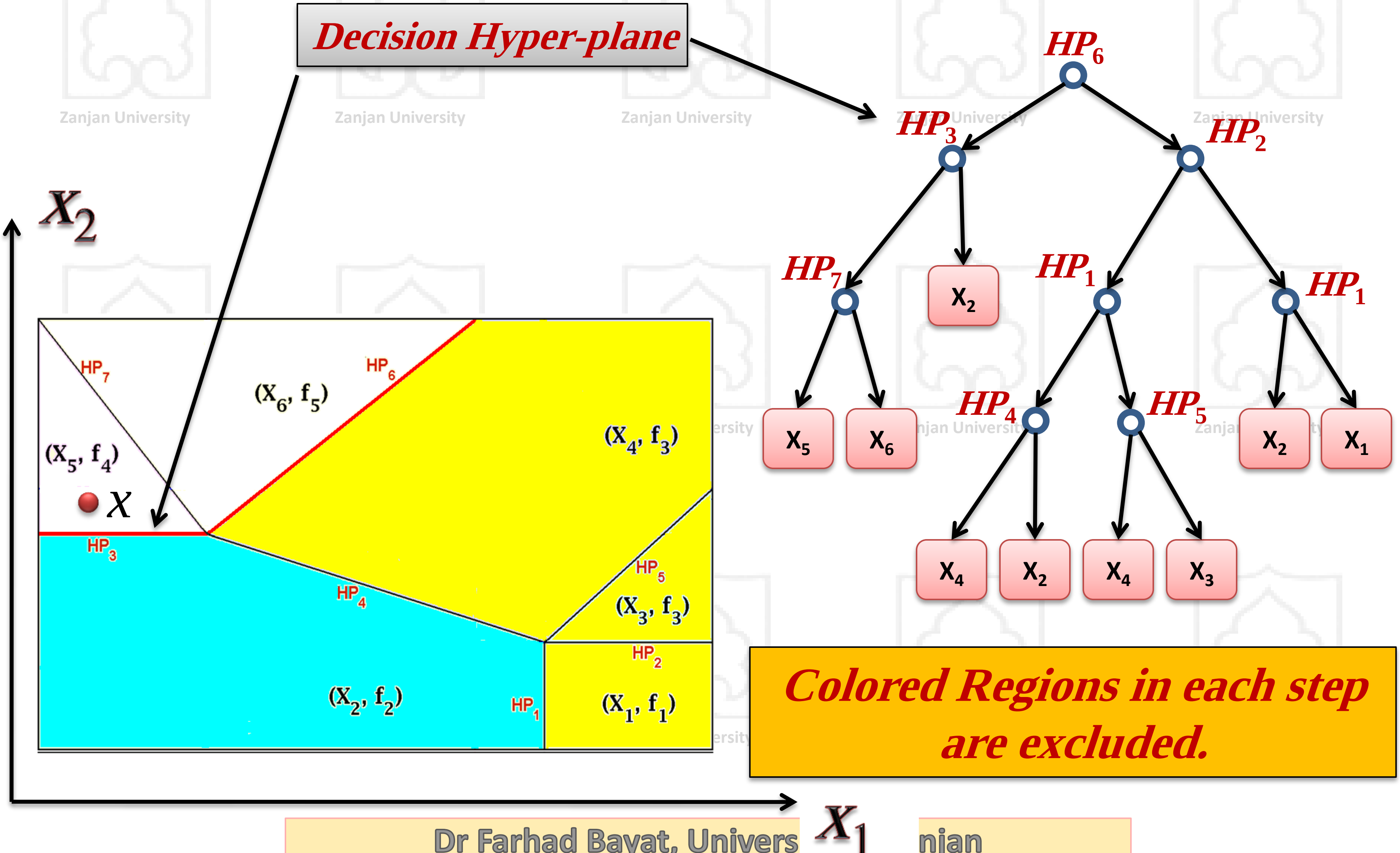
1- Binary Search Tree (BST)



Colored Regions in each step are excluded.

Binary Search Tree

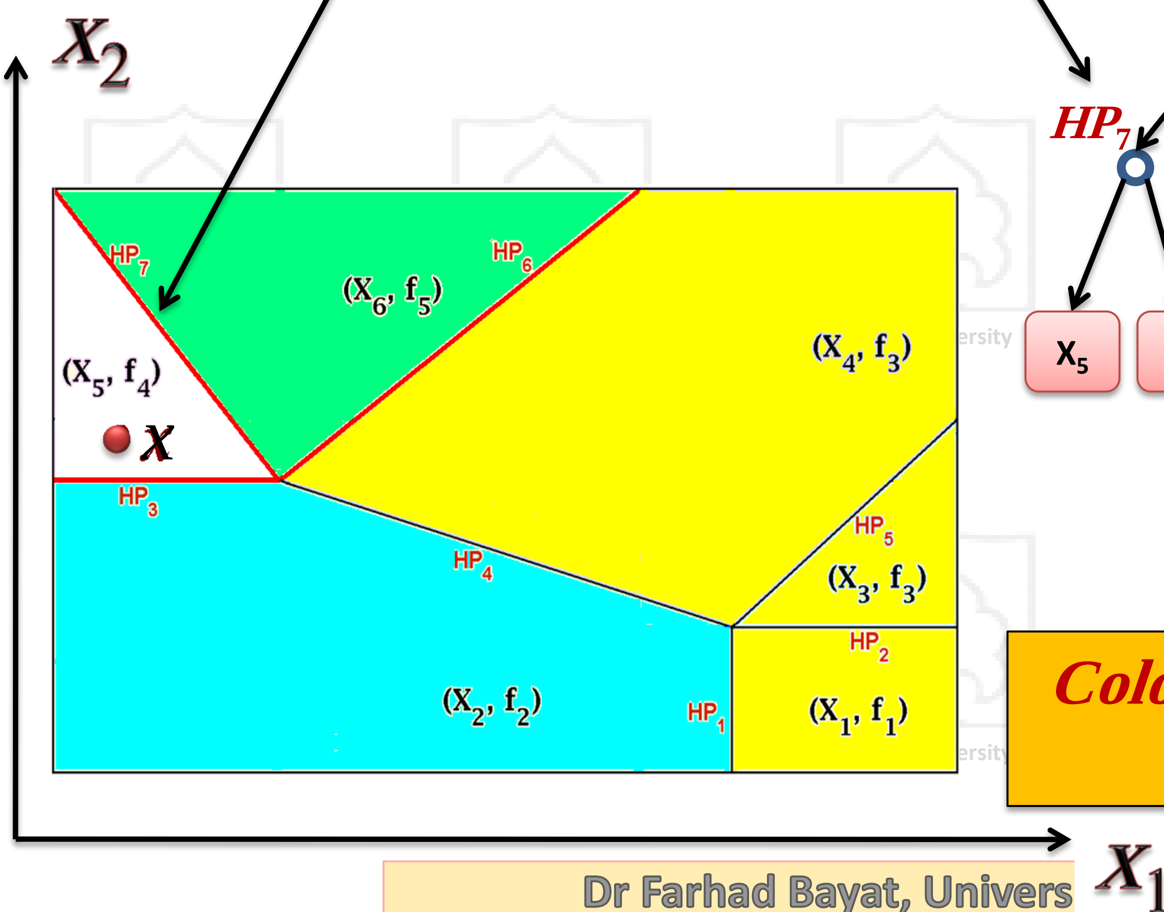
1- Binary Search Tree (BST)

