Recent Advances in Criterion Space Search Algorithms for Multi-objective Mixed Integer Programming

Martin Savelsbergh

Natashia Boland
Hadi Charkgard
Diego Pecin
Tyler Perini

Georgia Tech

Outline

- Motivation
- Basics
- Biojective Integer Programming
  - Balanced Box Method
- Biobjective Mixed Integer Programming
  - Boxed Line Method
  - Epsilon Tabu Constraint Method
  - Search-and-Remove Method
  - Computations
- Conclusions

Motivation

Application oriented:

- Increasing adoption of optimization-based decision support tools in industry and government (often embedding CPLEX, Gurobi, or Xpress-Optimiser)
- Most real-world problems involve multiple, often conflicting, goals

- A lack of multi-objective mixed integer programming solvers

Motivation (Cont.)

Research oriented:

- Availability of cheap computing power
- Availability of powerful single-objective mixed integer programming solvers

Multi-Objective Mixed Integer Programming

$$
\begin{aligned}
z_1(x) &= \min c_1 x + d_1 y \\
z_2(x) &= \min c_2 x + d_2 y \\
&\dots \\
z_k(x) &= \min c_k x + d_k y \\
& Ax + By \leq b \\
& x \in \{0,1\}^n, y \in \mathbb{R}^m
\end{aligned}
$$

$$
(x,y) \text{ efficient } \Leftrightarrow \nexists (x',y') : \begin{aligned}
z_1((x',y')) &\leqslant z_1((x,y)) \\
z_2((x',y')) &\leqslant z_2((x,y)) \\
&\dots \\
z_k((x',y')) &\leqslant z_k((x,y)) \\
z((x',y')) &\neq z((x,y))
\end{aligned}
$$

$$
(x,y) \text{ efficient } \Rightarrow z((x,y)) \text{ nondominated}
$$

Multi-Objective Mixed Integer Programming

- Solution Approaches
  - Decision space search methods
    - Purpose-built branch-and-bound algorithm
  - Criterion space search methods
    - Repeated solution of single-objective integer programs

Multi-Objective Mixed Integer Programming: Criterion Space Search Methods

- Pure integer programs:
  - Two objectives:
    - *Epsilon-constraint Method*
    - *Perpendicular Search Method, Augmented Weighted Tchebycheff Method, Balanced Box Method*
  - Three objectives:
    - *L-Shape Search Method, Quadrant Shrinking Method*
  - More than three objectives:
    - *Epsilon-constraint method, Enhanced Recursive Method, Full p-Split Method, Full (p-1)-Split Method*
- Mixed integer programs:
  - Two objectives:
    - *Triangle splitting method (2016), Epsilon Tabu Constraint Method (2016), Search-and-Remove Method (2017), Boxed-line Method (2017)*
  - Three objectives: ***none***
  - More than three objectives: ***none***

Biobjective Integer Program (BOIP)

Define a BOIP as:

$$\begin{aligned} \min \quad & \vec{z}(x) := [z_1(x), z_2(x)] \\ \text{s.t.} \quad & x \in X \end{aligned}$$

where:

1. $X \subseteq \mathbb{Z}^n$
2. X is nonempty and bounded

Goal: Find the complete nondominated frontier (NDF) for any BOIP.

Biobjective Mixed Integer Program (BOMIP)

Define a BOMIP as:

$$\begin{aligned} \min \quad & \vec{z}(x) := [z_1(x), z_2(x)] \\ \text{s.t.} \quad & x \in X \end{aligned}$$

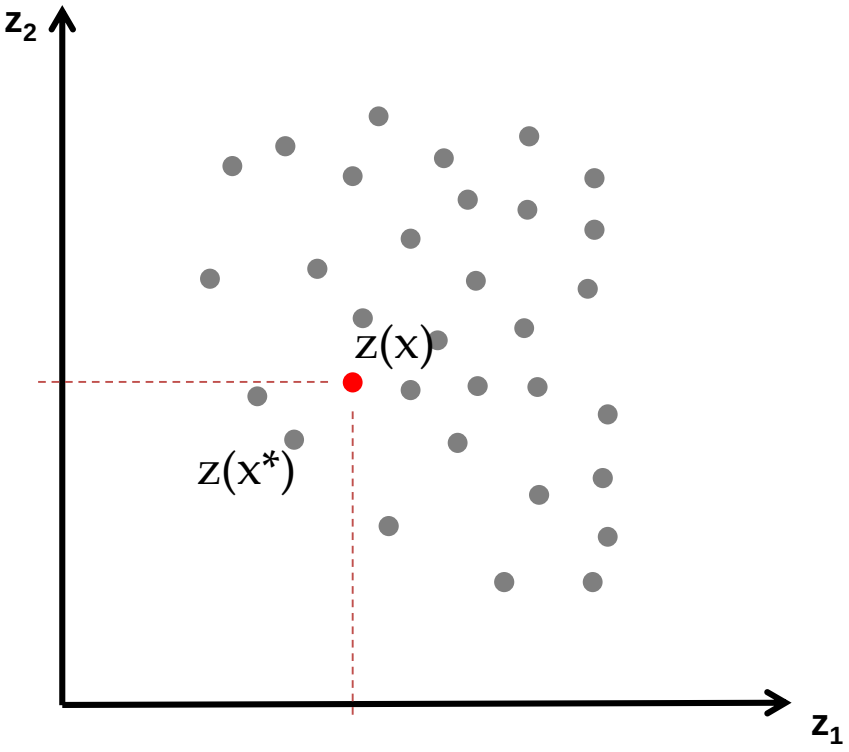
where:

1. $X \subseteq \mathbb{Z}^n \times \mathbb{R}^m$
2. X is nonempty and bounded

Goal: Approximate the full nondominated frontier (NDF) for any BOMILP.

Note: x decision vector of both binary and continuous variables

Nondominated Points

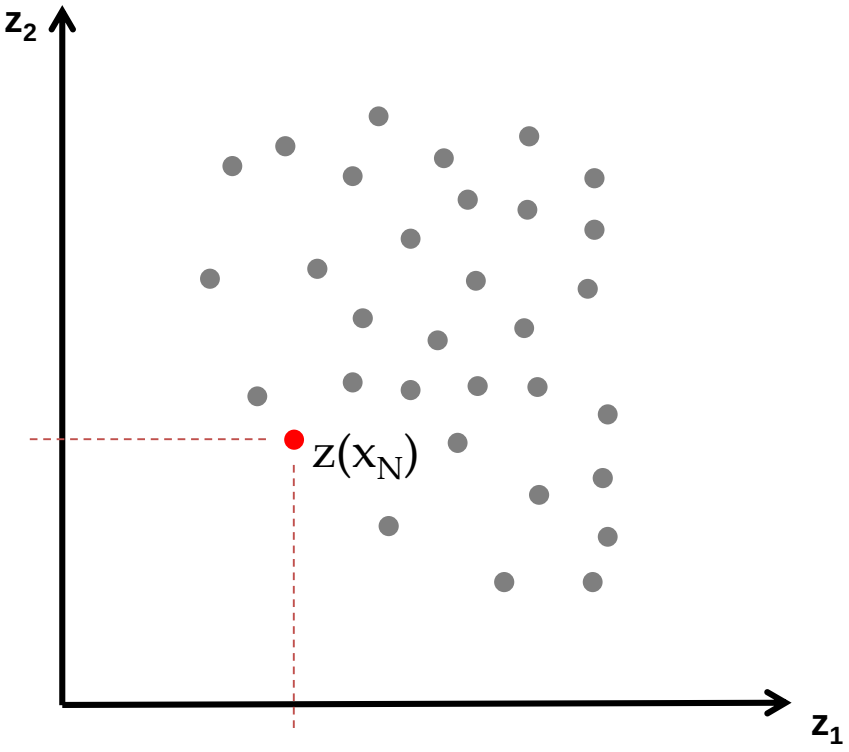


Definition:

Given feasible solutions x and x*, we say x* **dominates** x if

$$z(x) \neq z(x^*) \text{ and } z_i(x^*) \leq z_i(x) \ \forall i = 1, 2$$

Nondominated Points



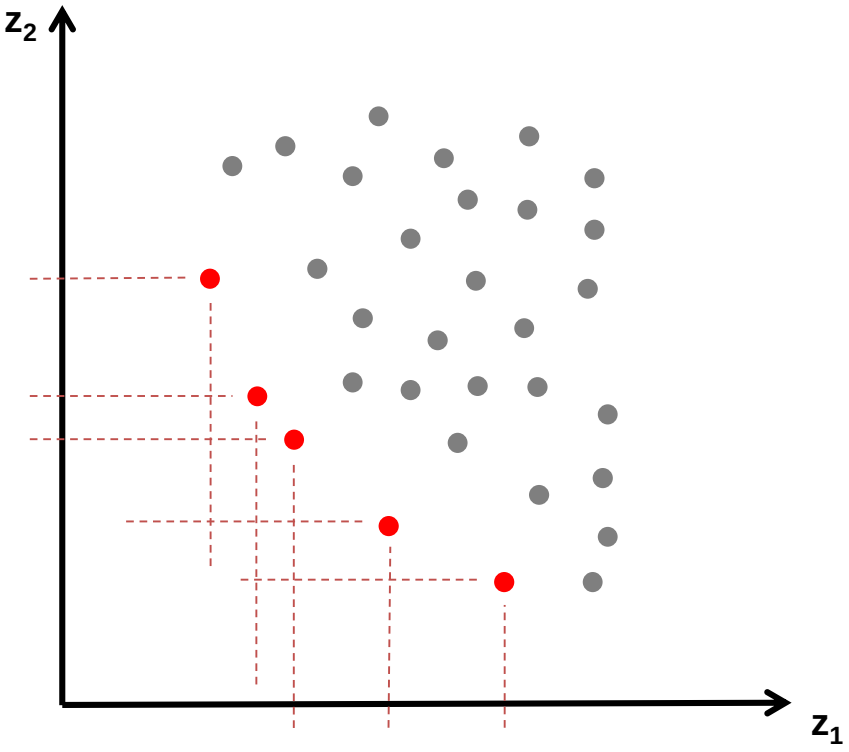
Definition:

Given feasible solutions x and x*, we say x* **dominates** x if

$$z(x) \neq z(x^*) \text{ and } z_i(x^*) \leq z_i(x) \ \forall i = 1, 2$$

$z(x_N)$ is a **nondominated point (NDP)** if x_N is feasible and no other feasible solution dominates it.

Nondominated Frontier



Definition:

Given feasible solutions x and x*, we say x* **dominates** x if

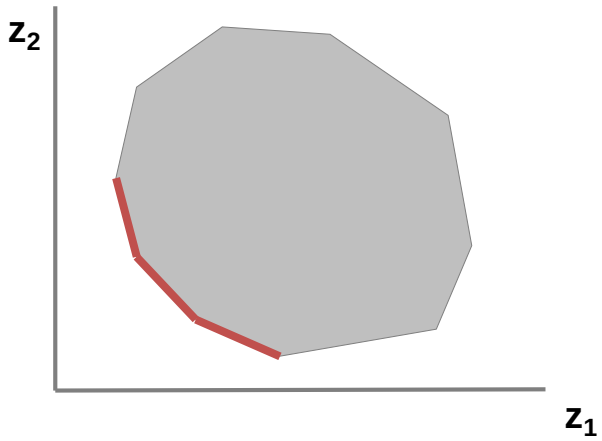
$$z(x) \neq z(x^*) \text{ and } z_i(x^*) \leq z_i(x) \ \forall i = 1, 2$$

$z(x_N)$ is a **nondominated point (NDP)** if x_N is feasible and no other feasible solution dominates it.

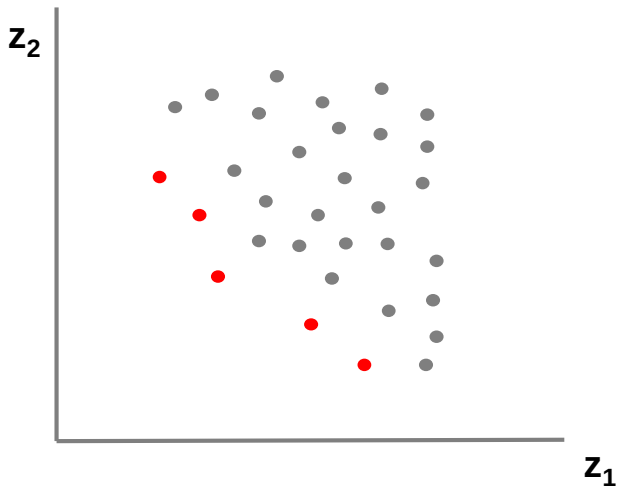
The **nondominated frontier** is the union of all NDPs.

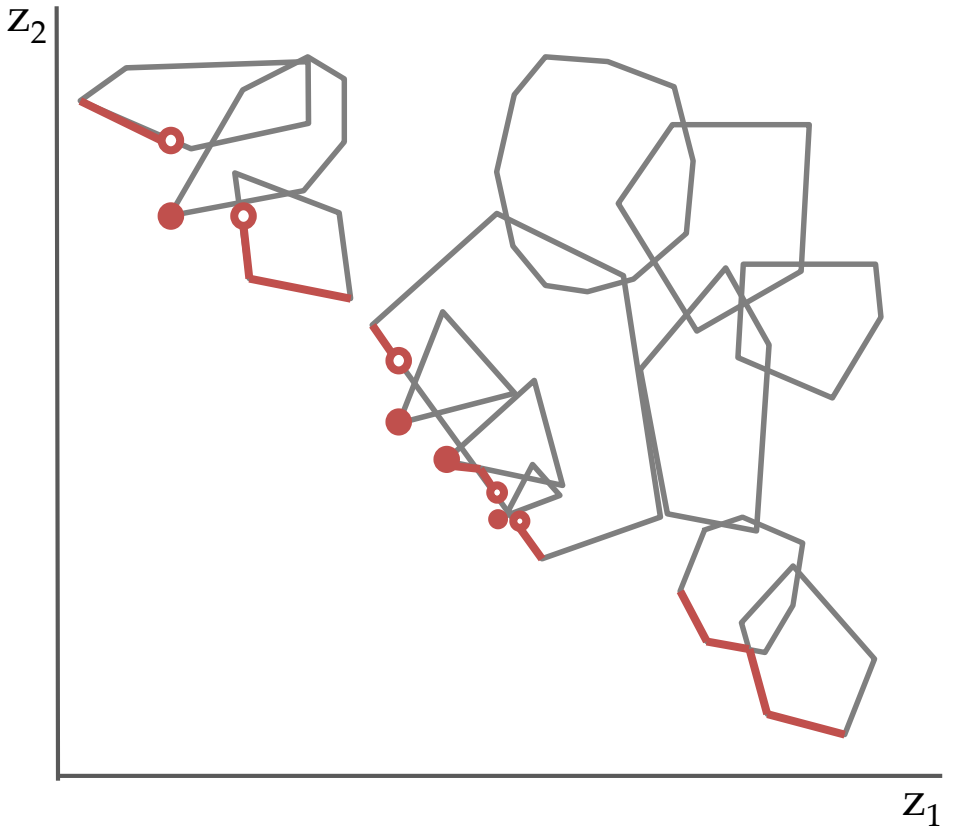
Nondominated Frontiers

LP



IP





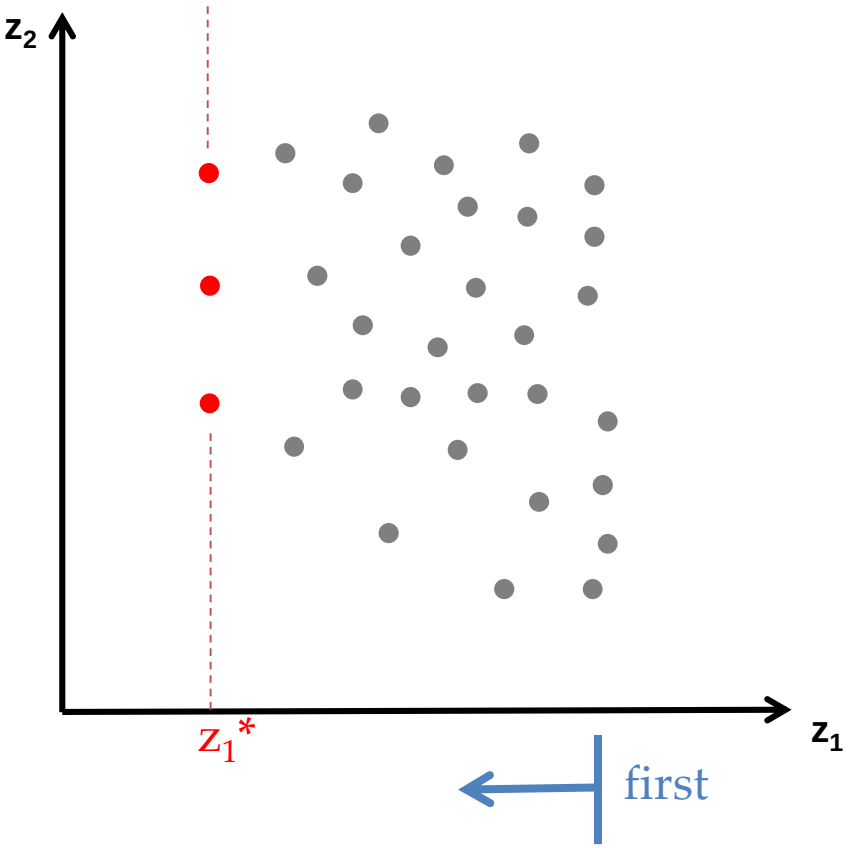
MIP

- isolated points
- open, half-open, and closed line segments

Finding NDPs

- NDPs are found in criterion-space search algorithms by solving single-objective IPs.
- These single-objective IPs are the "workhorses" of the algorithms
- Algorithms solves two types: Lexicographic IPs & Scalarized IPs