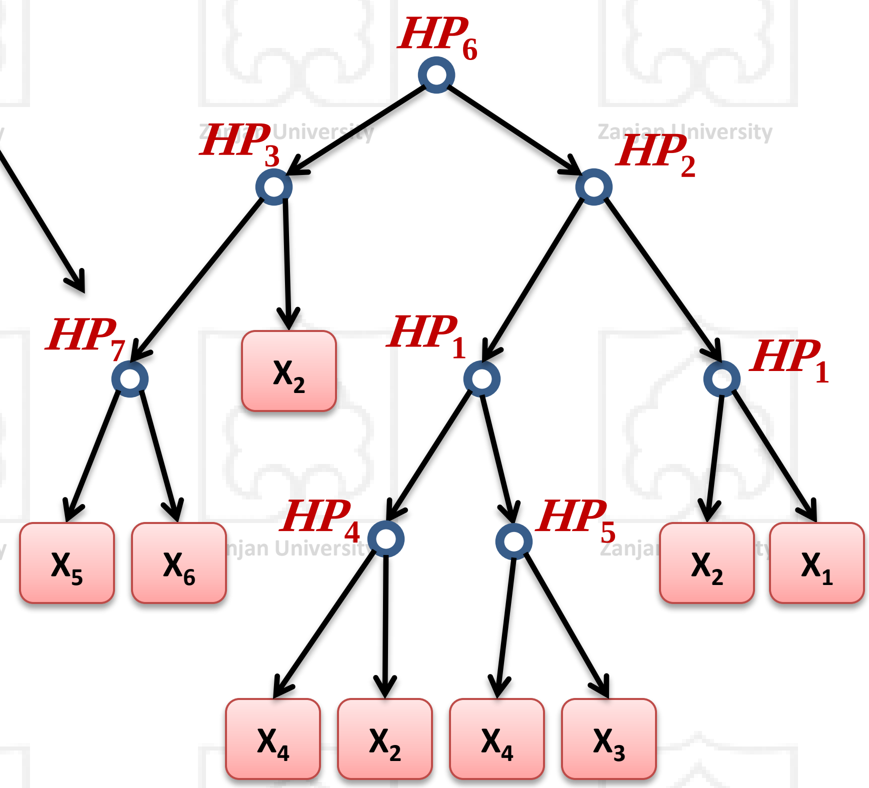


Colored Regions in each step are excluded.

1- Binary Search Tree (BST)

Decision Hyper-plane

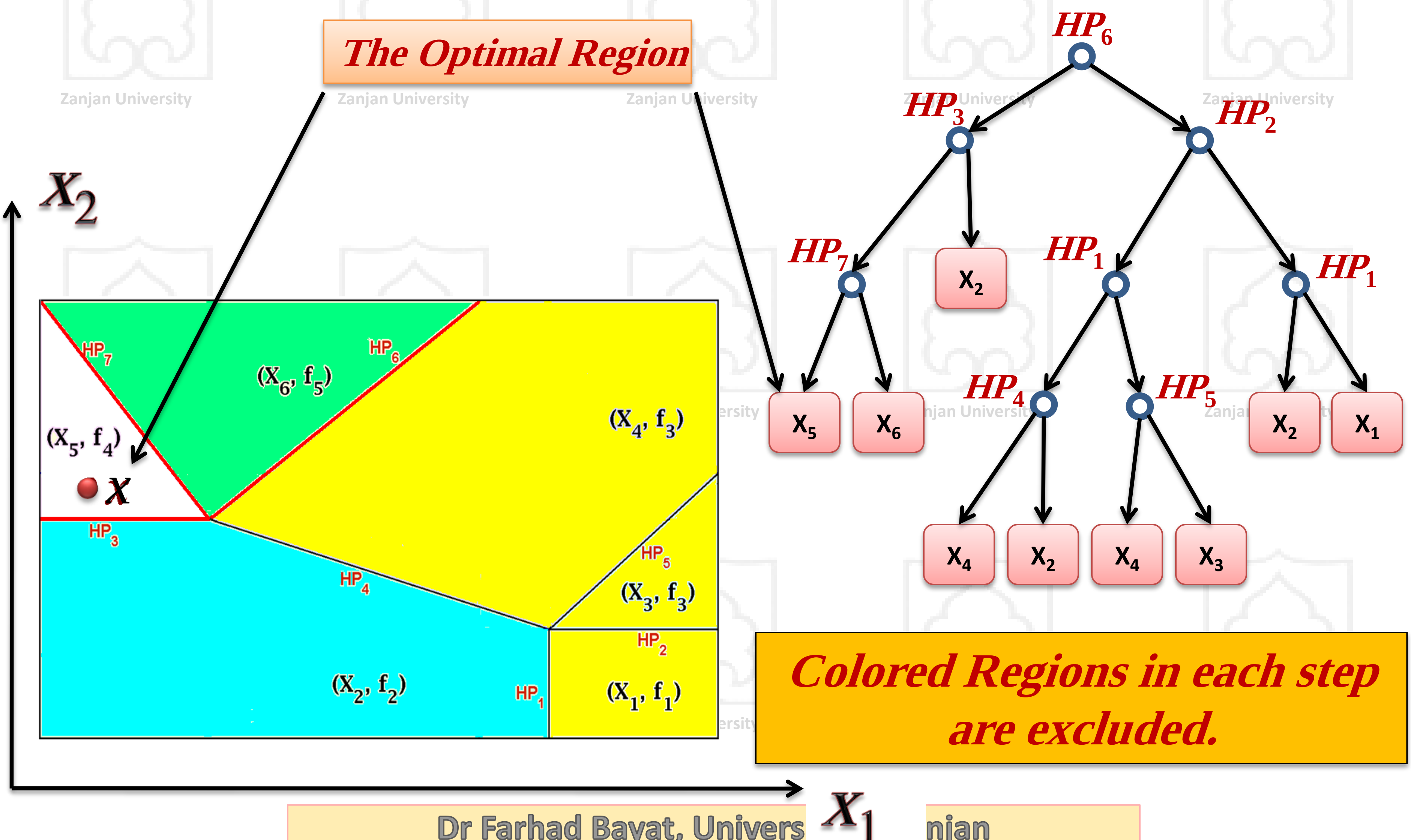




Colored Regions in each step are excluded.

Binary Search Tree

1- Binary Search Tree (BST)



Colored Regions in each step are excluded.

Binary Search Tree

1- Binary Search Tree (BST)

The BST method usually has very fast online processing,

BUT:

- ***For large number of regions it easily becomes prohibitive regarding Offline pre-processing time to construct a balanced (close to optimal) tree.***
- ***When the tree expands, the number of nodes increases exponentially (storage).***
- ***There is no flexibility to trade-off between the offline & online complexities, or the use of storage.***

Orthogonal Truncated Binary Search Tree

WHAT IS THE MAIN IDEA?

(i) Using **Orthogonal Decision Hyper-planes**, rather than the hyperplanes corresponding to the polyhedral regions.

(ii) Truncating the Orthogonal Binary Search Tree and using **Direct Search** (DS) to find optimal solution.