Lexicographic IP

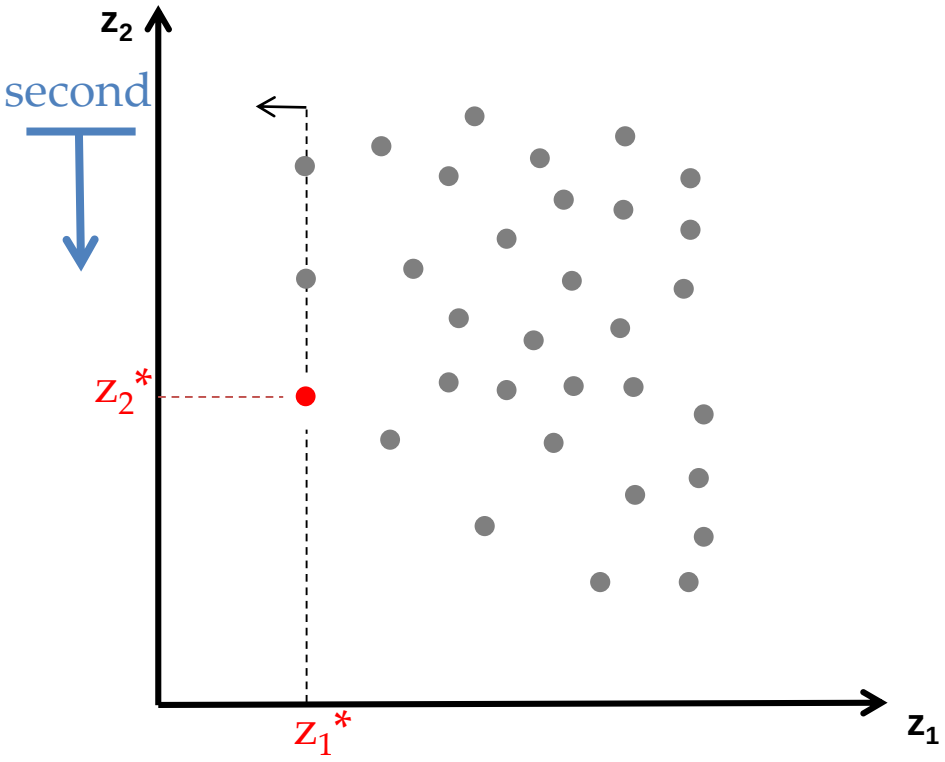


Minimize objectives sequentially:

(z_1, z_2) = lexmin $\{z_1(x), z_2(x)\}$ is defined by

$$z_1^* := \min\{z_1(x) : x \in X\}$$

Lexicographic IP



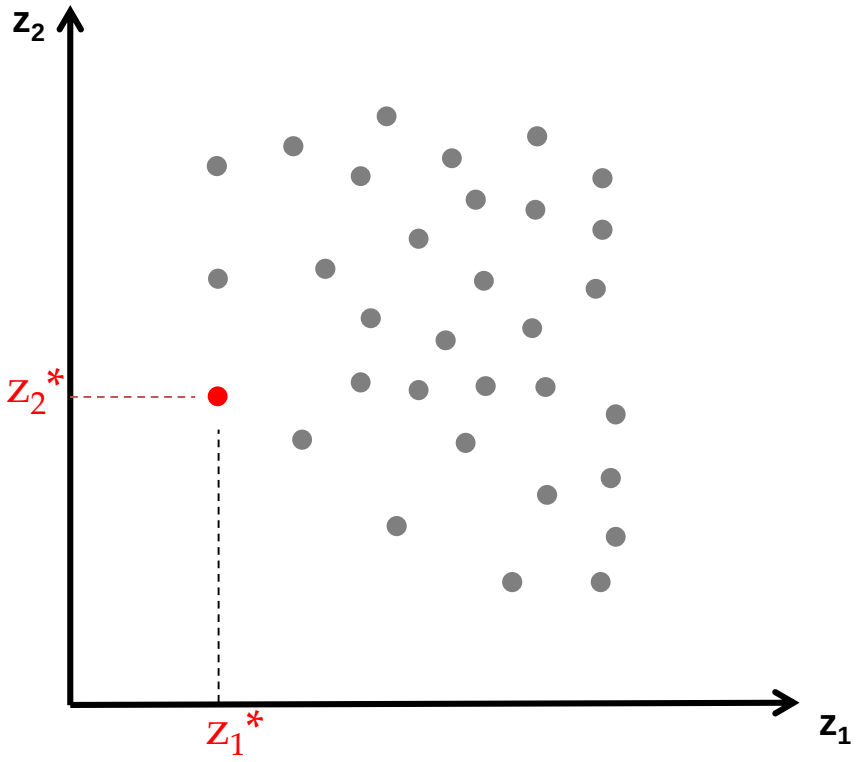
Minimize objectives sequentially:

$(z_1, z_2) = \text{lexmin}\ \{z_1(x), z_2(x)\}$ is defined by

$$z_1^* := \min\{z_1(x) : x \in X\}$$

$$z_2^* := \min\{z_2(x) : z_1(x) \leq z_1^*, x \in X\}$$

Lexicographic IP



Minimize objectives sequentially:

(z_1, z_2) = lexmin $\{z_1(x), z_2(x)\}$ is defined by

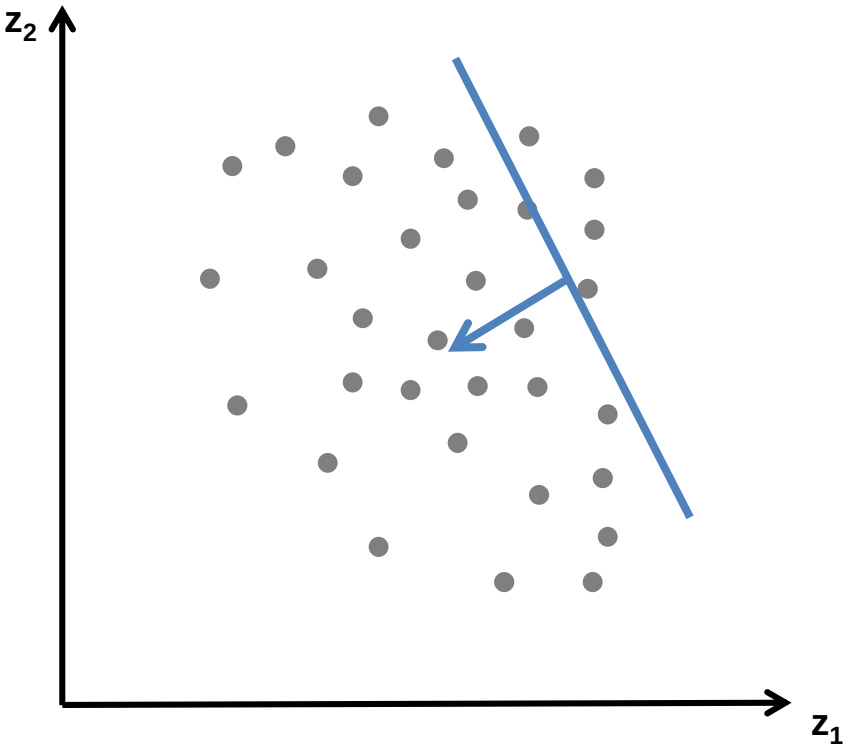
$$z_1^* := \min\{z_1(x) : x \in X\}$$

$$z_2^* := \min\{z_2(x) : z_1(x) \leq z_1^*, x \in X\}$$

(z_1^*, z_2^*) is guaranteed to be an NDP.

We count each lexicographic IP as two IPs, although the second IP usually solves rather quickly.

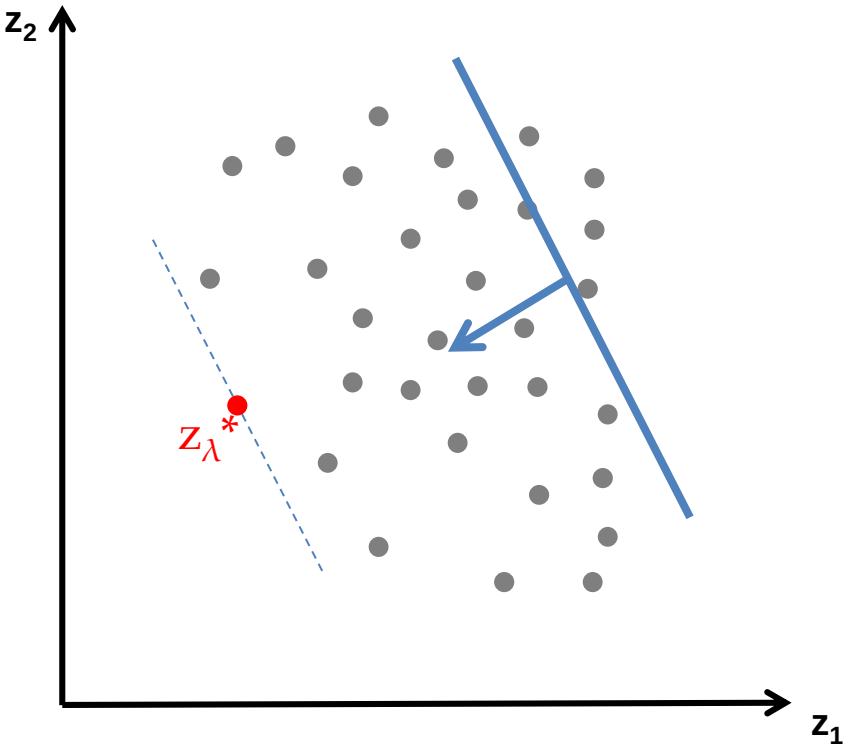
Scalarized IP



Transform vector of objective values into a scalar with positive weight $0 < \lambda < 1$:

$$z_\lambda^* := \min\{\lambda z_1(x) + (1-\lambda) z_2(x) : x \in X\}$$

Scalarized IP



Transform vector of objective values into a scalar with positive weight $0 < \lambda < 1$:

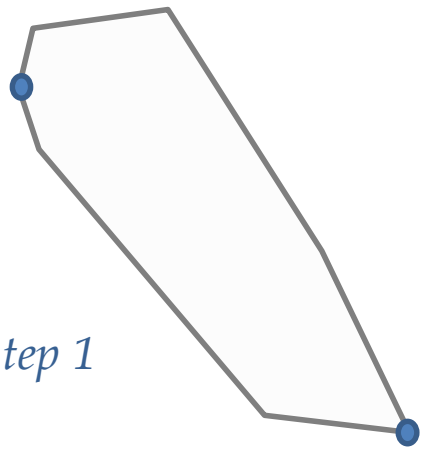
$$z_\lambda^* := \min\{\lambda z_1(x) + (1-\lambda) z_2(x) : x \in X\}$$

z_λ^* is guaranteed to be an NDP.

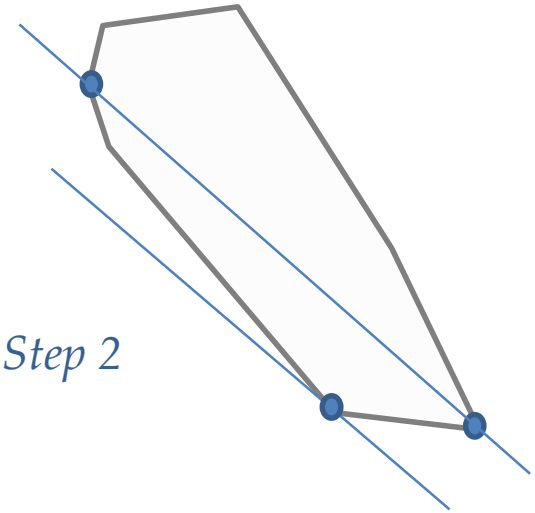
Dichotomic Search

Dichotomic search

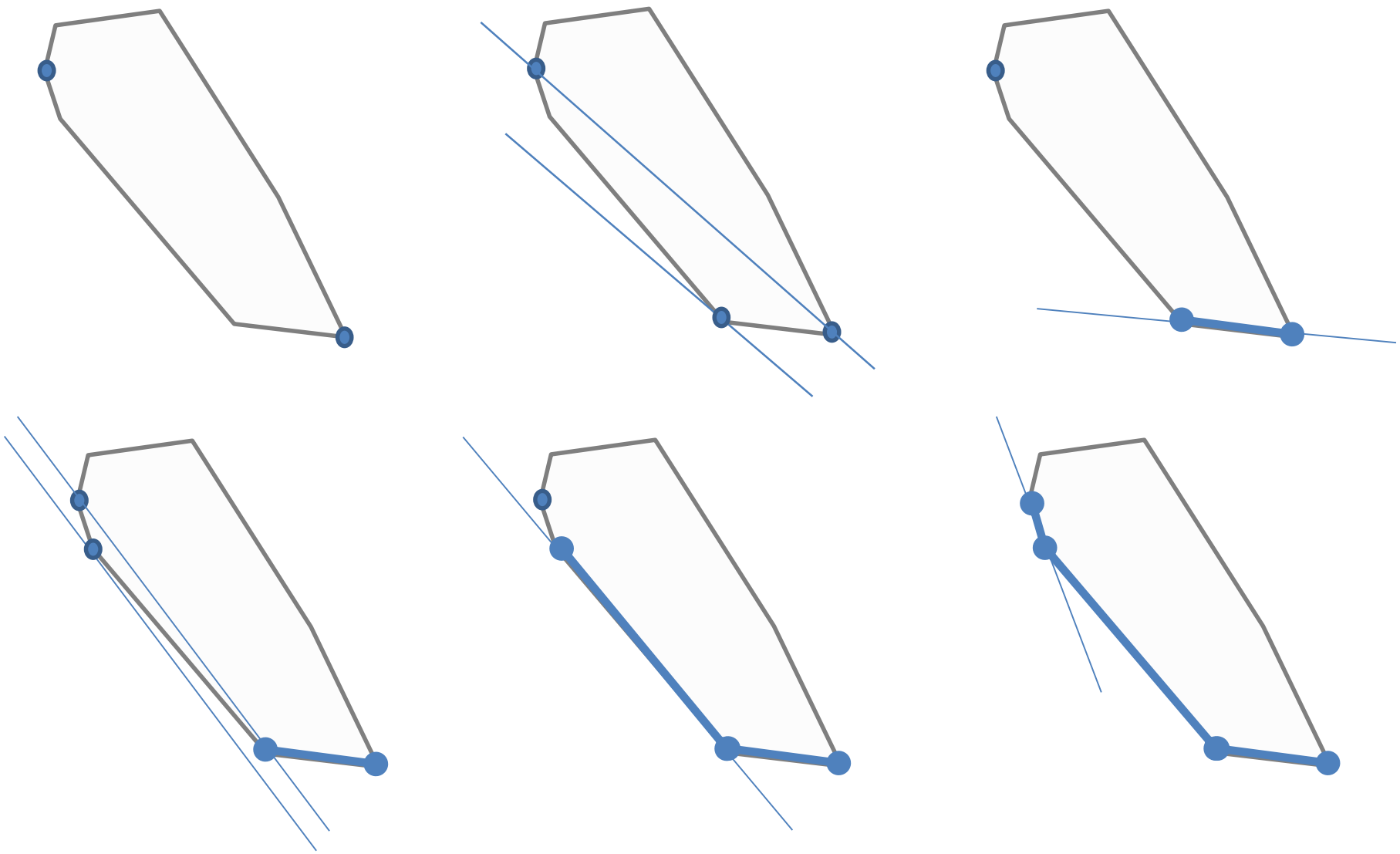
- An algorithm to find a set of nondominated points
- Step 1: lexmin $\{z_1(x), z_2(x)\}$, lexmin $\{z_2(x), z_1(x)\}$
- Step 2,…: Scalarized LP/IP



Step 1

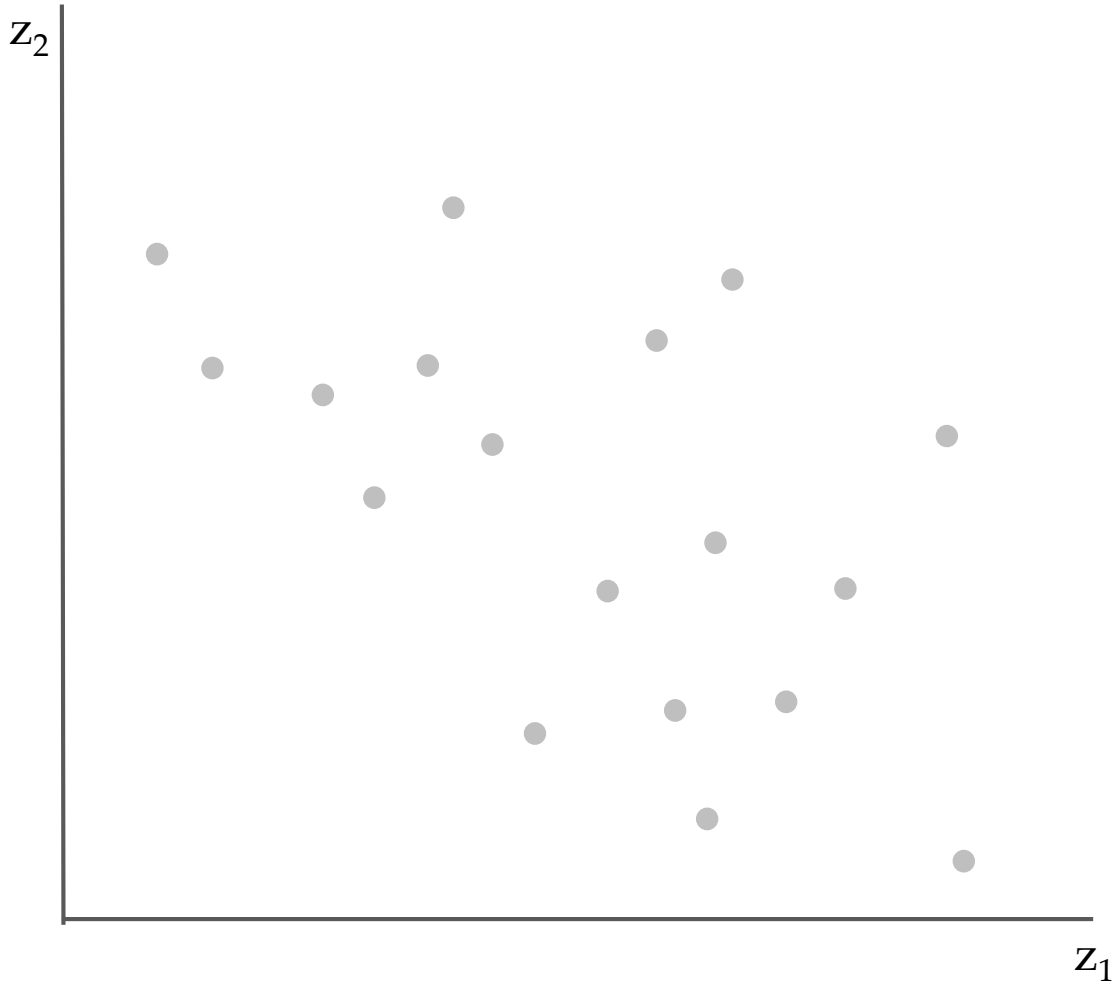


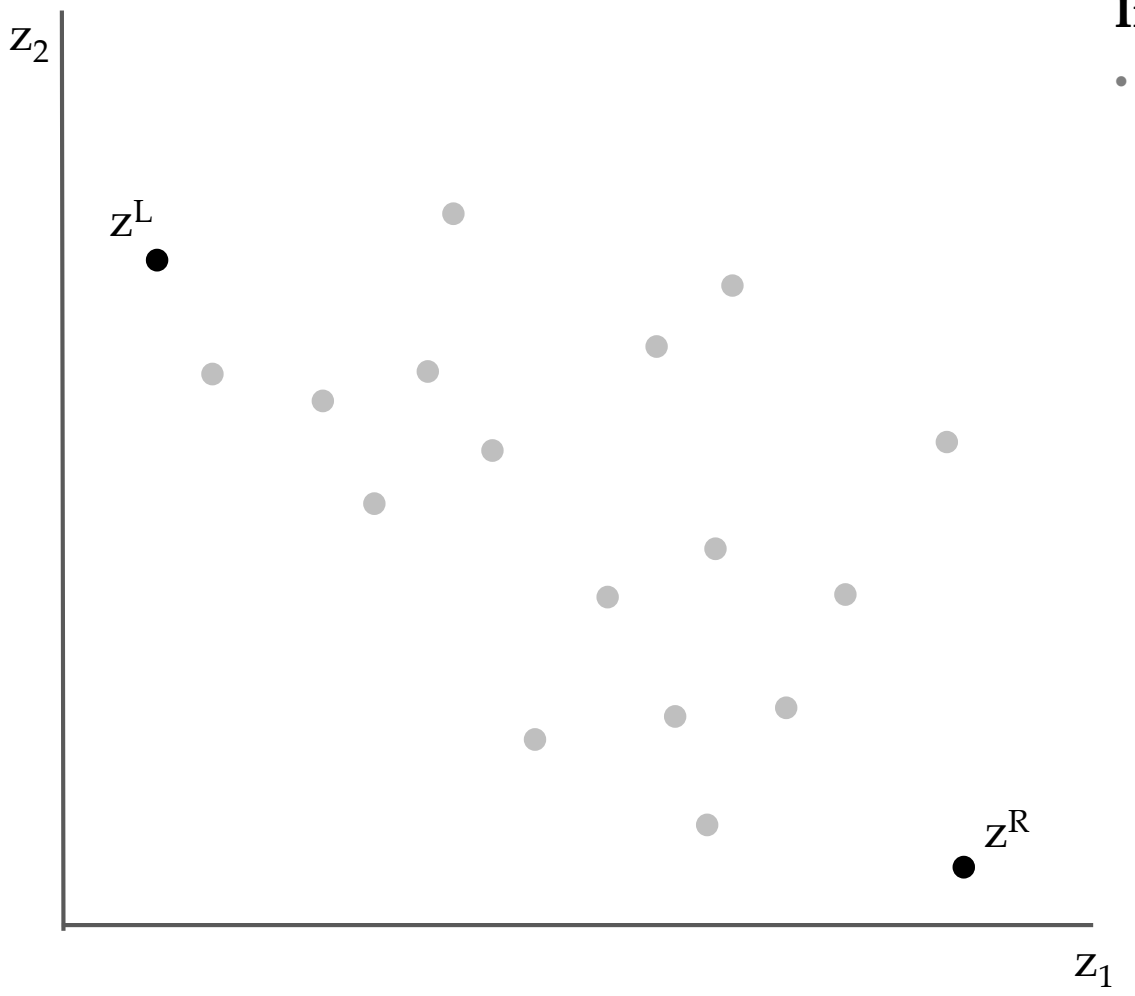
Step 2



Note: Requires only LP solves when
(1) MOLP or (2) MOMIP with IP part of solution is fixed

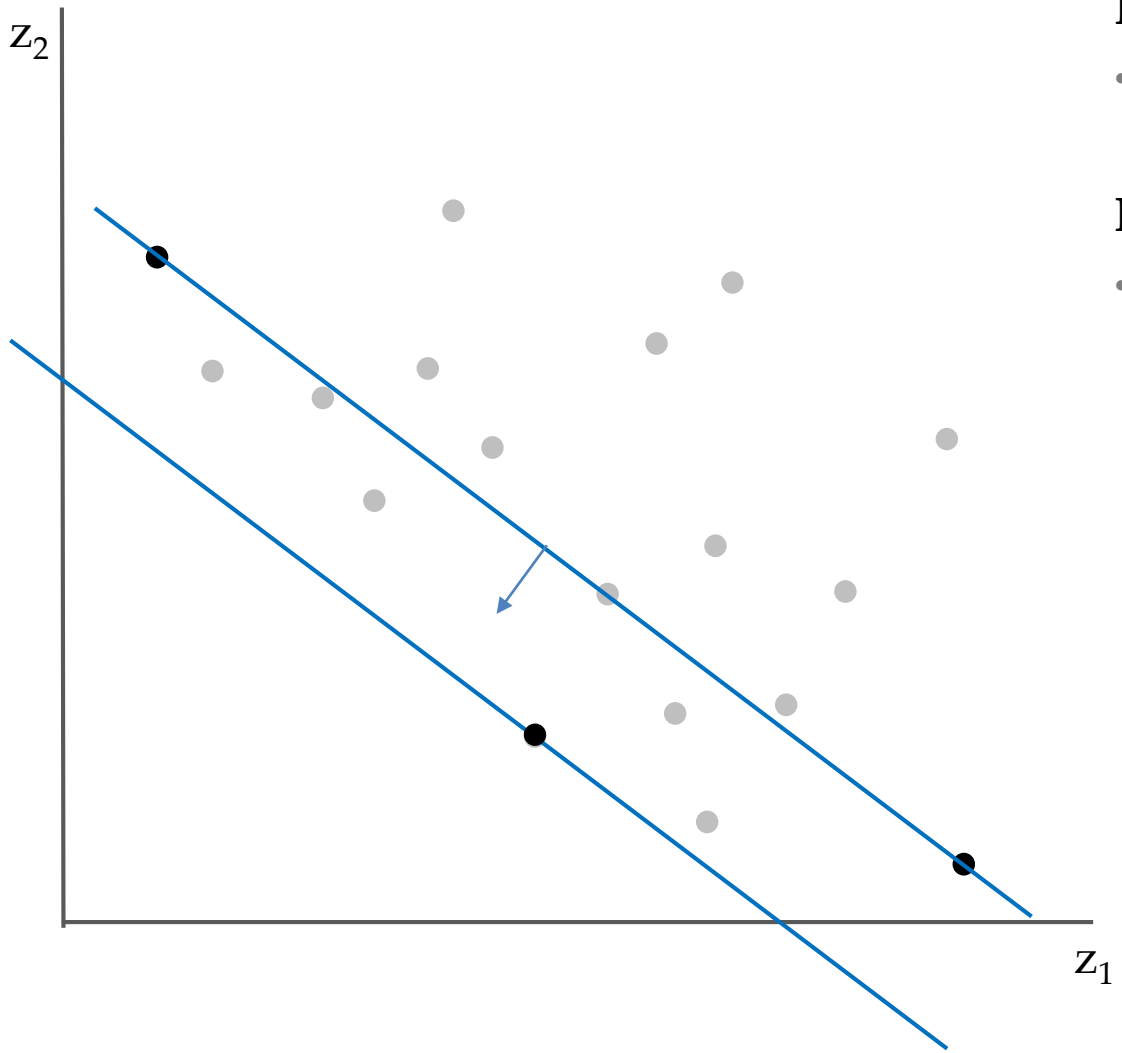
Dichotimic search for pure IPs





Initialization:

- Find corner point NDPs.

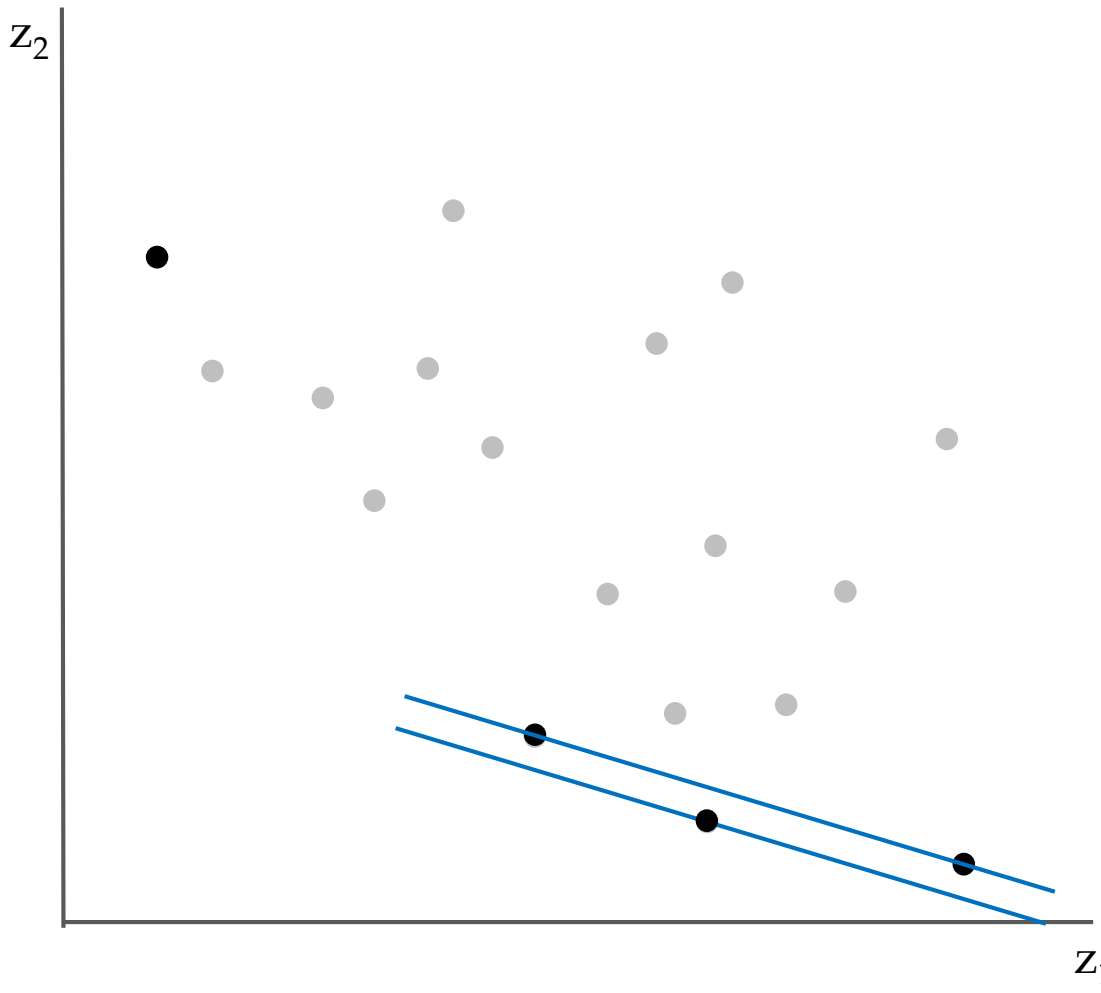


Initialization:

- Find corner point NDPs.

Main loop:

- Solve scalarized IP to find next NDP

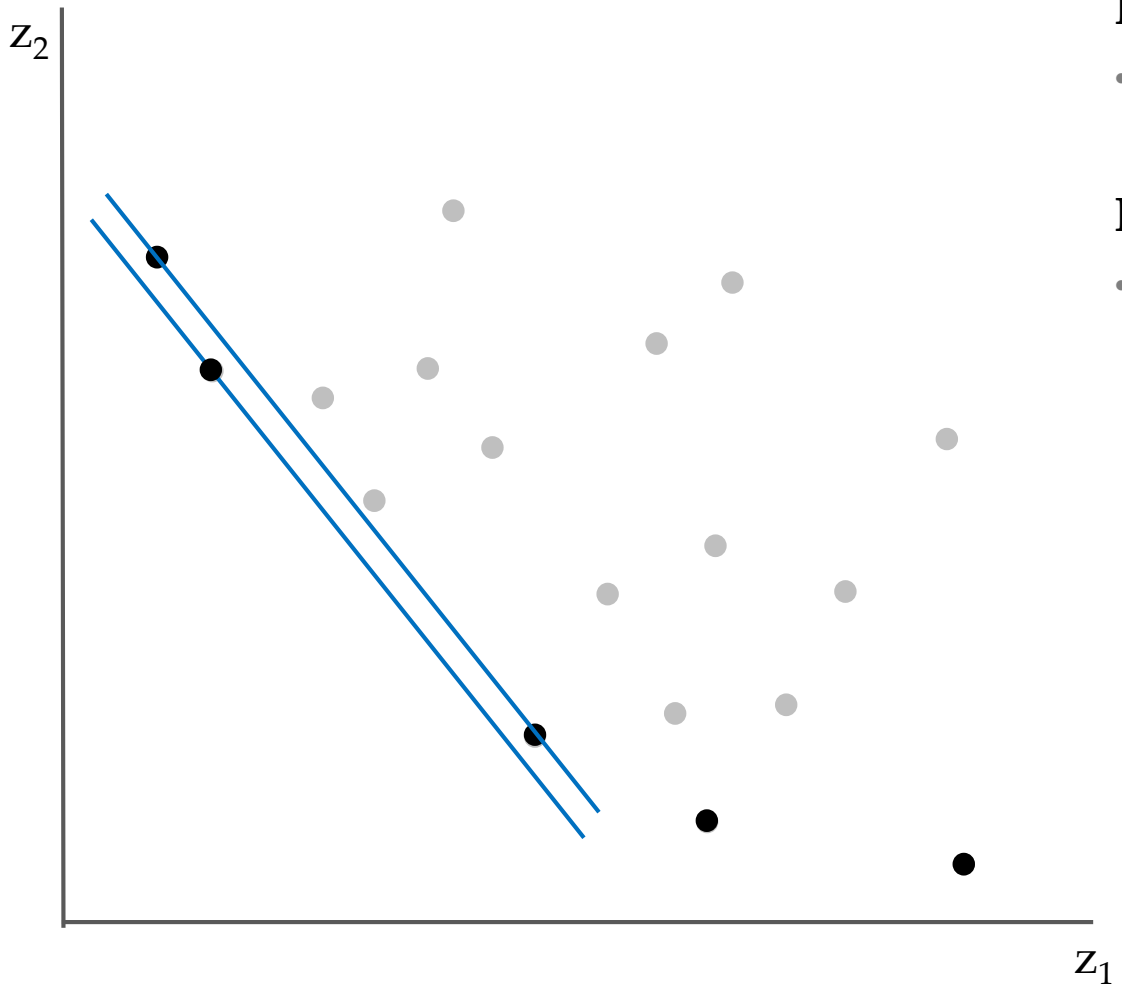


Initialization:

- Find corner point NDPs.