Orthogonal Truncated Binary Search Tree (OTBST)

Orthogonal Truncated Binary Search Tree

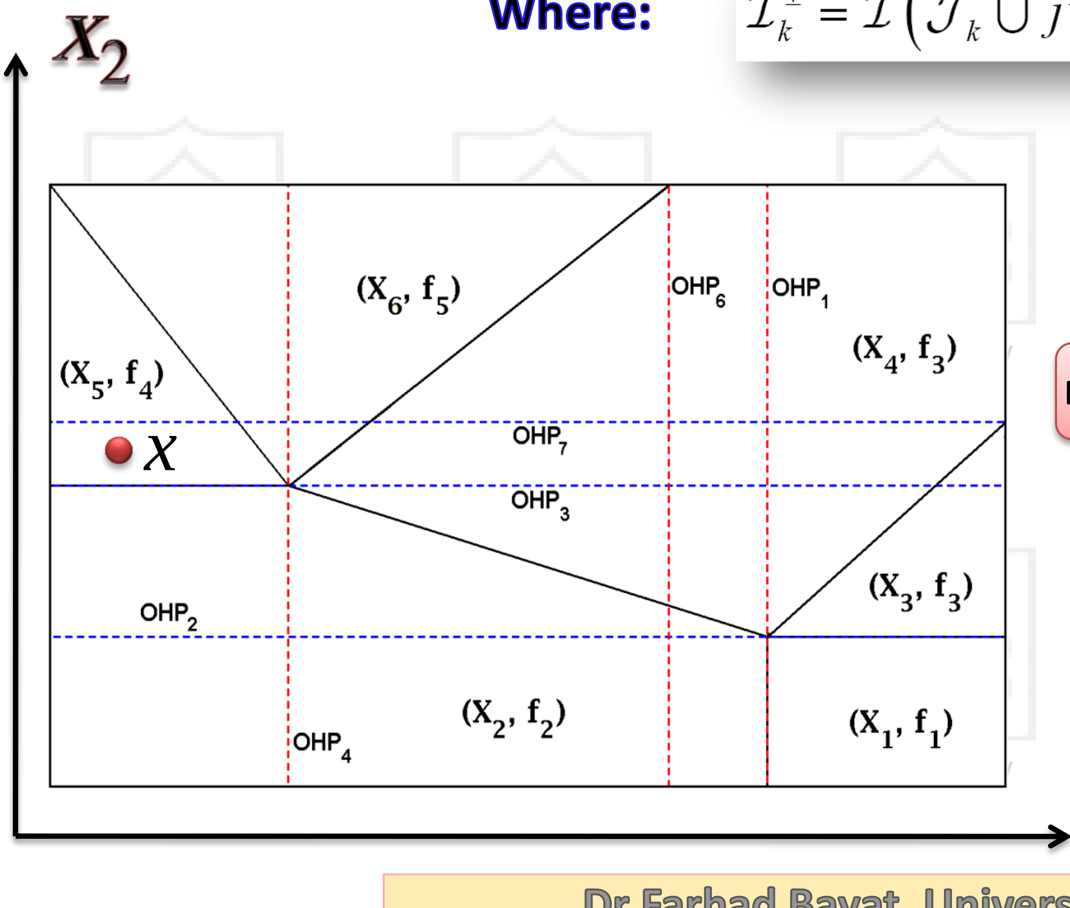
2- Orthogonal Truncated Binary Search Tree (OTBST)

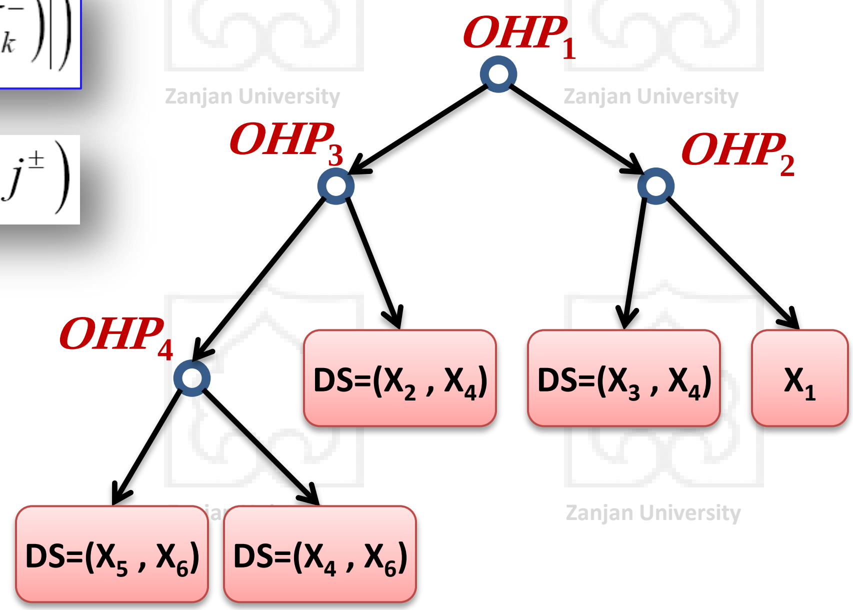
Decision OHP:

$$j_k = \arg\min_j \max\left(\left|\mathcal{F}\left(\mathcal{I}_k^+\right)\right|, \left|\mathcal{F}\left(\mathcal{I}_k^-\right)\right|\right)$$

Where:

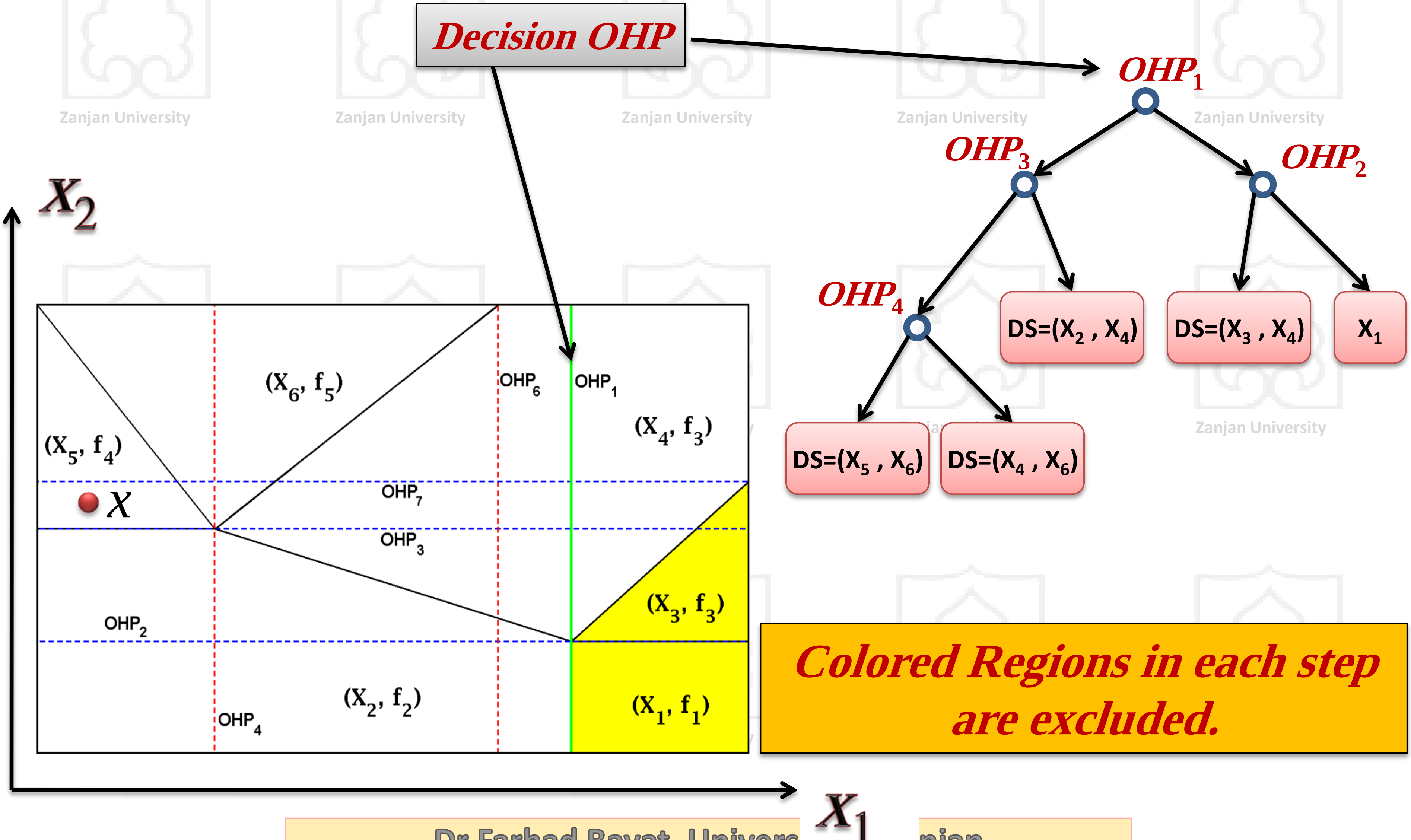
$$\mathcal{I}_k^{\pm} = \mathcal{I}\left(\mathcal{J}_k \cup j^{\pm}\right)$$