Main loop:

- Solve scalarized IP to find next NDP

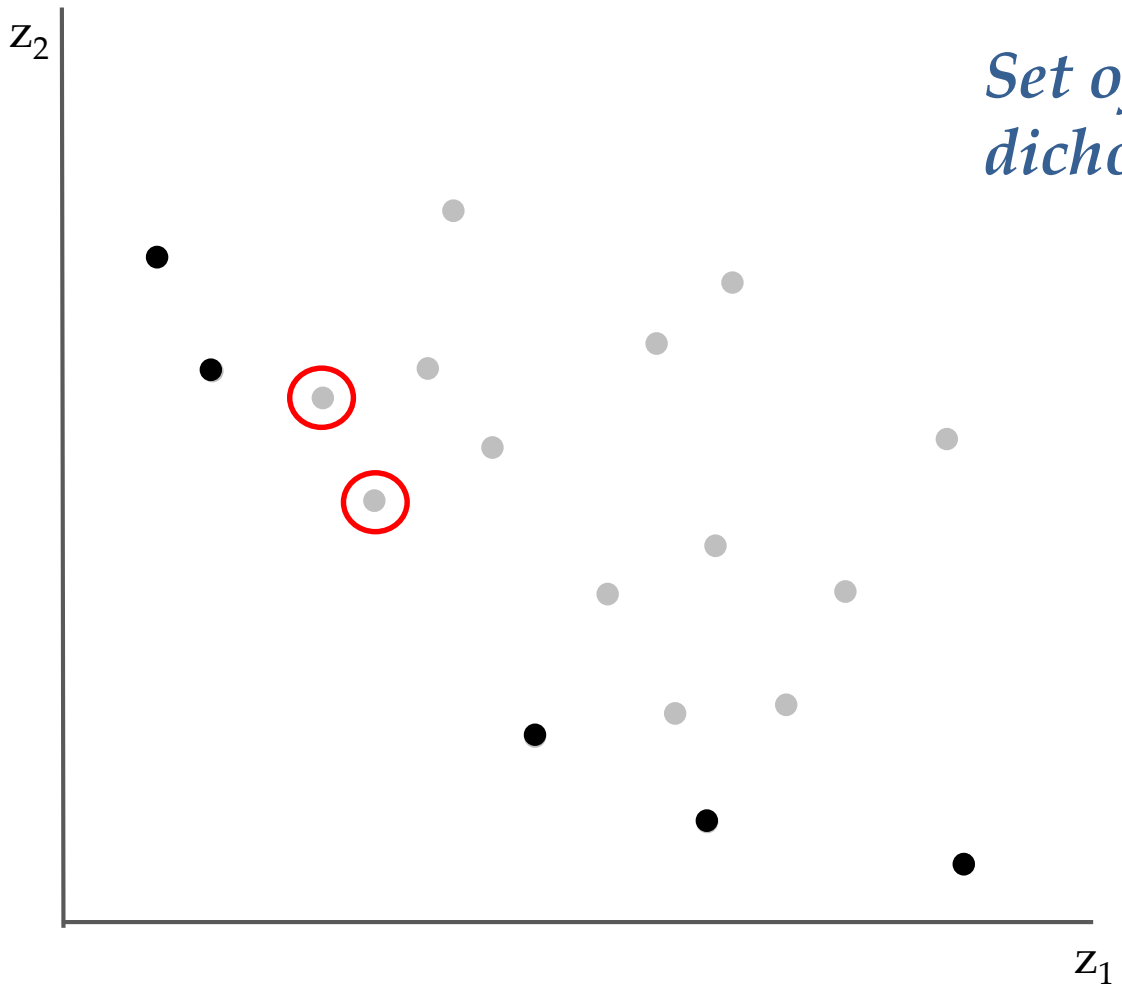


Initialization:

- Find corner point NDPs.

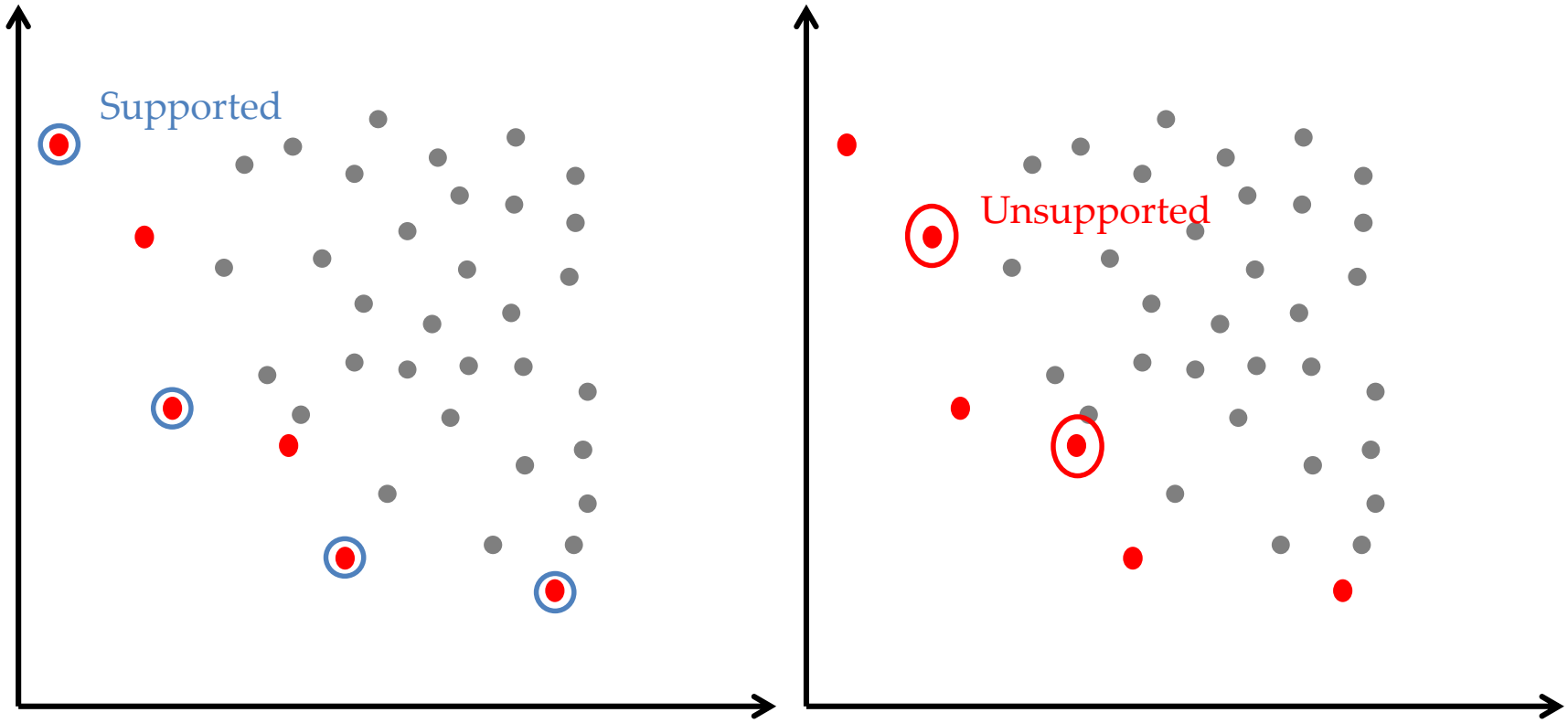
Main loop:

- Solve scalarized IP to find next NDP



Set of NDPs found by dichotomic search

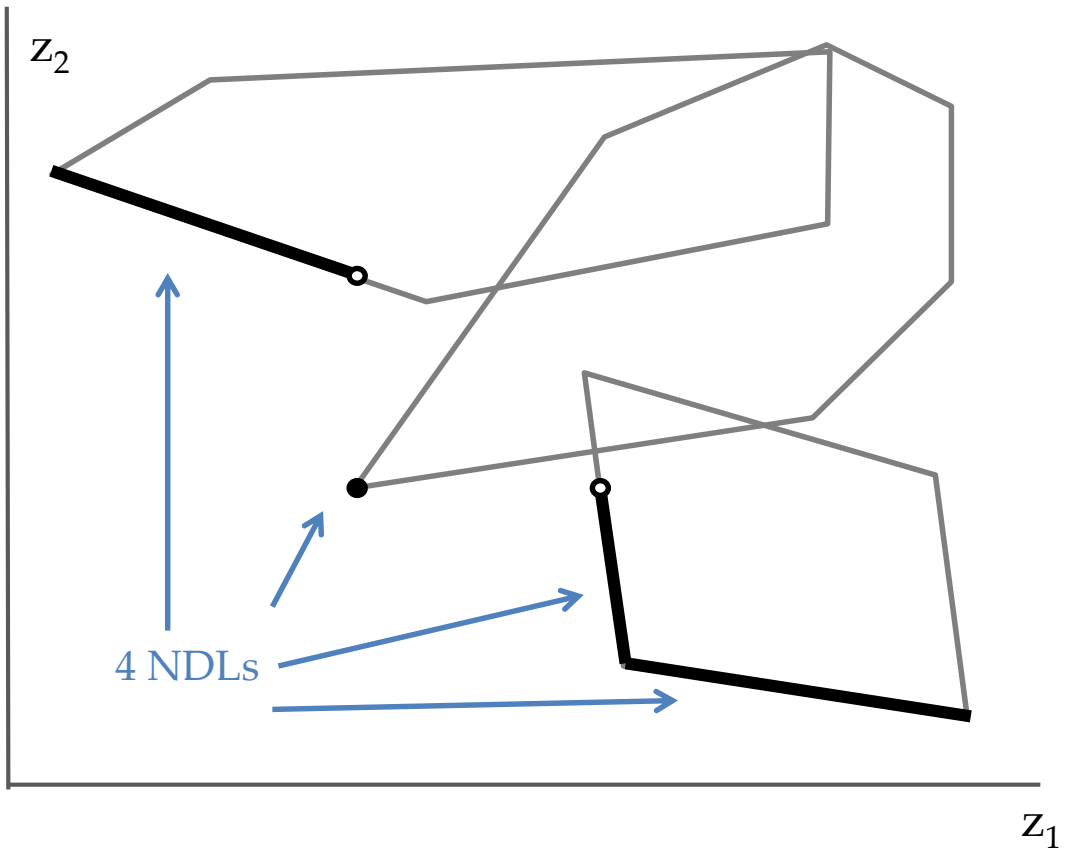
Nonsupported NDPs



The existence of unsupported NDPs makes finding the nondominated frontier of an IP much more difficult than finding the nondominated frontier of an LP

Nondominated frontier of a BOMIP

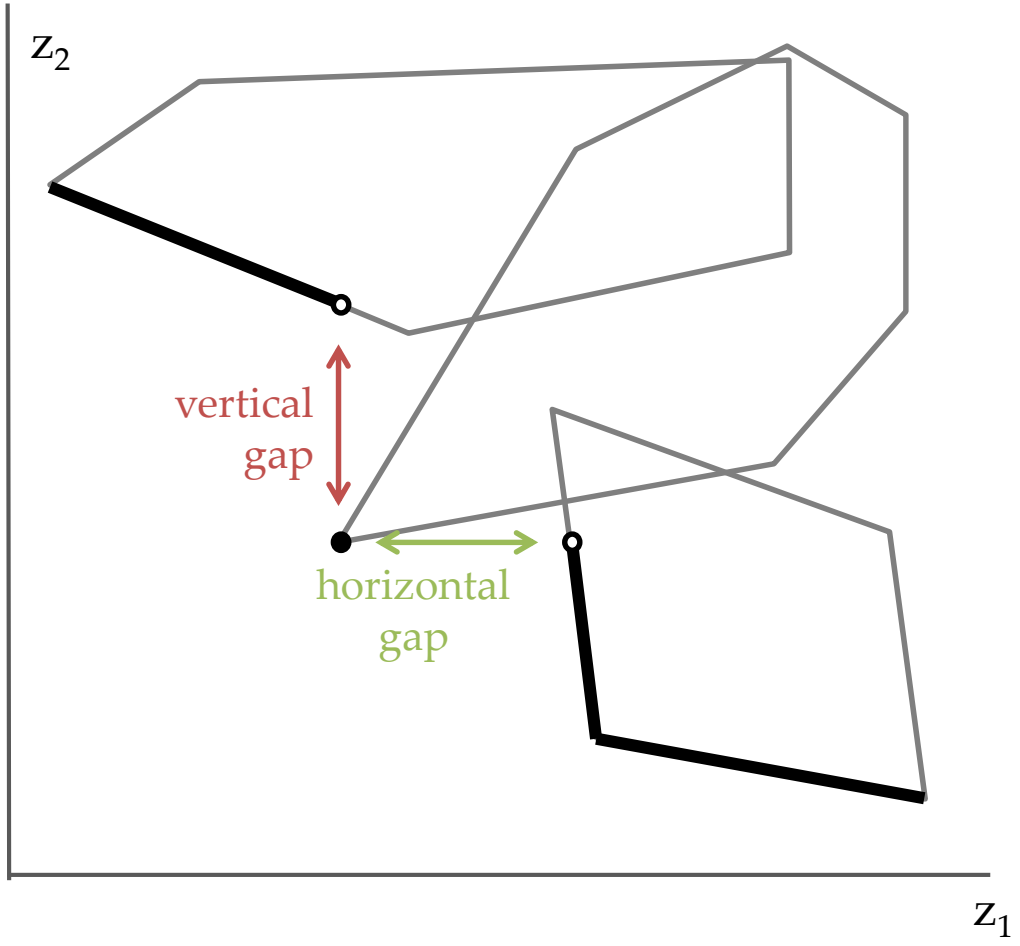
- Nondominated line segment (NLS)



The existence of NDLs makes finding the nondominated frontier of a MIP much more difficult than finding the nondominated frontier of an IP

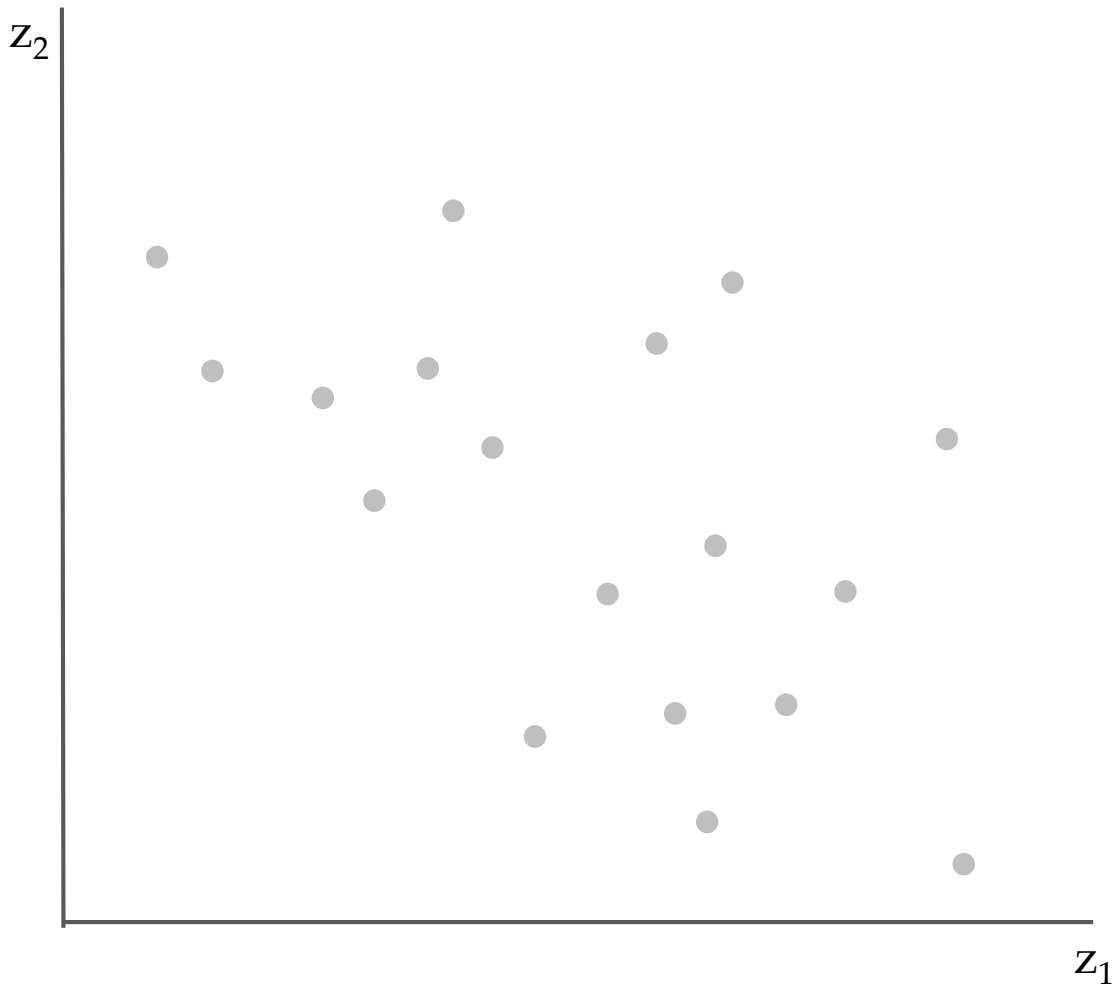
Nondominated frontier of a BOMIP

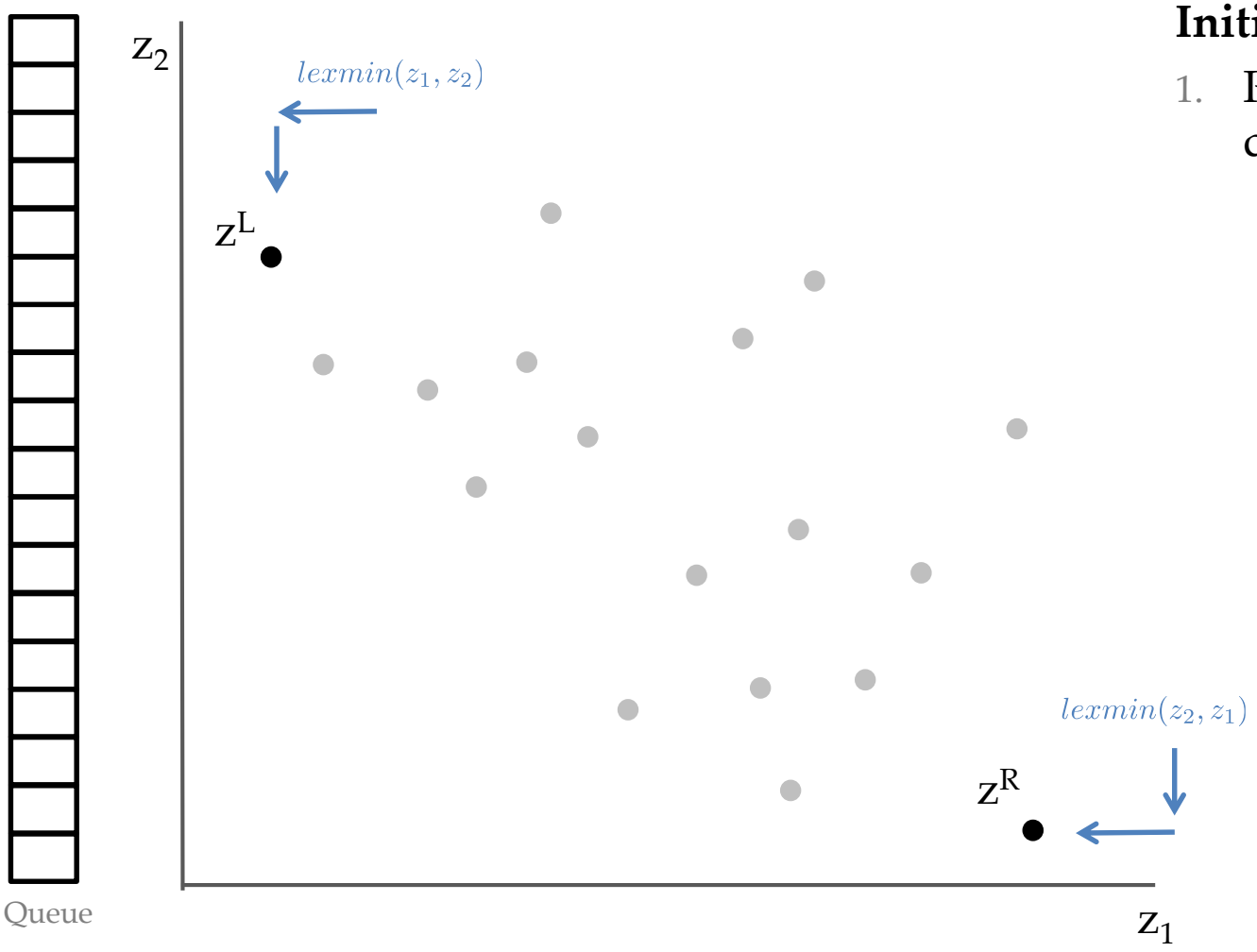
- Vertical gaps
- Horizontal gaps



Balanced Box Method

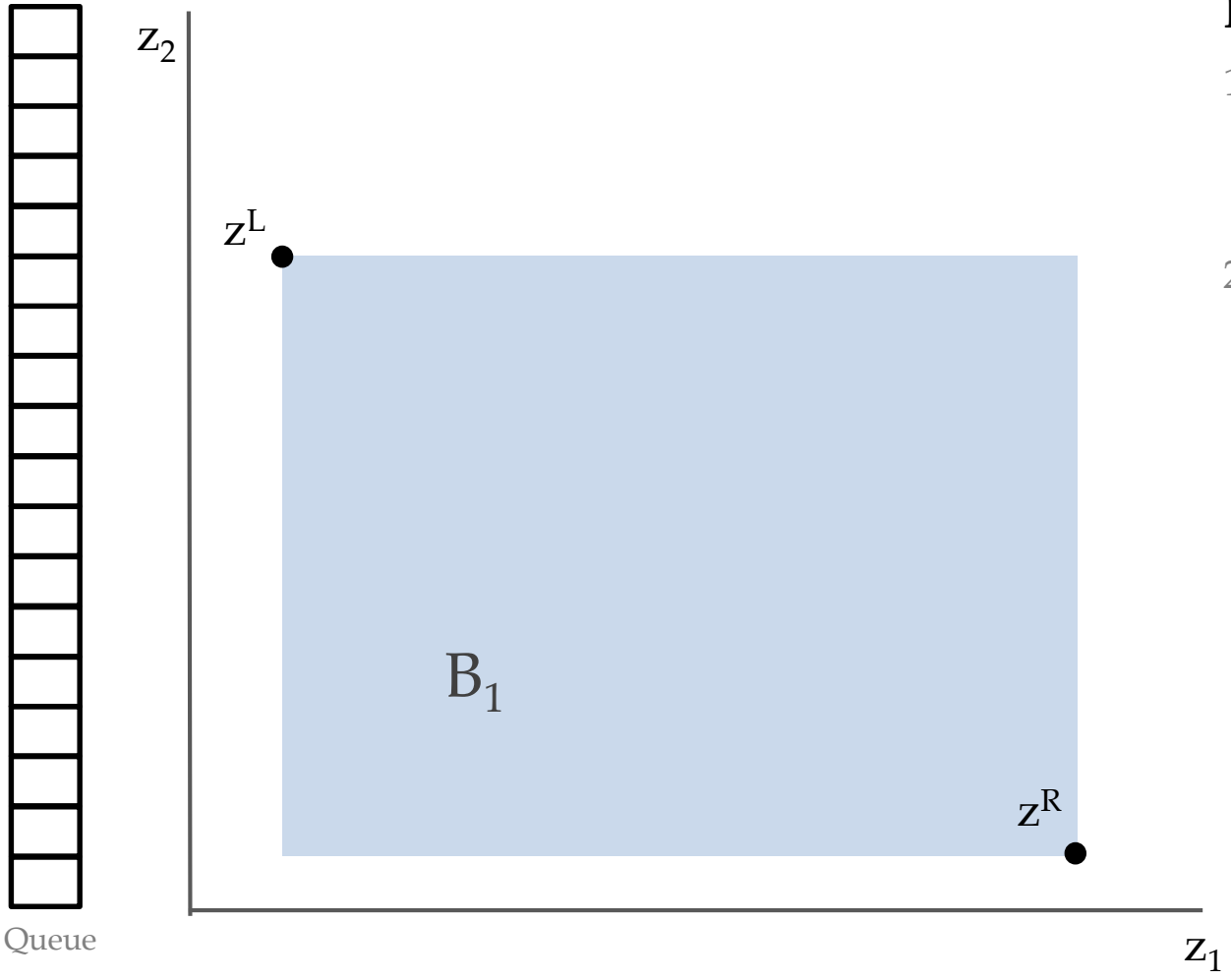
For Biobjective Integer Programs


z_2
z_1



Initialization:

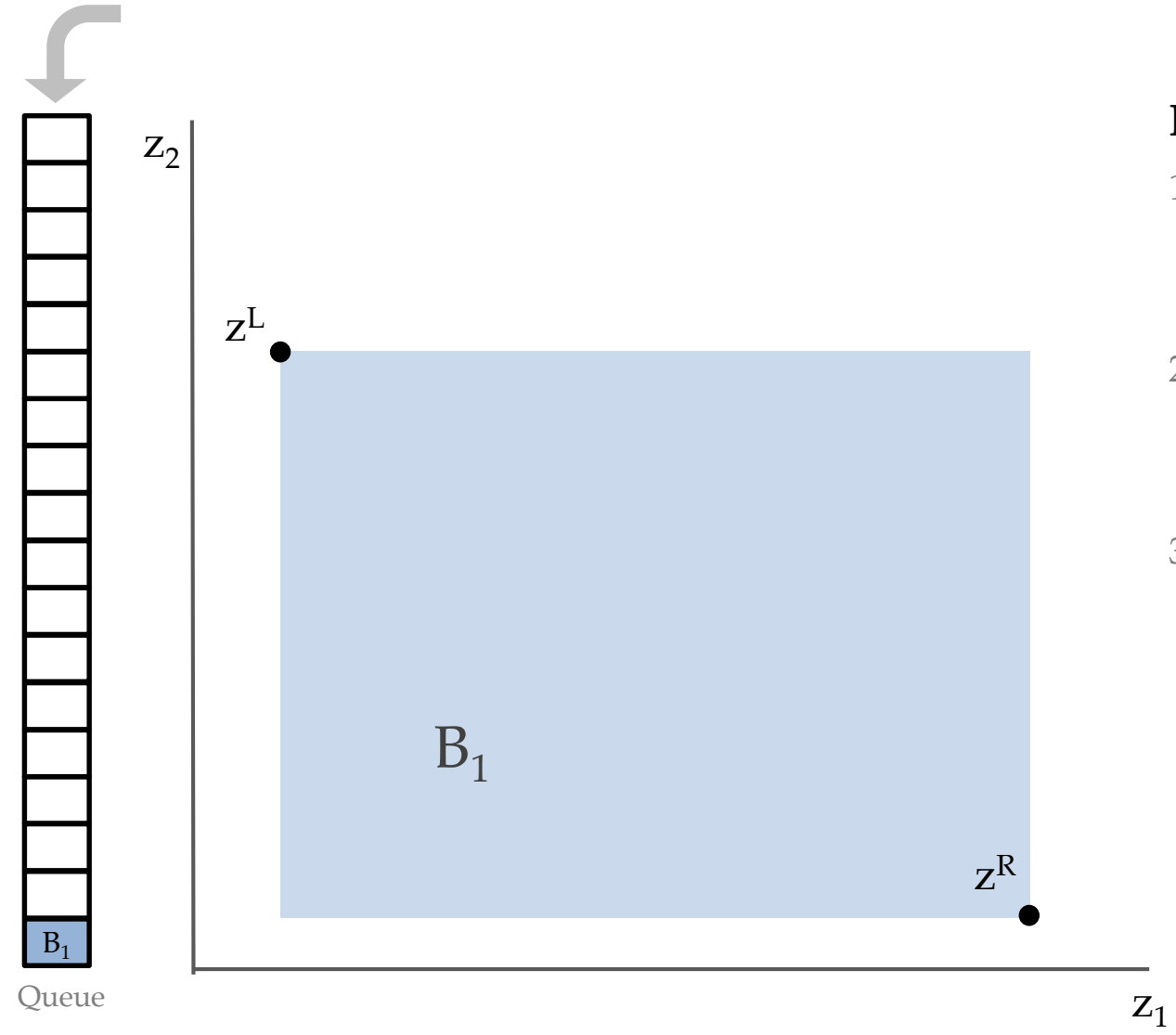
1. Perform two lexmins to discover first two NDPs.



Initialization:

1. Perform two lexmins to discover first two NDPs.

2. These define the corner points of the first box.

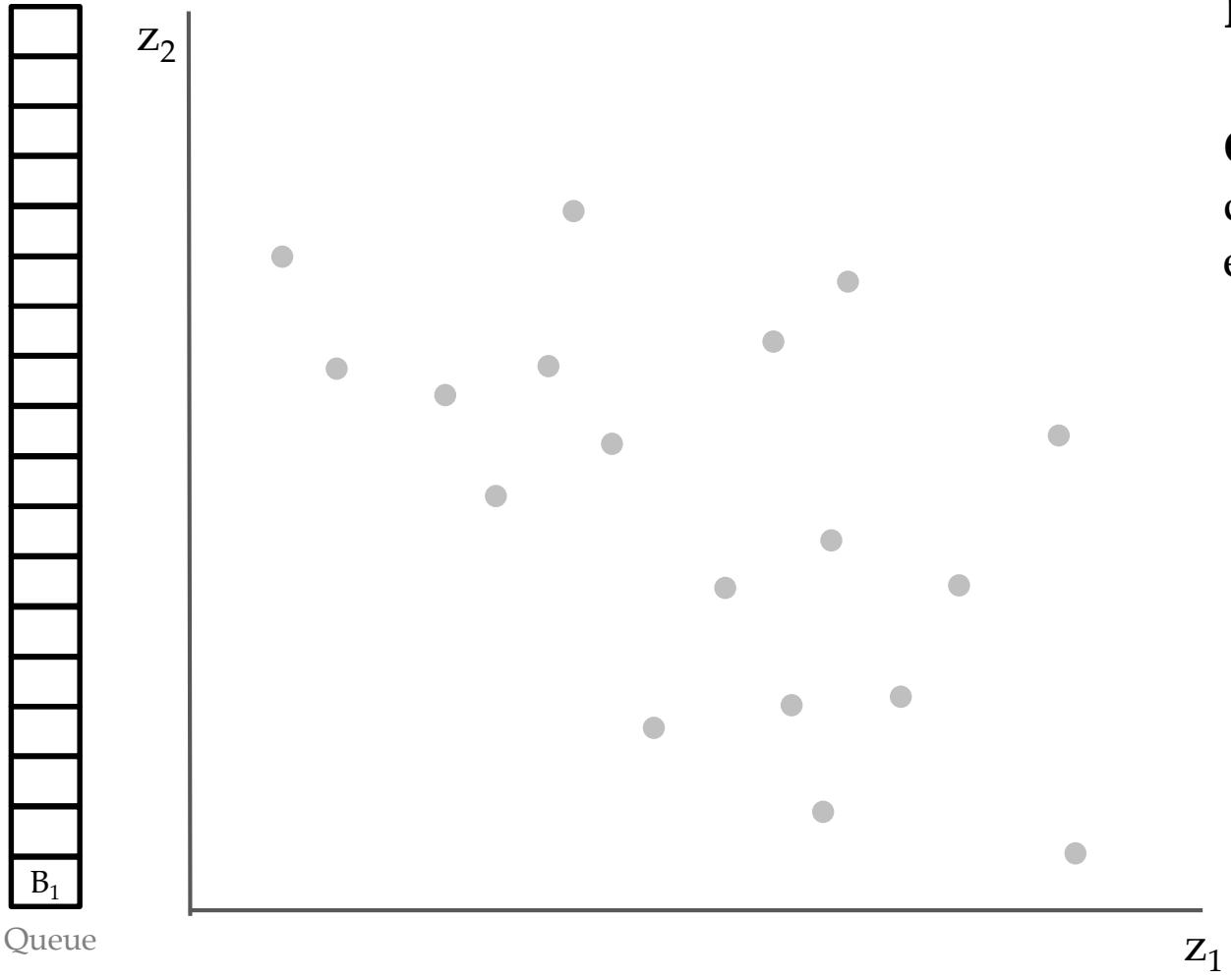


Initialization:

1. Perform two lexmins to discover first two NDPs.
2. These define the corner points of the first box.
3. If the box is nontrivial, add it to queue.