Orthogonal Truncated Binary Search Tree

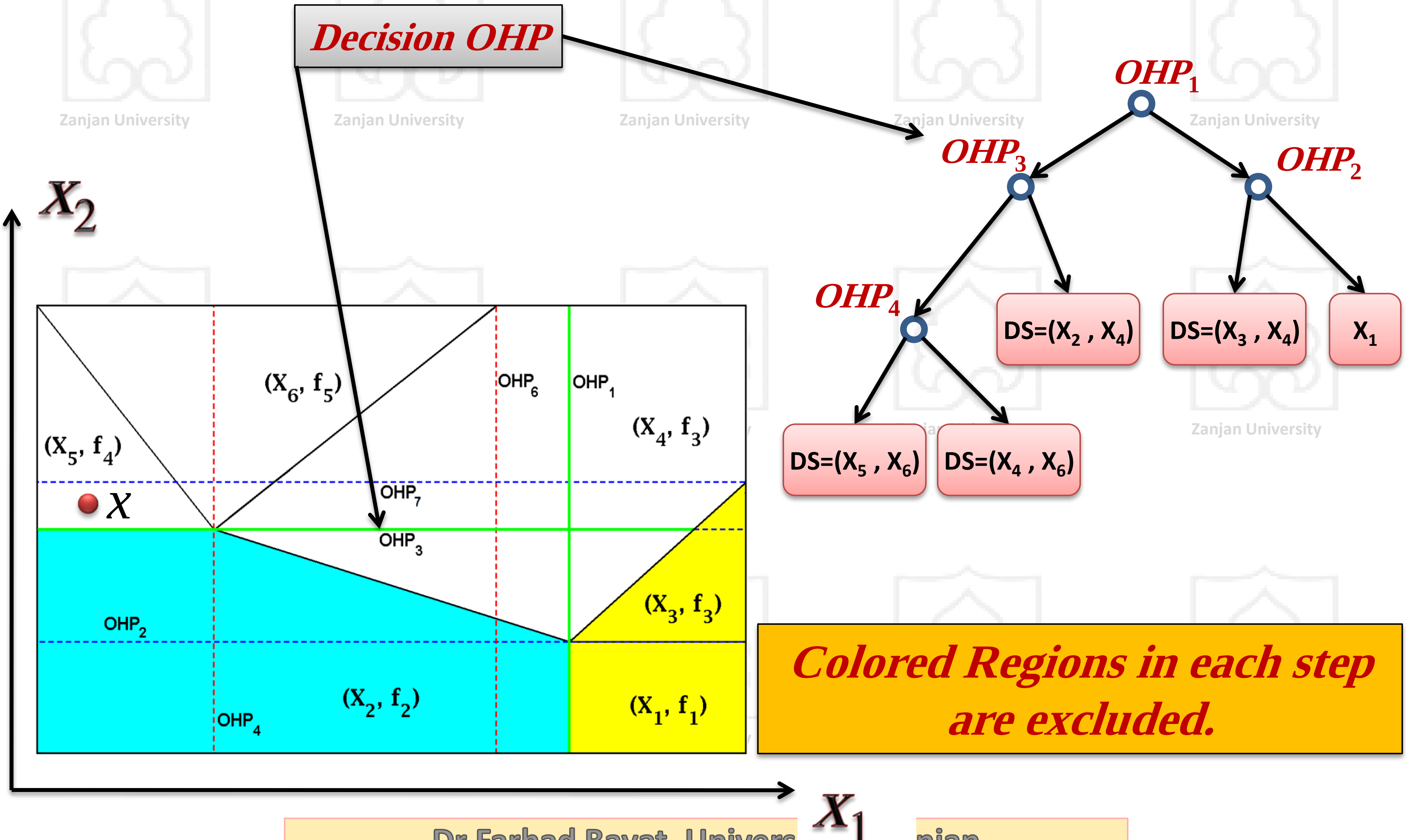
2- Orthogonal Truncated Binary Search Tree (OTBST)



Colored Regions in each step are excluded.

Orthogonal Truncated Binary Search Tree

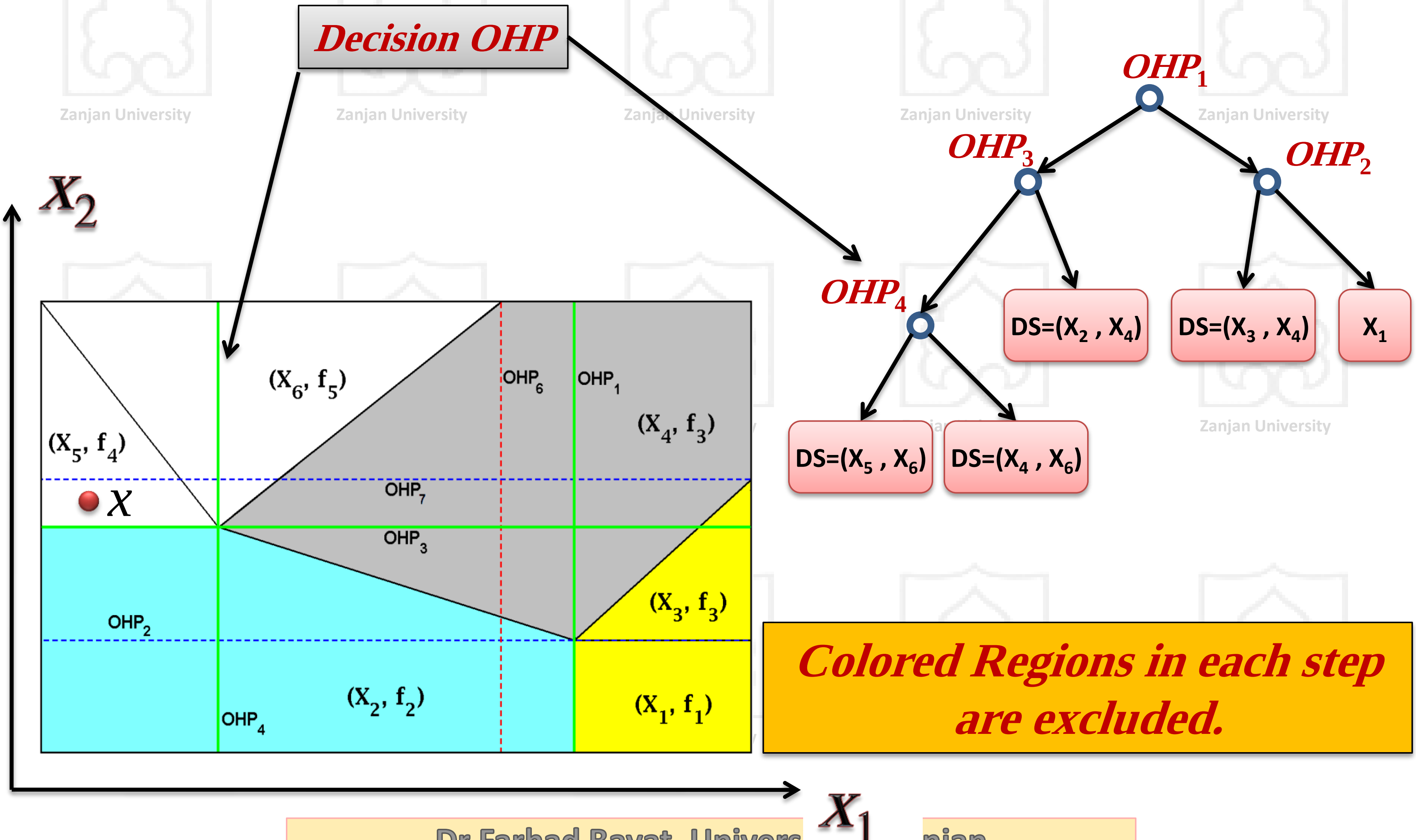
2- Orthogonal Truncated Binary Search Tree (OTBST)



Colored Regions in each step are excluded.

Orthogonal Truncated Binary Search Tree

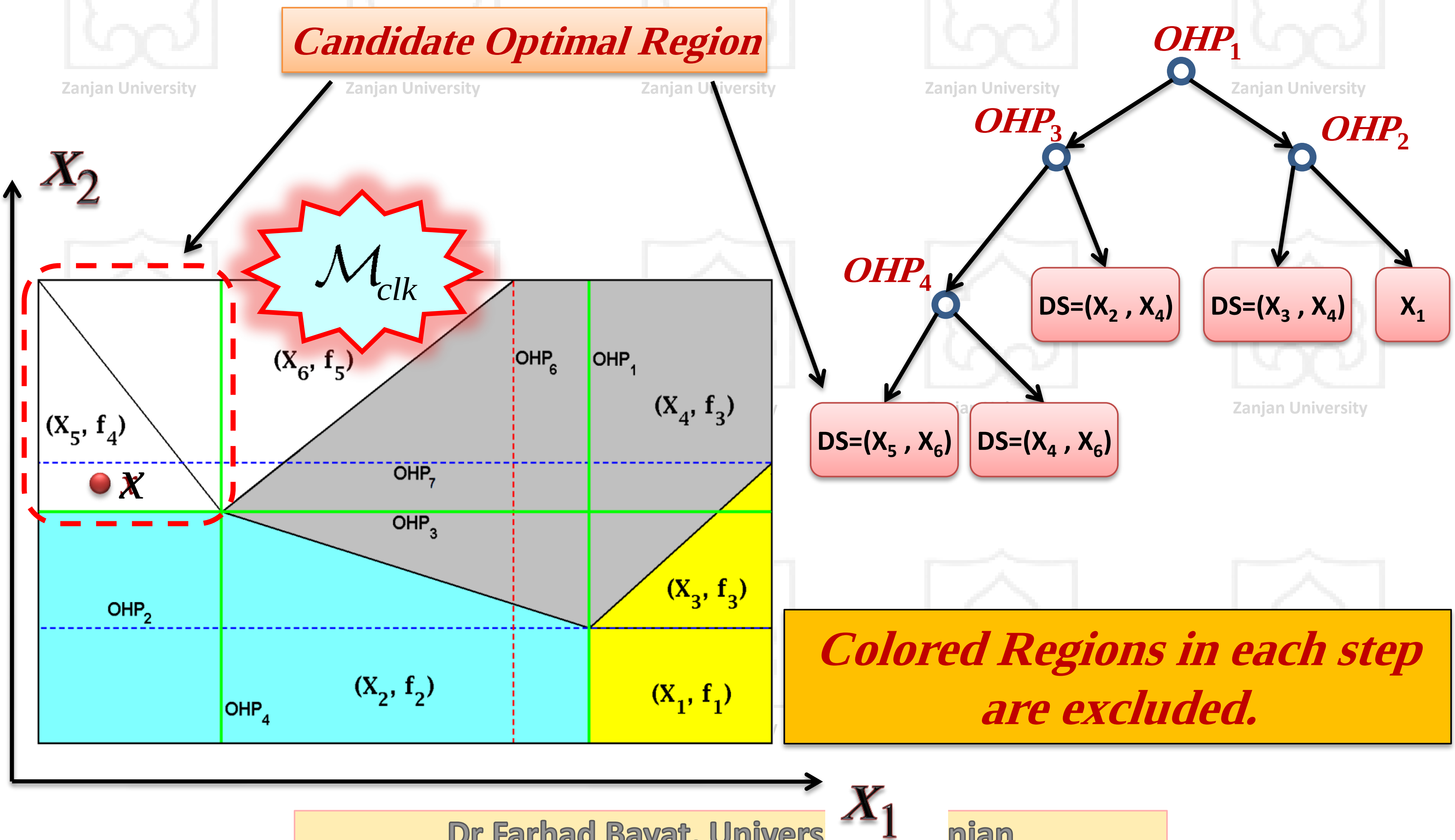
2- Orthogonal Truncated Binary Search Tree (OTBST)



Colored Regions in each step are excluded.

Orthogonal Truncated Binary Search Tree

2- Orthogonal Truncated Binary Search Tree (OTBST)



Colored Regions in each step are excluded.

Orthogonal Truncated Binary Search Tree

How much truncation of the Orthogonal BST?

A criterion is necessary …

Let $\mathcal{M}_{clk}$ be the **maximum admissible number of clock cycles** allocated for online piecewise function evaluation in the processor.

Accordingly,

we introduce $\mathcal{C}_k$ denoting the **number of clock cycles required** to compute the optimal region through the candidate optimal regions using direct search (or any other alternative).

Orthogonal Truncated Binary Search Tree

How much truncation of the Orthogonal BST?

A criterion is necessary …

$\mathcal{C}_k$ and $\mathcal{M}_{clk}$ are used as decision variables to decide if the Orthogonal Binary Search Tree should be truncated or not.

IF $\mathcal{C}_k \leq \mathcal{M}_{clk}$ **Then** → **Truncate node k.**

$$\boxed{\mathcal{C}_k = \mathcal{C}_k^{ST} + \mathcal{C}_k^{DS} = \mathcal{H}_k^{ST}\mathcal{M}_{OH} + \mathcal{H}_k^{DS}\mathcal{M}_H}$$

$$\boxed{\mathcal{M}_{OH} = C_{Comp} + 3C_{Mem} + C_{Branch}}$$

the number of clock cycles required for evaluation of each OHP.

Orthogonal Truncated Binary Search Tree

The main features of the proposed OTBST method, compared to the BST:

(i) **Fewer discriminating hyperplanes leads to less pre-processing time,**

(ii) **Simpler regions (hyper-rectangles rather than general polyhedra) gives further reduction in the pre-processing demands,**

(iii) **The OTBST approach enables the designer to trade-off between offline and online complexities, and**

(iv) **In the OTBST the online evaluation of each node in the search tree is more efficient since the hyper-planes are simpler (orthogonal)**

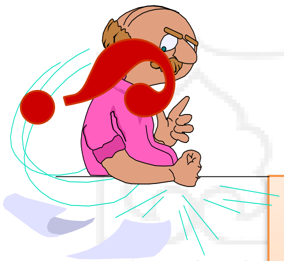
Approximate Orthogonal Truncated Binary Search Tree

Replacing Direct Search (DS) in the OTBST algorithm with an efficient alternative is of interest.