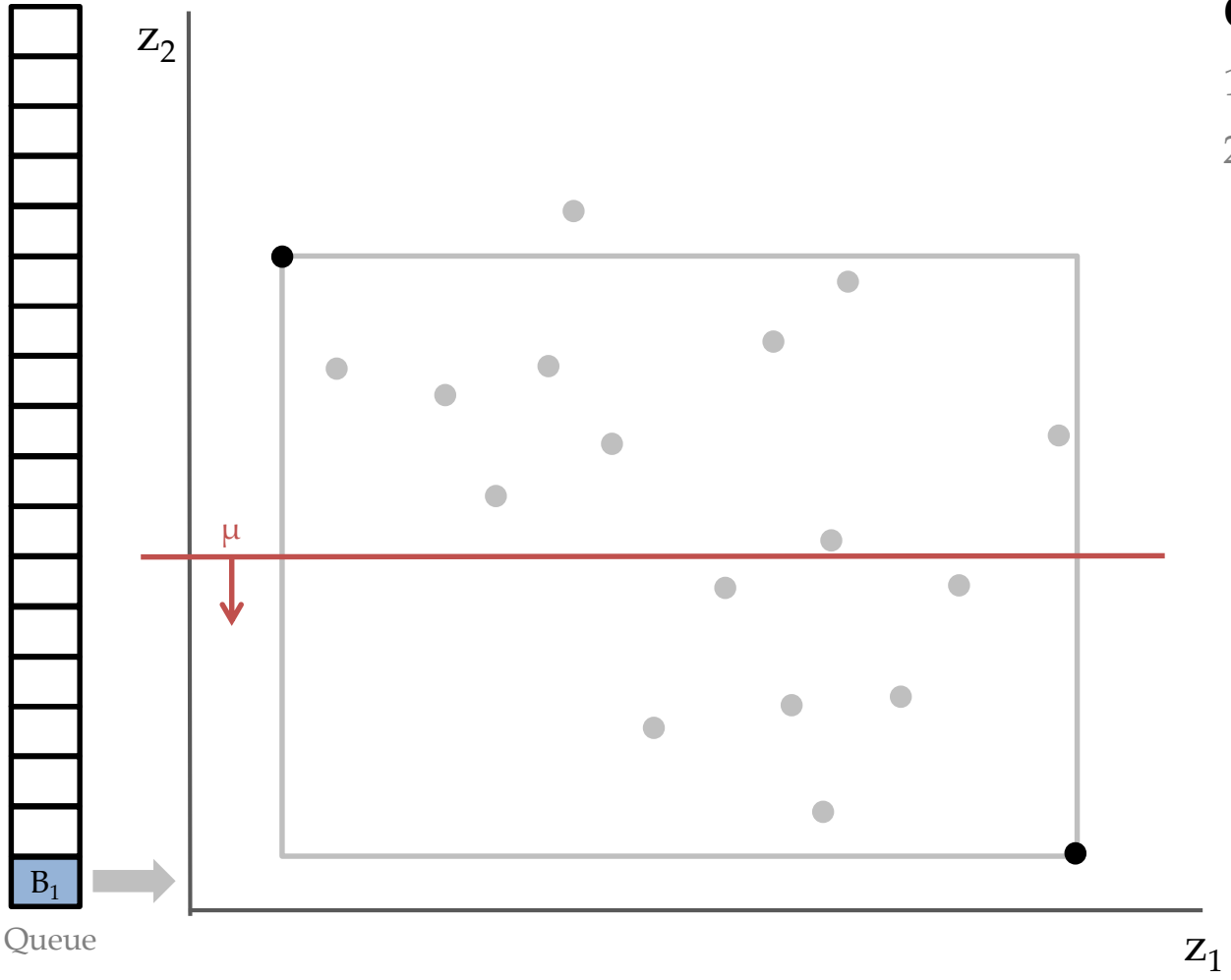
* We always use two distinct NDPs to define a box.

** "Trivially small" boxes are never added to the queue.



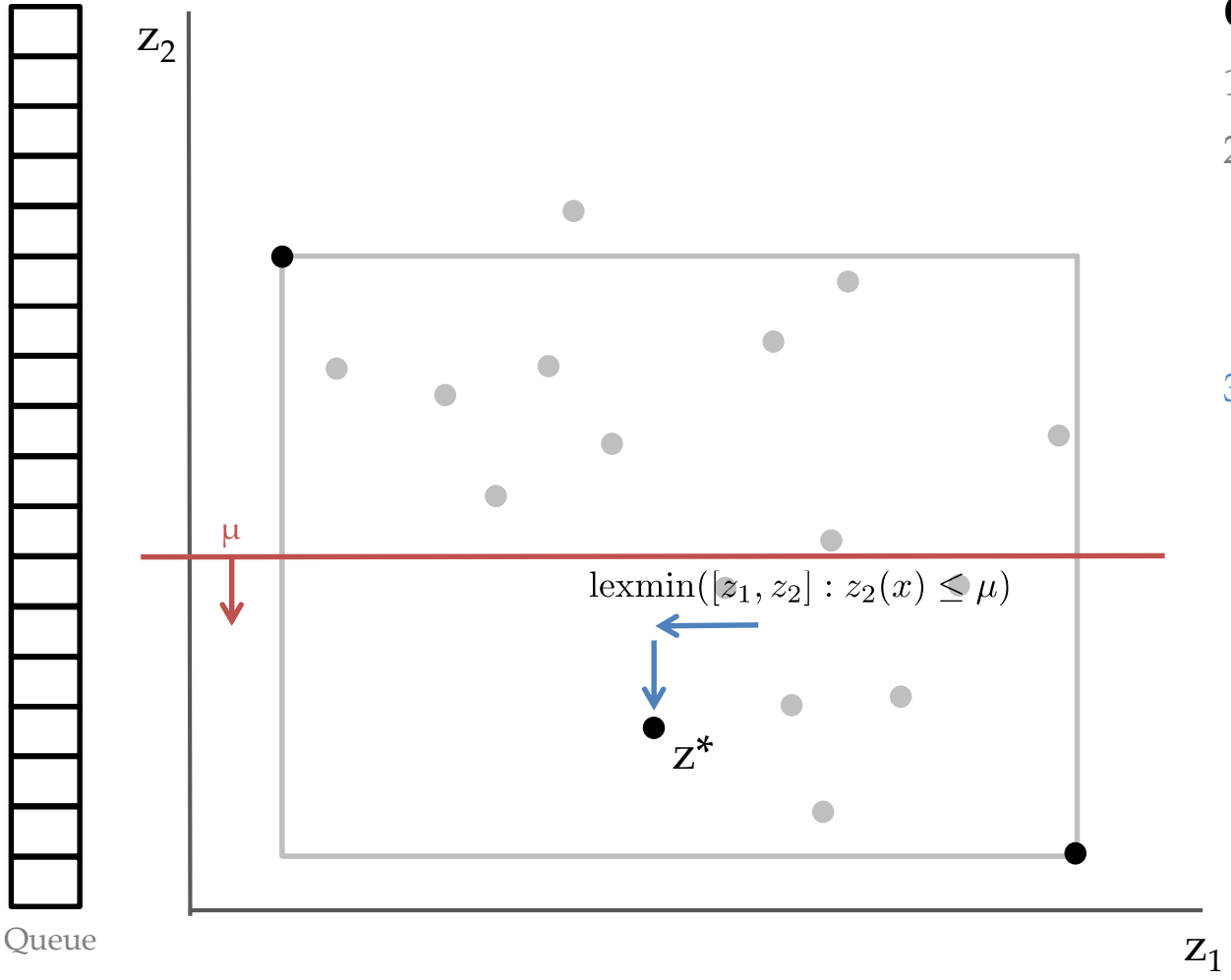
Initialization Complete.

Outer Loop Initiates and continues until queue is empty.



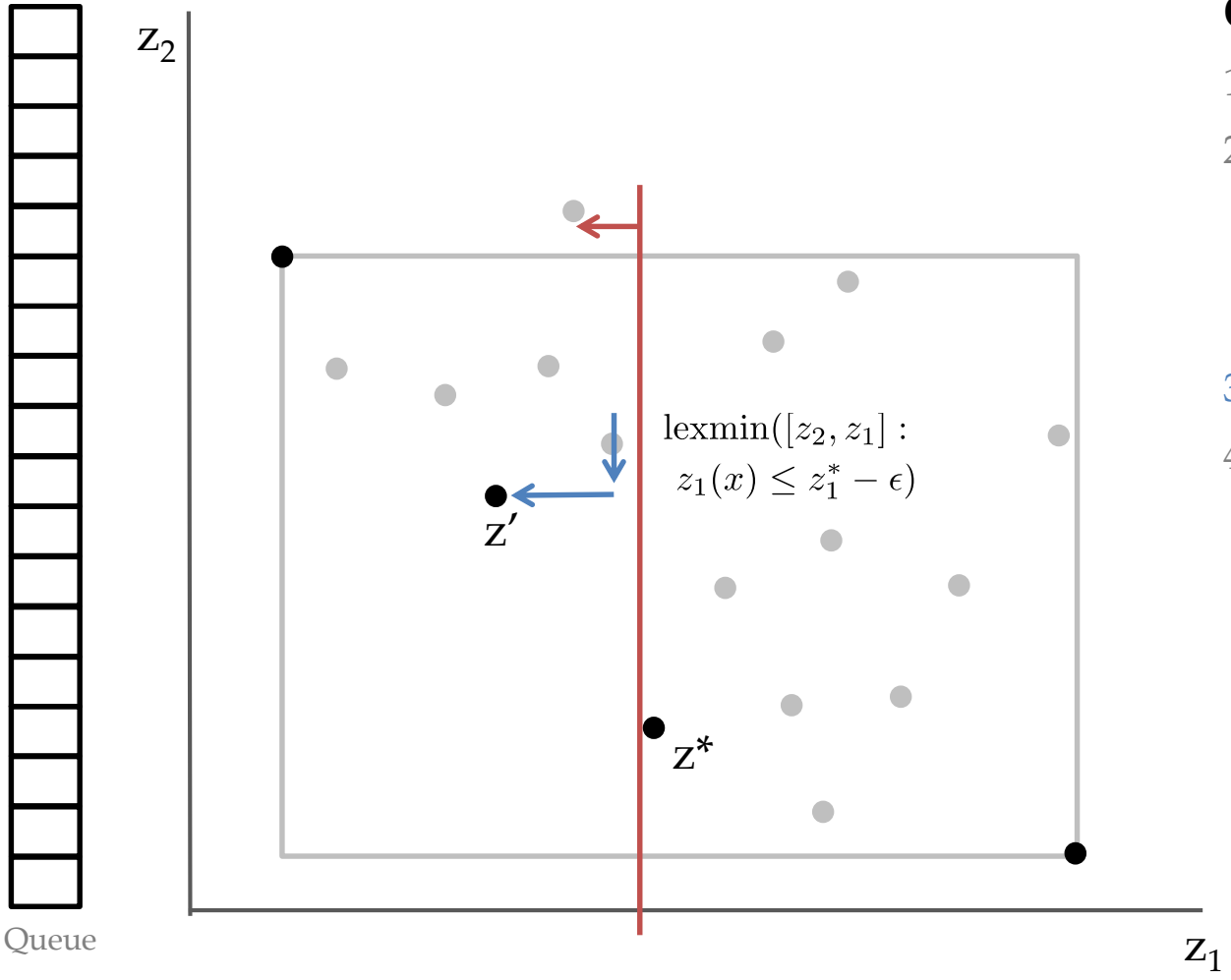
Outer Loop:

1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).



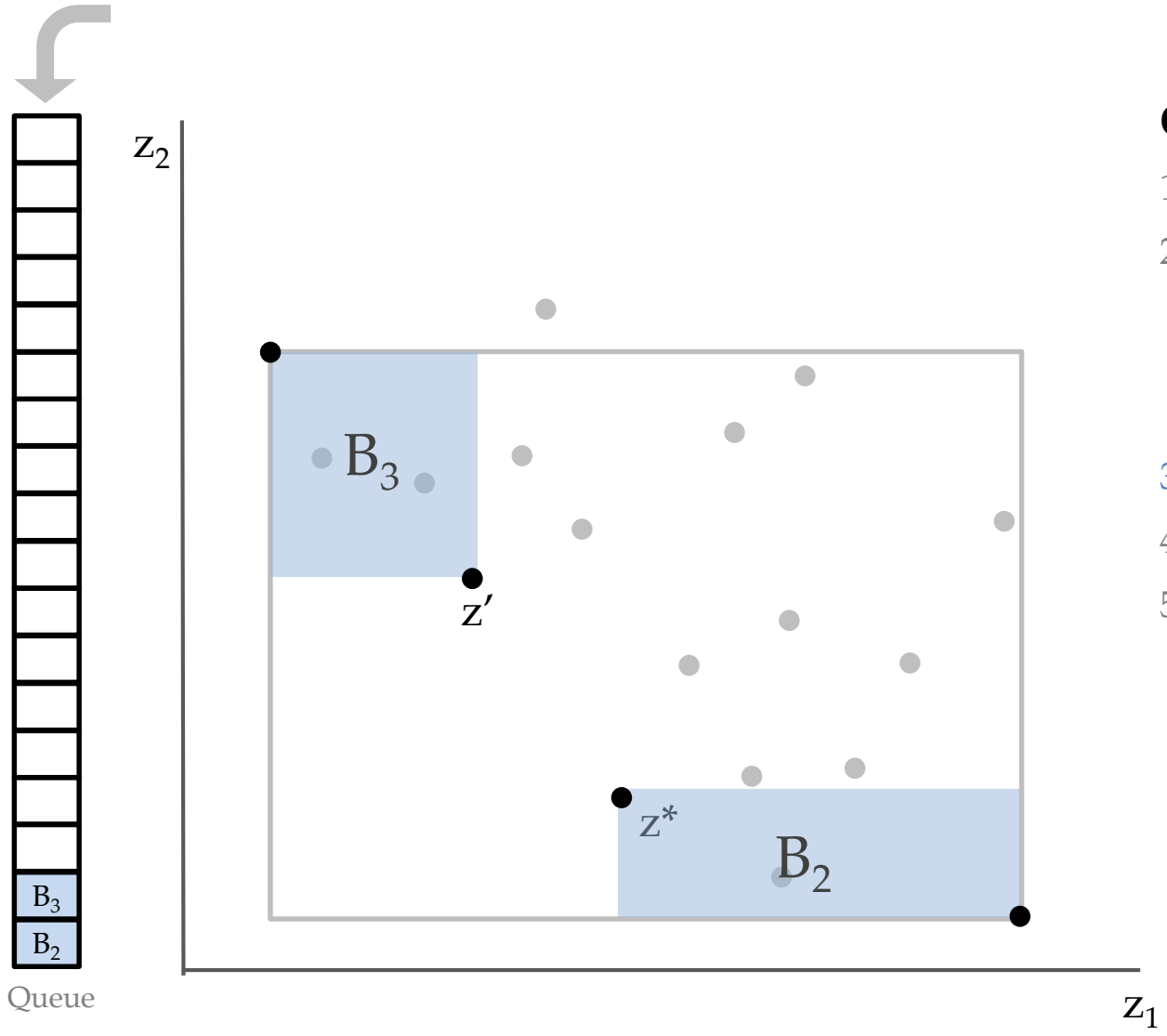
Outer Loop:

1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line.



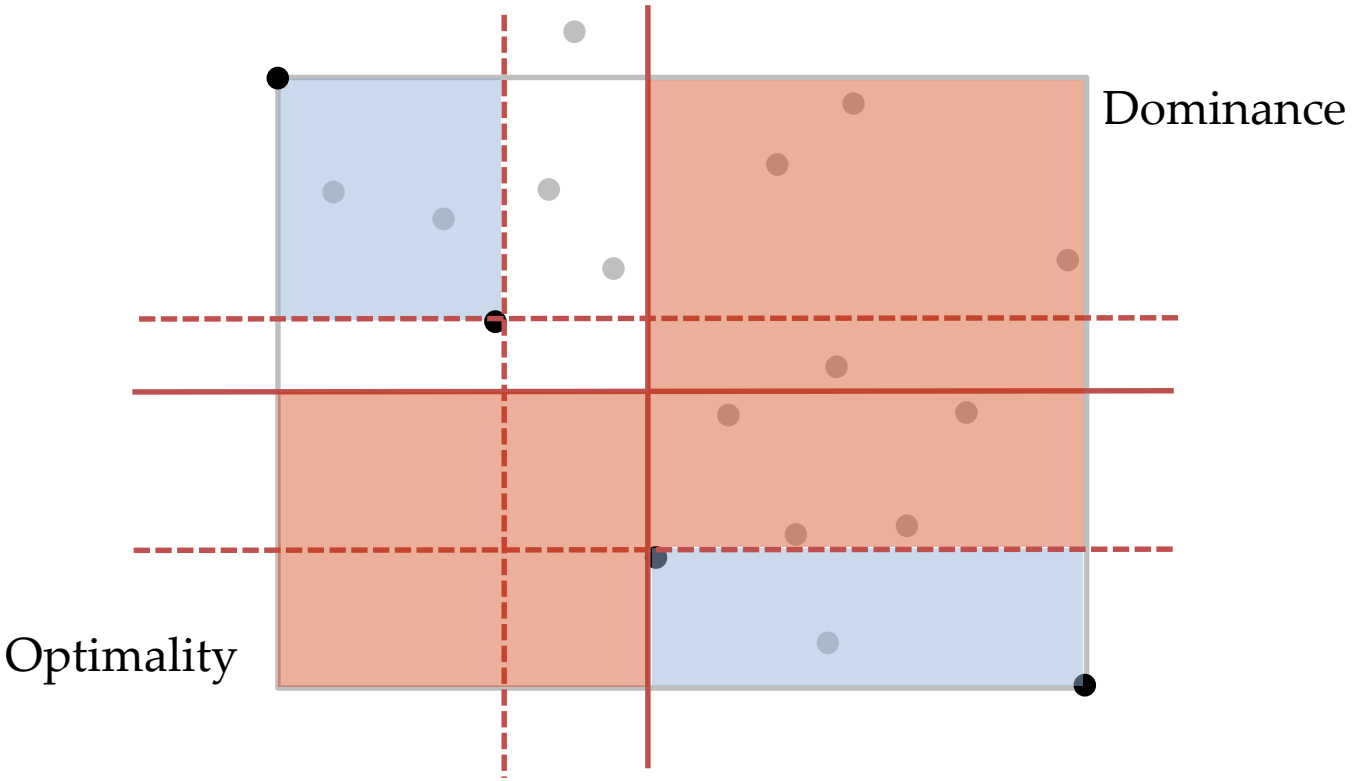
Outer Loop:

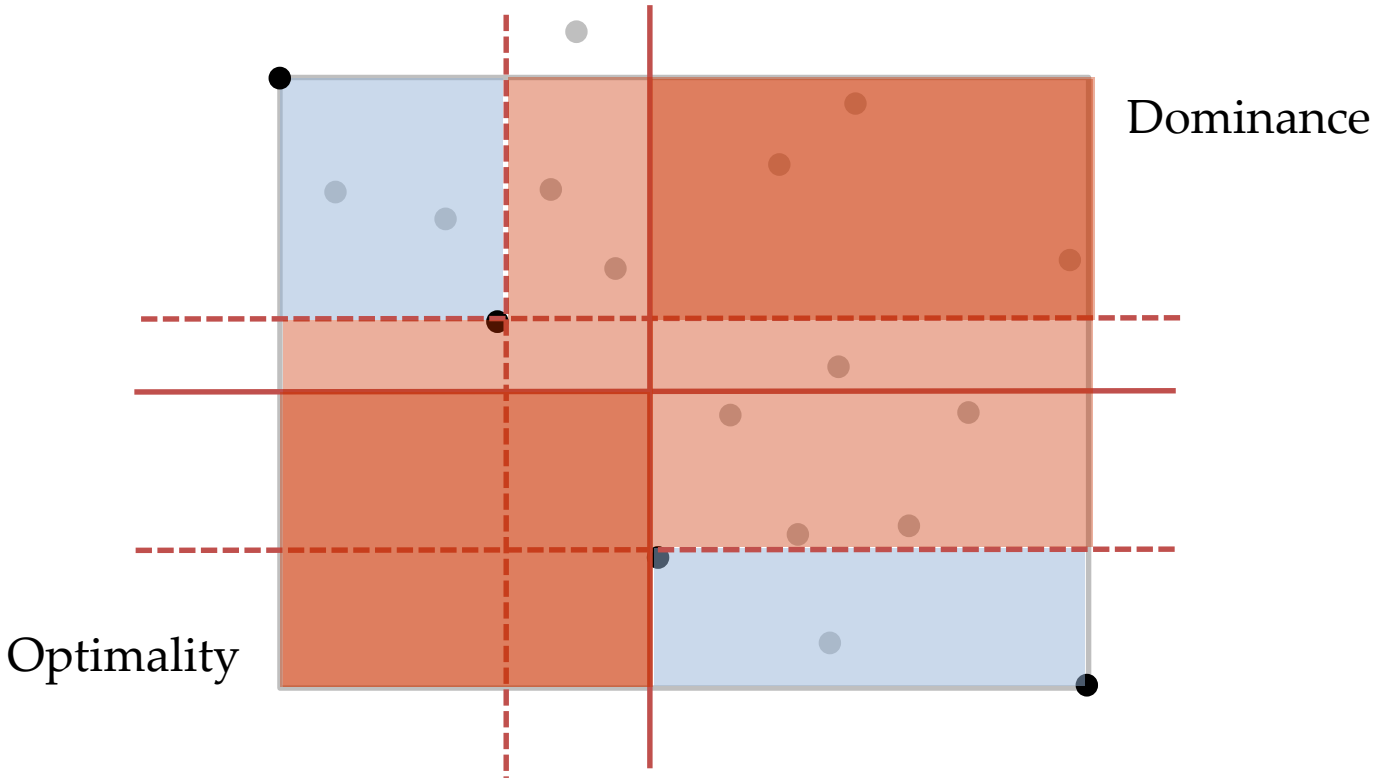
1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line.
4. Lexmin to the left of z*.



Outer Loop:

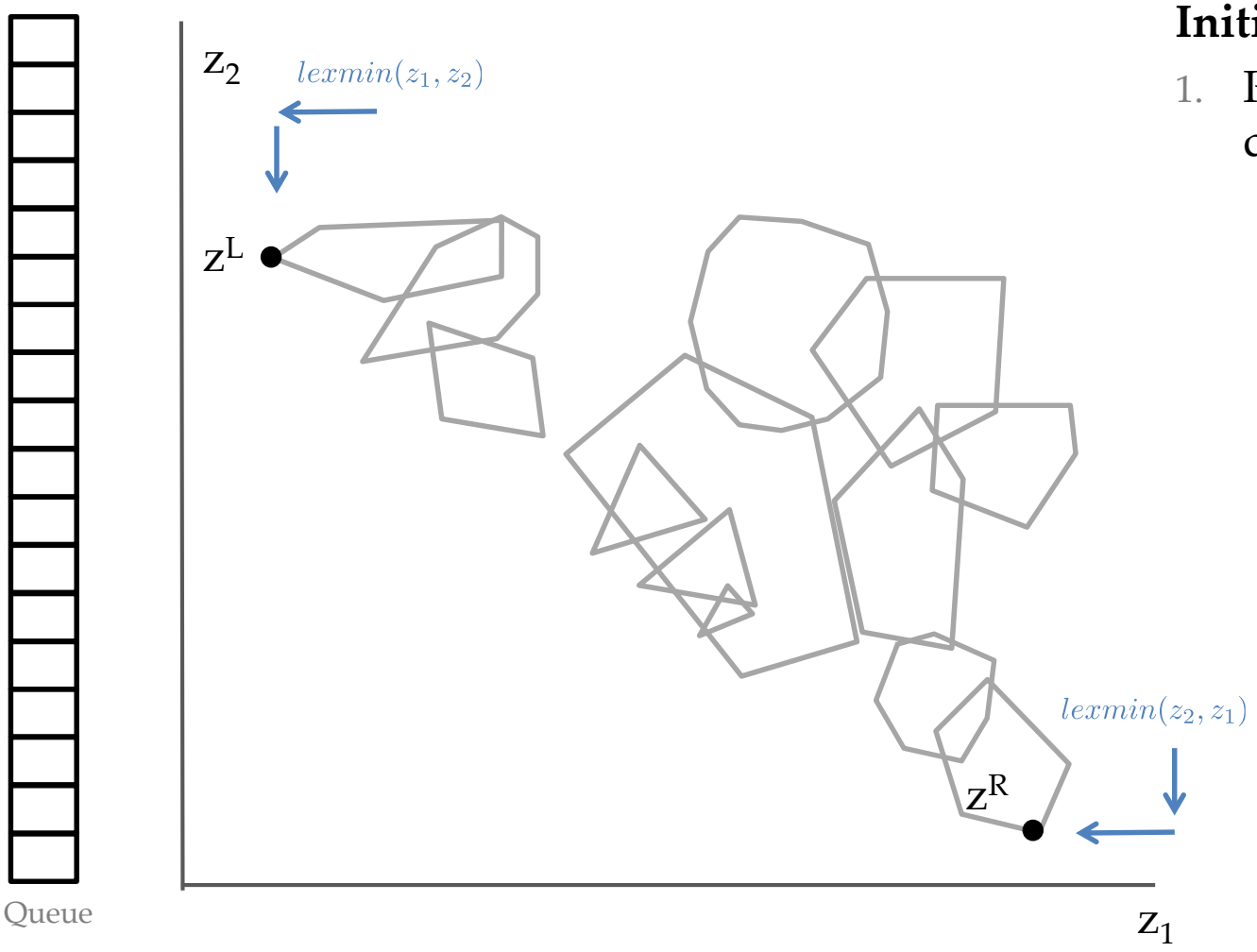
1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line.
4. Lexmin to the left of z*.
5. Add two boxes to queue.


Dominance
Optimality


Dominance
Optimality

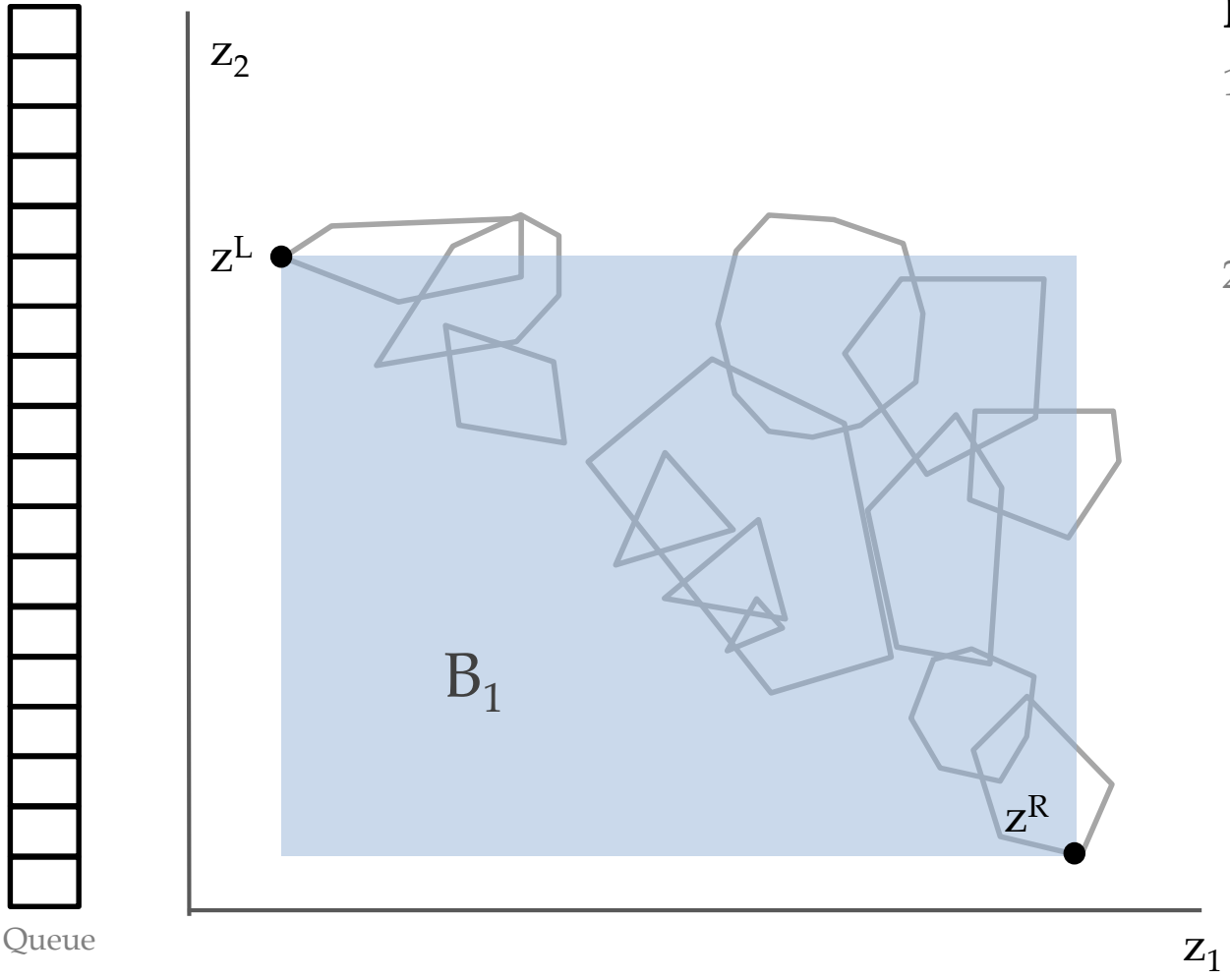
Boxed Line Method

For Biobjective Mixed Integer Programs



Initialization:

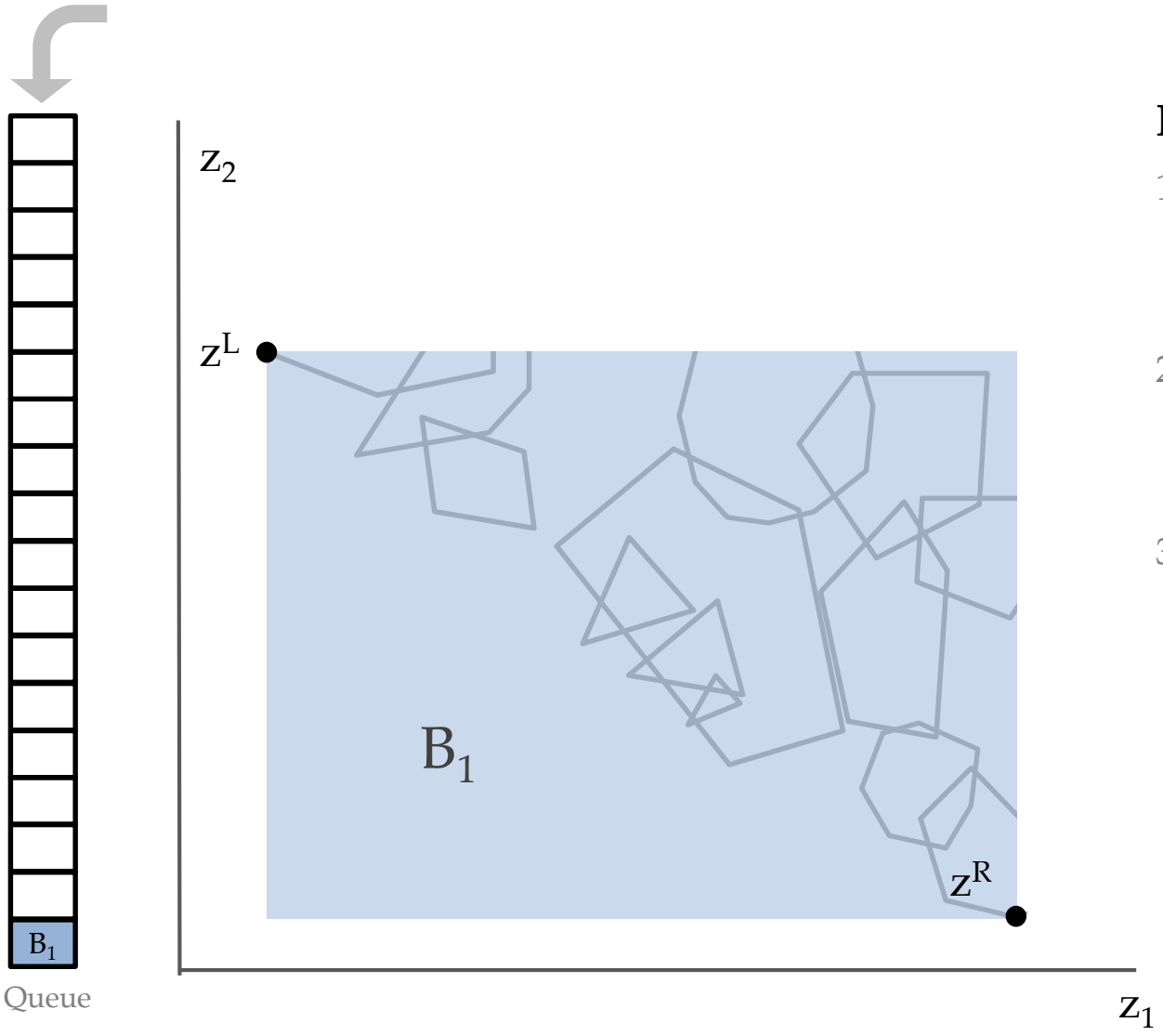
1. Perform two lexmins to discover first two NDPs.