Approximating piece of PWA function defined over each truncated leaf

AOTBST

Approximate Orthogonal Truncated Binary Search Tree



Main idea:

Assuming eMPC applications with **input** and **soft-state (output) constraints** and **continuous solution,** with the price of sub-optimality it is possible to avoid storing the whole feasible partition by using a simple approach to approximate the PWA function piece in each **truncated leaf**.

NOTE:

If the piecewise control law is continuous then **relatively small regions** have in practice little influence on the closed-loop stability and performance of the overall system, when perturbing or approximating the control law. Also the existence of measured noise in practice is another factor which makes the relatively small regions less important.

Approximate Orthogonal Truncated Binary Search Tree

Algorithm: Approximate OTBST (AOTBST)

Offline Procedure

1. Let $\mathcal{S} = |\mathcal{I}|$ denotes the number of candidate regions, then compute $\boxed{\mathbf{R} = \{\mathcal{R}_1, ..., \mathcal{R}_\mathcal{S}\},}$ $\mathcal{R}_i = \mathcal{P}(\mathcal{J}) \cap \mathcal{X}_{\mathcal{I}(i)},\ i = 1, ..., \mathcal{S}$.
2. Compute and store the Chebyshev center and squared radius for all $\mathcal{R}_i$, i.e. $\mathbf{c} = \{c_1, ..., c_\mathcal{S}\}, \mathbf{r}^2 = \{r_1, ..., r_\mathcal{S}\}$, as presented in Bemporad et al. (2002b). The square is used to avoid online square root operation.
3. For $i \in \{1, ..., \mathcal{S}\}$, if $\mathbf{0}_{n_x \times 1} \in \mathcal{R}_i$ then set $c_i = \mathbf{0}_{n_x \times 1}$.

Algorithm: Approximate OTBST (AOTBST)

Online Procedure

1. Compute $\Delta x = \{dx_1, ..., dx_{\mathcal{S}}\}$, $dx_i = \|x - c_i\|_2^2$.
2. For $i \in \{1, ..., \mathcal{S}\}$, if $\boxed{dx_i \leq r_i}$ then $\underline{u(x) = f_{\mathcal{I}(i)}(x)}$, else
$$\underline{u(x) = \left(\sum_{i=1}^{\mathcal{S}} (r_i/dx_i)^{\alpha}\right)^{-1} \sum_{i=1}^{\mathcal{S}} \left((r_i/dx_i)^{\alpha} f_{\mathcal{I}(i)}(x)\right)},$$
where α is a positive integer tuning parameter, e.g. $\alpha = 1$ or 2.
3. Apply an appropriate saturation to preserve input constraints, i.e. $\boxed{\tilde{u}(x) = sat(u(x)).}$

Simulation Results

Example 1: **Double Integrator**

$$x(t+1) = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix} u(t) \quad T_s = 0.05s$$

System is subject to input constraints, $|u(t)| \le 1$, and output constraint $|y(t) = x_2| \le 1$.

The explicit solutions were obtained using the following parameters:

***p=2, Q=diag([1, 0]), R = 1,* and *Q_f= 0* and**

for different horizons *N=2, 4, 8 and 12*.

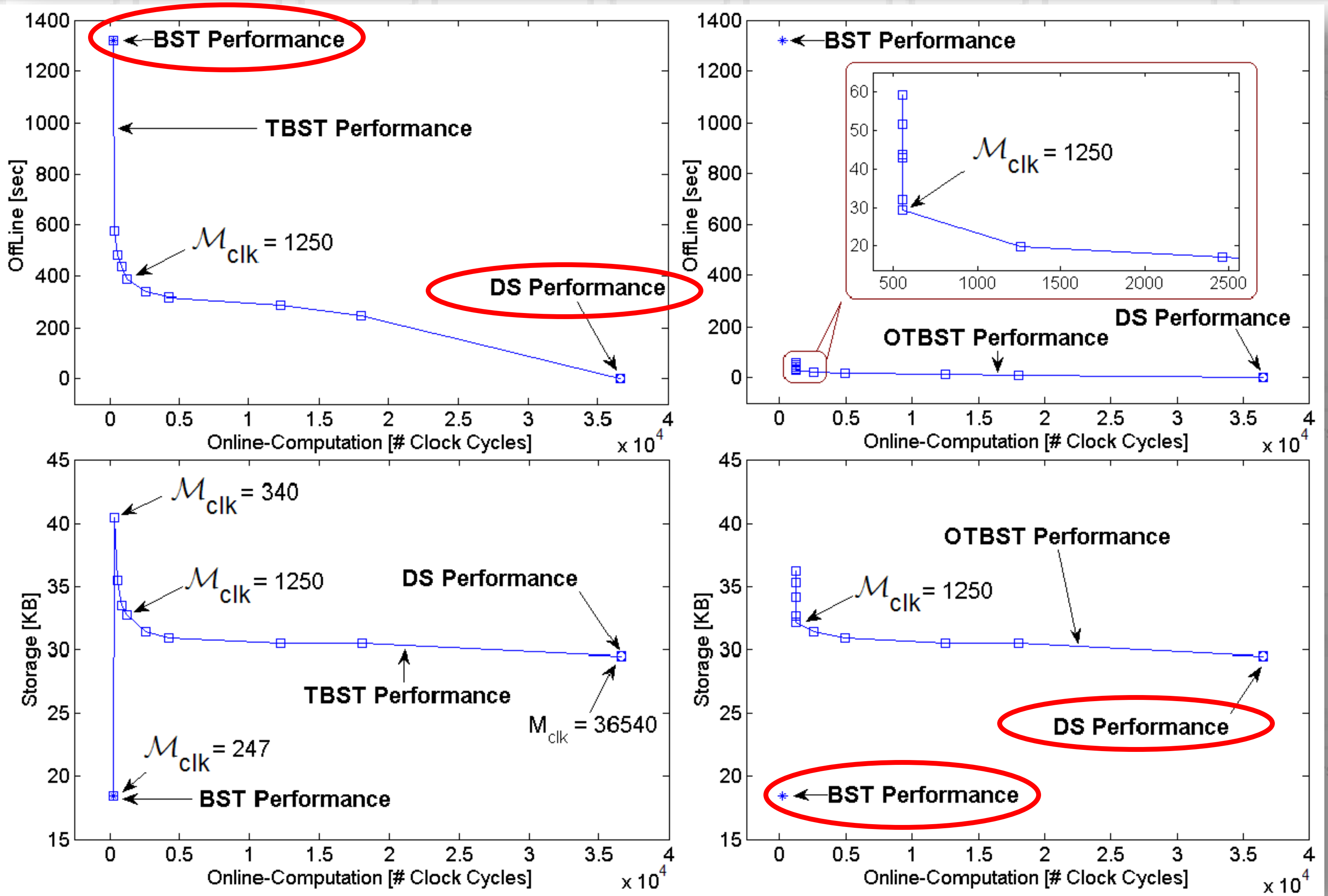
Example 1: Double Integrator

N	Method	N_r	$\mathcal{M}_{clk}$	N_n	Storage (Numbers)	Preprocessing Time(sec)	Online Clock-Cycles Min	Online Clock-Cycles Max
2	BST	83	—	188	752	27	147	187
	TBST		550	22	1096	11	107	513
	OTBST			33	1160	3	62	454
	AOTBST				862	3.5	62	242
4	BST	219	—	421	1792	160	147	207
	TBST		1000	33	2825	62	107	933
	OTBST			39	2890	6	73	952
	AOTBST				1779	7	73	474
8	BST	627	—	1177	4736	1321	187	247
	TBST		1250	84	8204	514	107	1227
	OTBST			90	8230	40	62	1165
	AOTBST				4070	42	62	619
12	BST	1325	—	2181	8704	5769	207	267
	TBST		1700	166	17202	2420	127	1655
	OTBST			131	17095	118	73	1531
	AOTBST				6794	121	73	793

NOTE: All simulations were done based on a low cost processor, e.g. AVR-XMEGA series, with 32 MHz clock frequency.

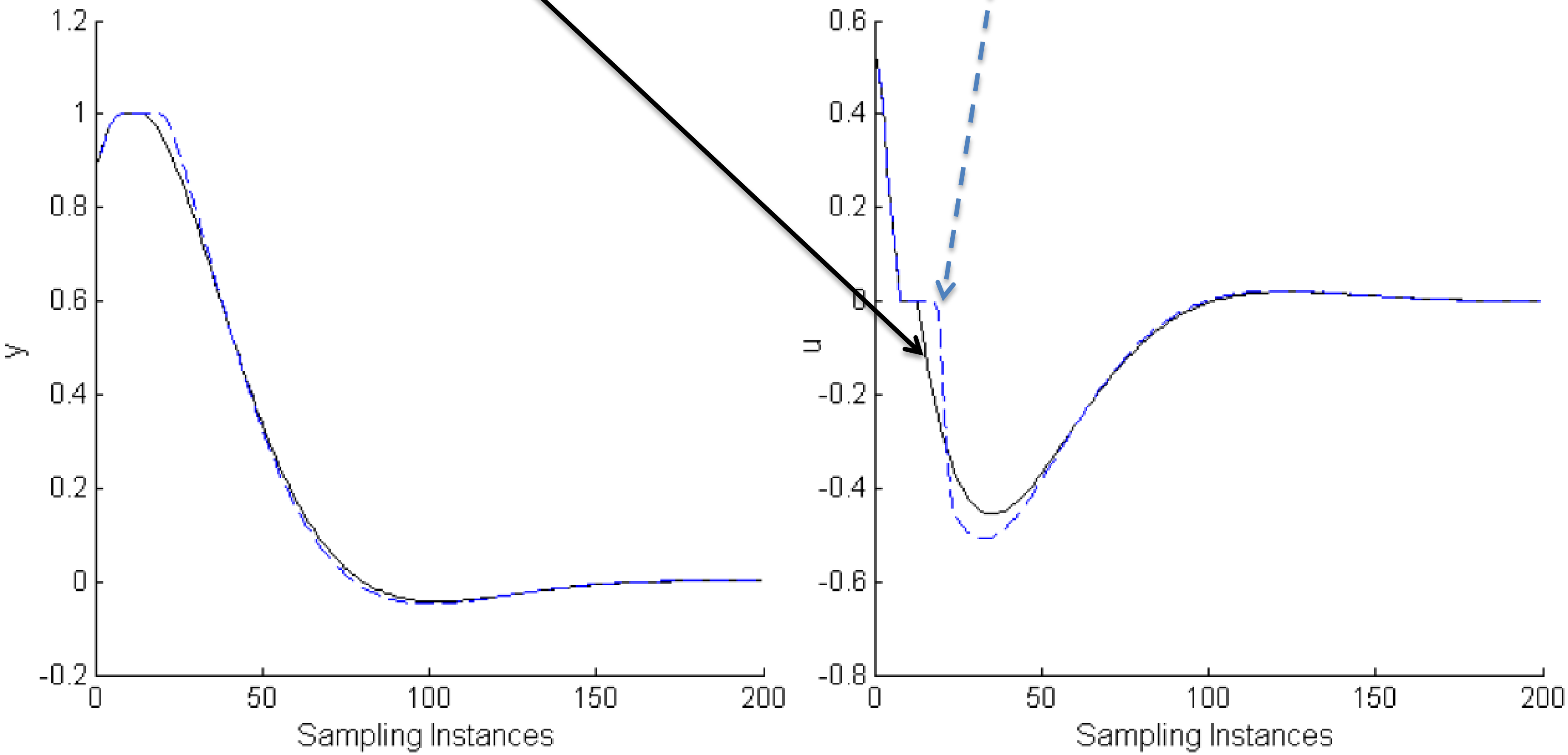
Example 1: Double Integrator, N=8

Trade-off characteristics of the TBST and OTBST methods, parameterizing the performance trade-off with the BST and DS approaches.



Example 1: Double Integrator, N=8

The evolutions of output and input signals,
exact solution (solid) and **approximation (dashed)**.



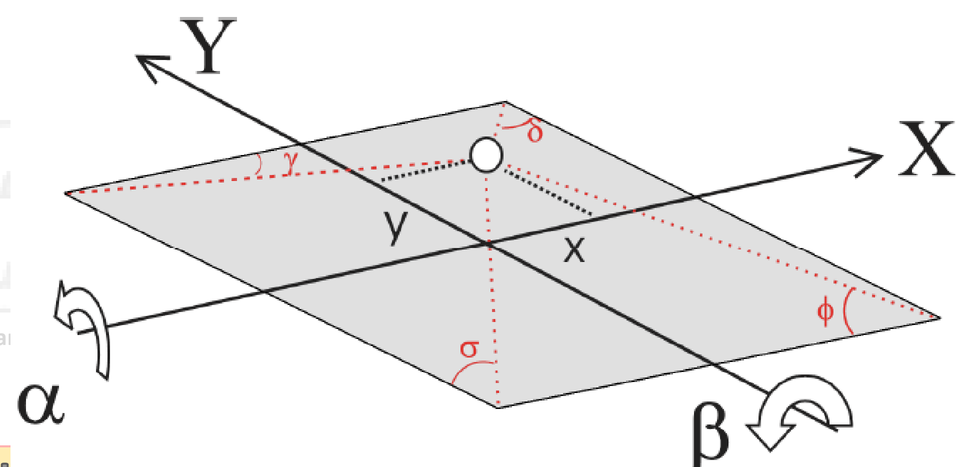
Example 2: Ball & Plate

$$
\text{Y-axis }(\alpha):\begin{cases}\dot{x}_\alpha(t)=\begin{bmatrix}0&1&0&0\\0&0&700&0\\0&0&0&1\\0&0&0&-34.69\end{bmatrix}x_\alpha(t)+\begin{bmatrix}0\\0\\0\\3.1119\end{bmatrix}u_\alpha(t)\\ y_\alpha(t)=x_\alpha(t)\end{cases}
$$

$$
\text{X-axis }(\beta):\begin{cases}\dot{x}_\beta(t)=\begin{bmatrix}0&1&0&0\\0&0&-700&0\\0&0&0&1\\0&0&0&33.18\end{bmatrix}x_\beta(t)+\begin{bmatrix}0\\0\\0\\3.7921\end{bmatrix}u_\beta(t)\\ y_\beta(t)=x_\beta(t)\end{cases}
$$

$$T_s = 0.03\,\text{sec}$$

$$
\begin{aligned}
&-30 \le y(t) \le 30,\\
&-15 \le \dot{y}(t) \le 15,\\
&-0.26 \le \alpha(t) \le 0.26,\\
&-1 \le \dot{\alpha}(t) \le 1,\\
&-10 \le u(t) \le 10,
\end{aligned}
$$



Example 2: Ball & Plate

$$T_s = 0.03\,\text{sec}$$

$$\min_{U=\{u_t,\ldots,u_{t+9}\}} \left\{ J\left(U, x(t)\right) = x_{t+10}^T P x_{t+10} + \sum_{k=0}^{9} \left(x_{t+k}^T Q x_{t+k} + u_{t+k}^T R u_{t+k} \right) \right\}$$

$$\begin{aligned} subj.\ to \quad & x_\alpha^{\min} \le x_\alpha(t+k) \le x_\alpha^{\max}, \quad k = 1, \ldots, 10 \\ & u_\alpha^{\min} \le u_\alpha(t+k) \le u_\alpha^{\max}, \quad k = 0, 1, \ldots, 9 \\ & x_t = x_\alpha(t), \qquad u_t = u_\alpha(t), \end{aligned}$$

$$x_\alpha = [y, \dot{y}, \alpha, \dot{\alpha}]^T$$

$$-30 \le y(t) \le 30,$$

$$-15 \le \dot{y}(t) \le 15,$$

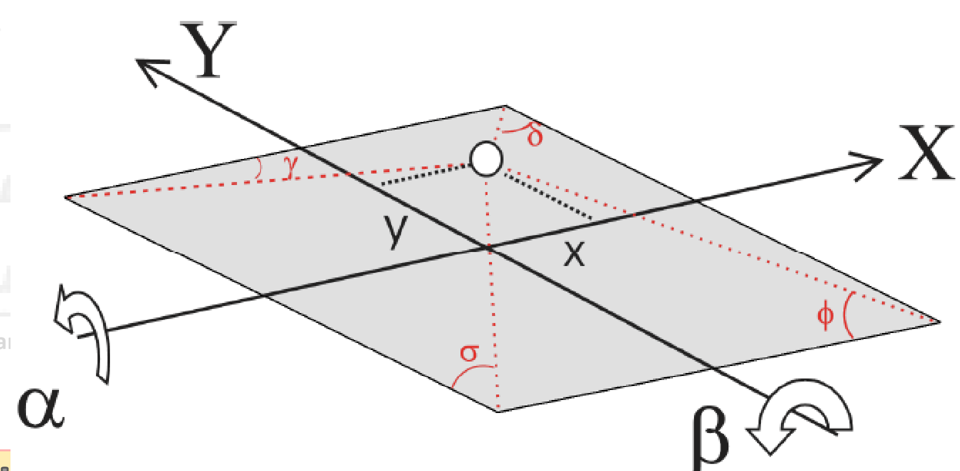
$$-0.26 \le \alpha(t) \le 0.26,$$

$$-1 \le \dot{\alpha}(t) \le 1,$$

$$-10 \le u(t) \le 10,$$

$$R = 1, \ \ Q = diag\{[6, 0.1, 500, 100]\}$$

$$P = Q$$



Example 2: Ball & Plate

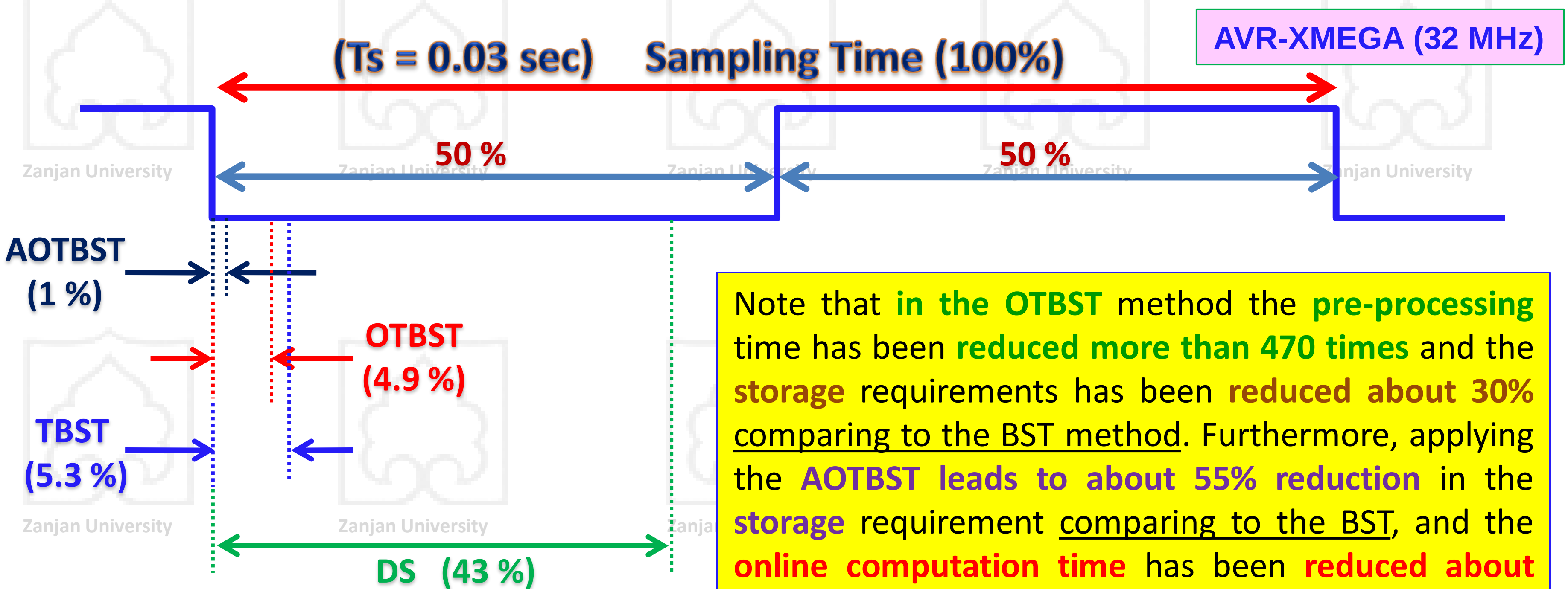
Method	N_r	$\mathcal{M}_{clk}$	N_n	Storage (Numbers)	Preprocessing Time(sec)	Online Clock-Cycles	
						Min	Max
BST	2024	—	23514	141084	343297	373	613
TBST		51e3	35	88136	11591	223	50911
OTBST			69	91196	724	68	47370
AOTBST				63868	917	68	9761

NOTE:

All simulations were done based on a low cost processor, e.g. AVR-XMEGA series, with 32 MHz clock frequency.

Example 2: Ball & Plate

Method	N_r	$\mathcal{M}_{clk}$	N_n	Storage (Numbers)	Preprocessing Time(sec)	Online Clock-Cycles Min	Online Clock-Cycles Max
BST	2024	—	23514	141084	343297	373	613
TBST		51e3	35	88136	11591	223	50911
OTBST			69	91196	724	68	47370
AOTBST				63868	917	68	9761



Note that **in the OTBST** method the **pre-processing** time has been **reduced more than 470 times** and the **storage** requirements has been **reduced about 30%** comparing to the BST method. Furthermore, applying the **AOTBST leads to about 55% reduction** in the **storage** requirement comparing to the BST, and the **online computation time** has been **reduced about 80%** comparing to the exact OTBST approach.

Conclusion

Efficient evaluation of piecewise functions defined over polyhedral partitions was addressed

+ The BST approach is modified by truncating (TBST) and then choosing decision hyper planes among a carefully selected set of axis-orthogonal hyper planes leading to significant reduction in pre-processing time (OTBST).

+ For continuous PWA functions, e.g. explicit MPC application, we proposed to replace the required direct search in the OTBST with a simple function approximation method (AOTBST) leading to significant reduction in the online processing time (with the cost of sub-optimality) comparing to the exact OTBST approach.

+ The AOTBST reduces extensively the storage complexity and offline computation time compared to the BST approach, while guaranteeing the required online computation time.

+ Enables the designer to trade-off between preprocessing time, storage requirement and online computation time.

+ OTBST is a global approach that can be applied to general piecewise nonlinear (PWNL) functions including discontinuous and overlapping.

Thank you for your attention