Initialization:

1. Perform two lexmins to discover first two NDPs.

2. These define the corner points of the first box.

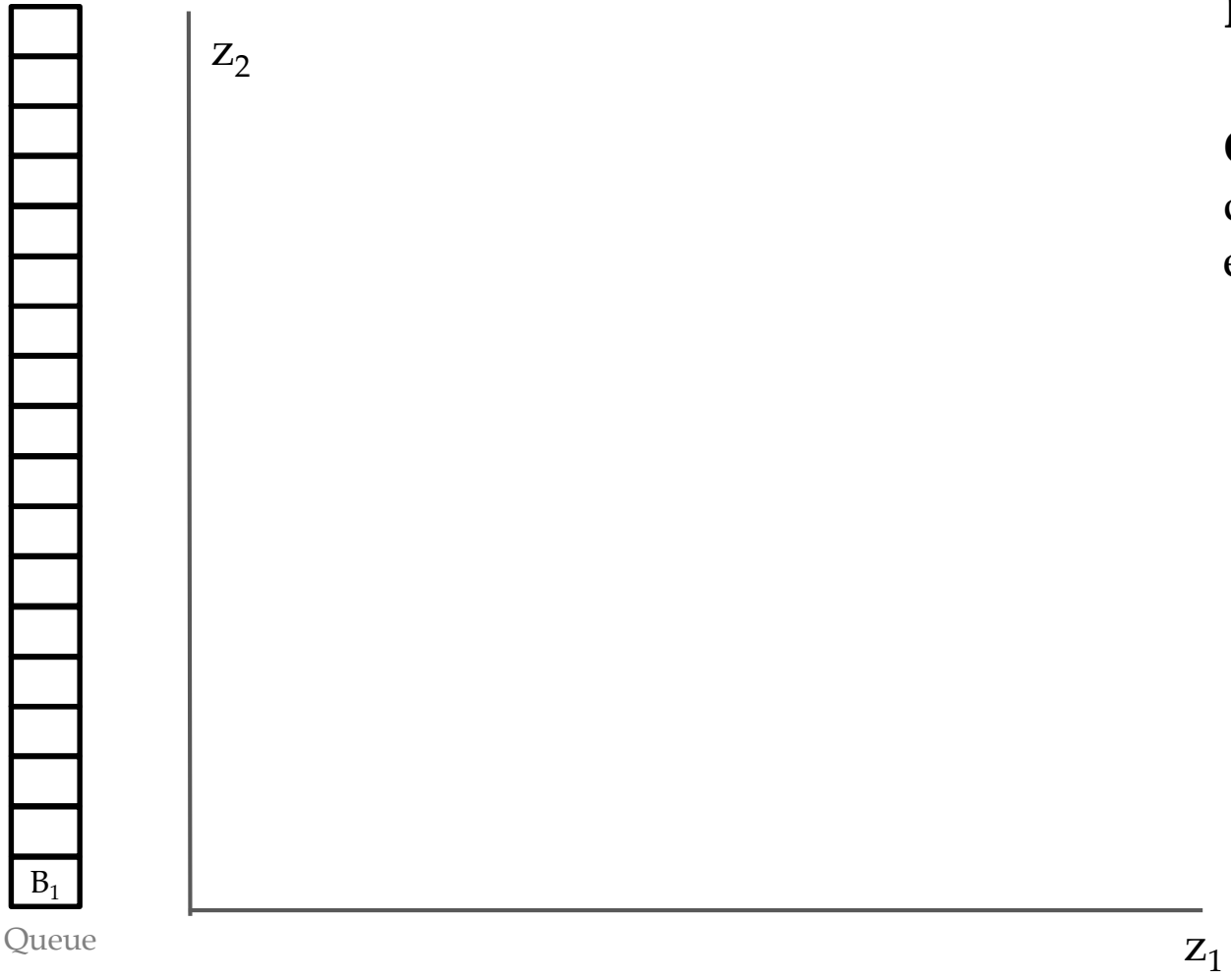


Initialization:

1. Perform two lexmins to discover first two NDPs.
2. These define the corner points of the first box.
3. If the box is nontrivial, add it to queue.

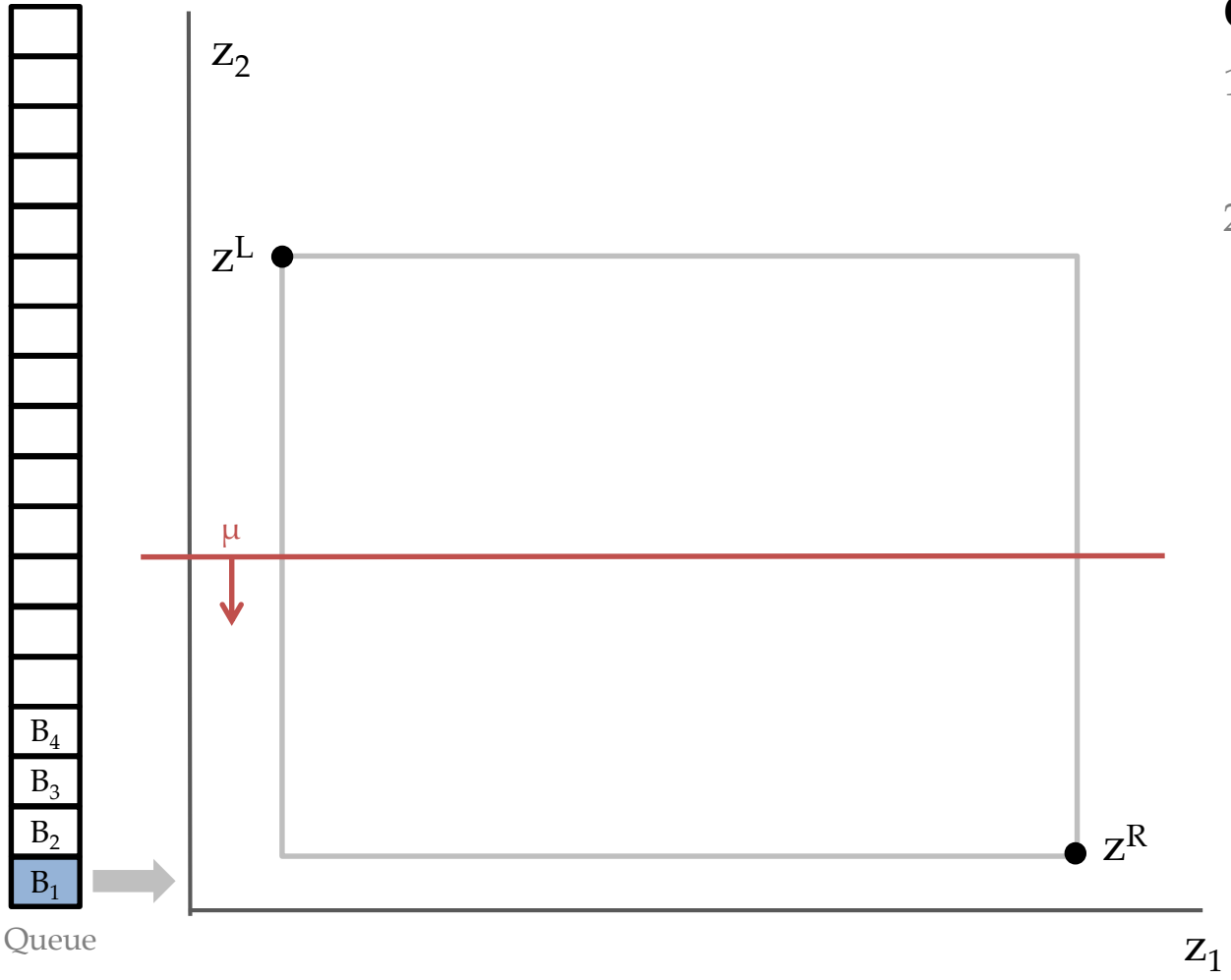
* We always use two distinct NDPs to define a box.

** "Trivially small" boxes are never added to the queue.



Initialization Complete.

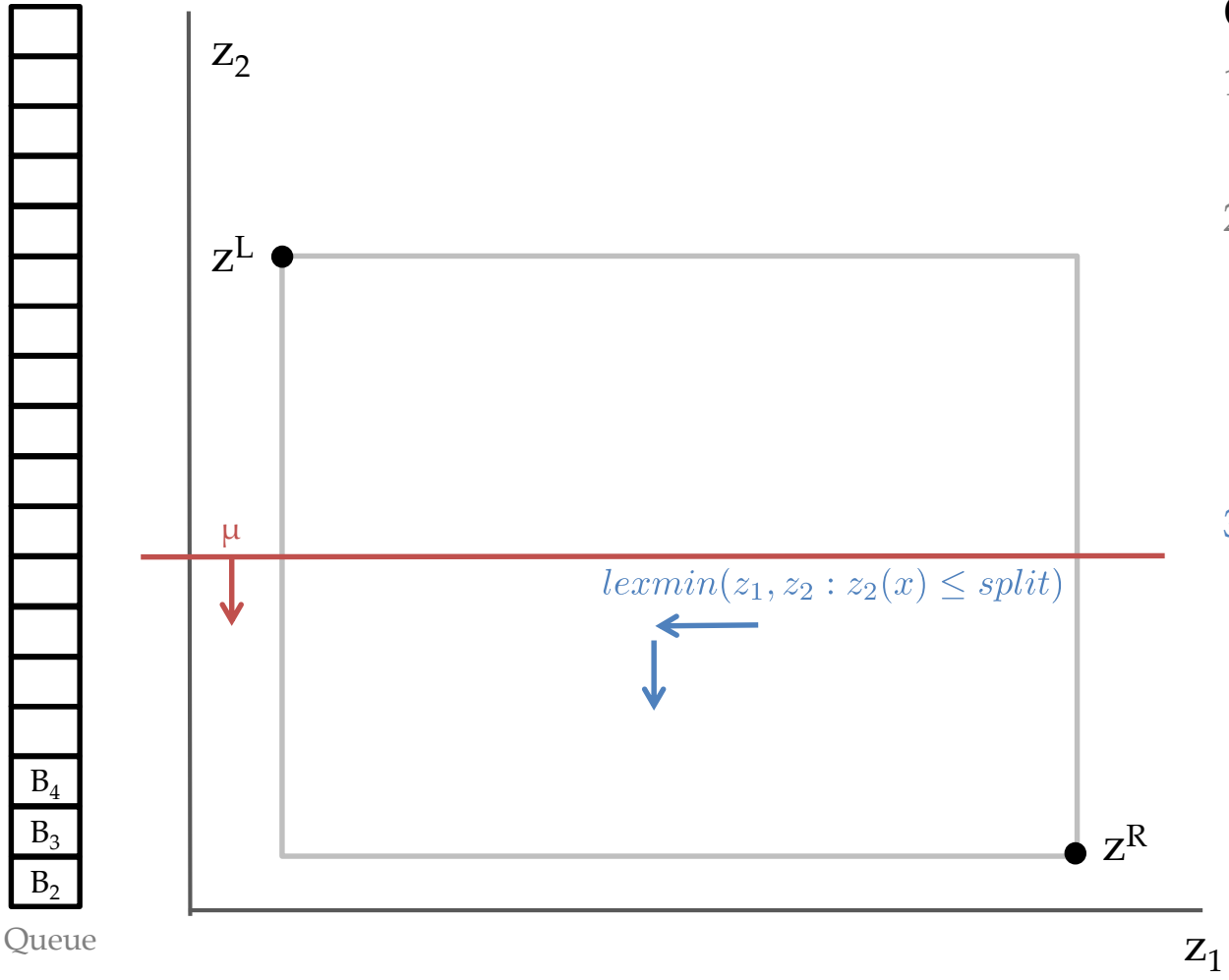
Outer Loop Initiates and continues until queue is empty.



Outer Loop:

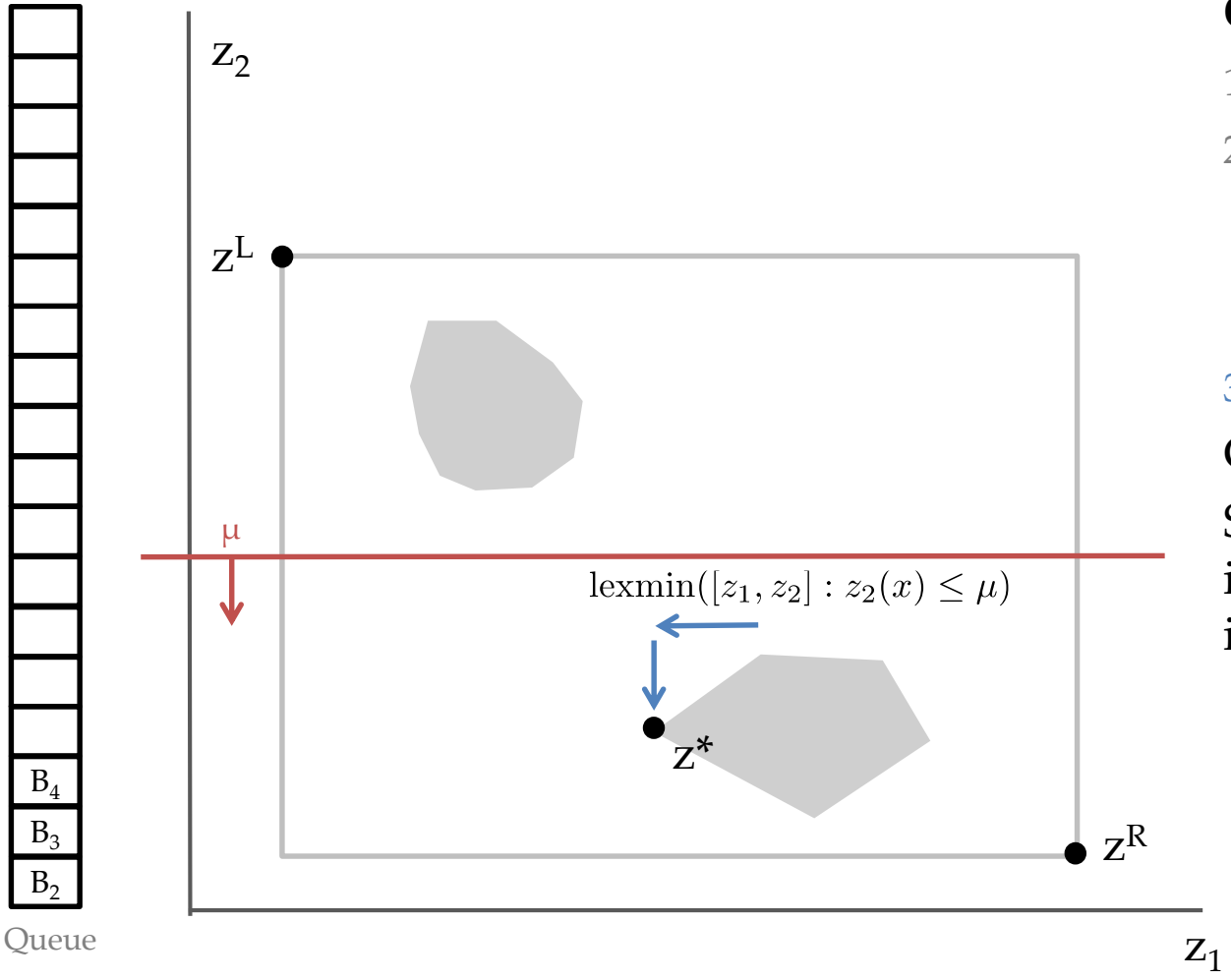
1. Retrieves box from queue.

2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).



Outer Loop:

1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line. (Two cases.)

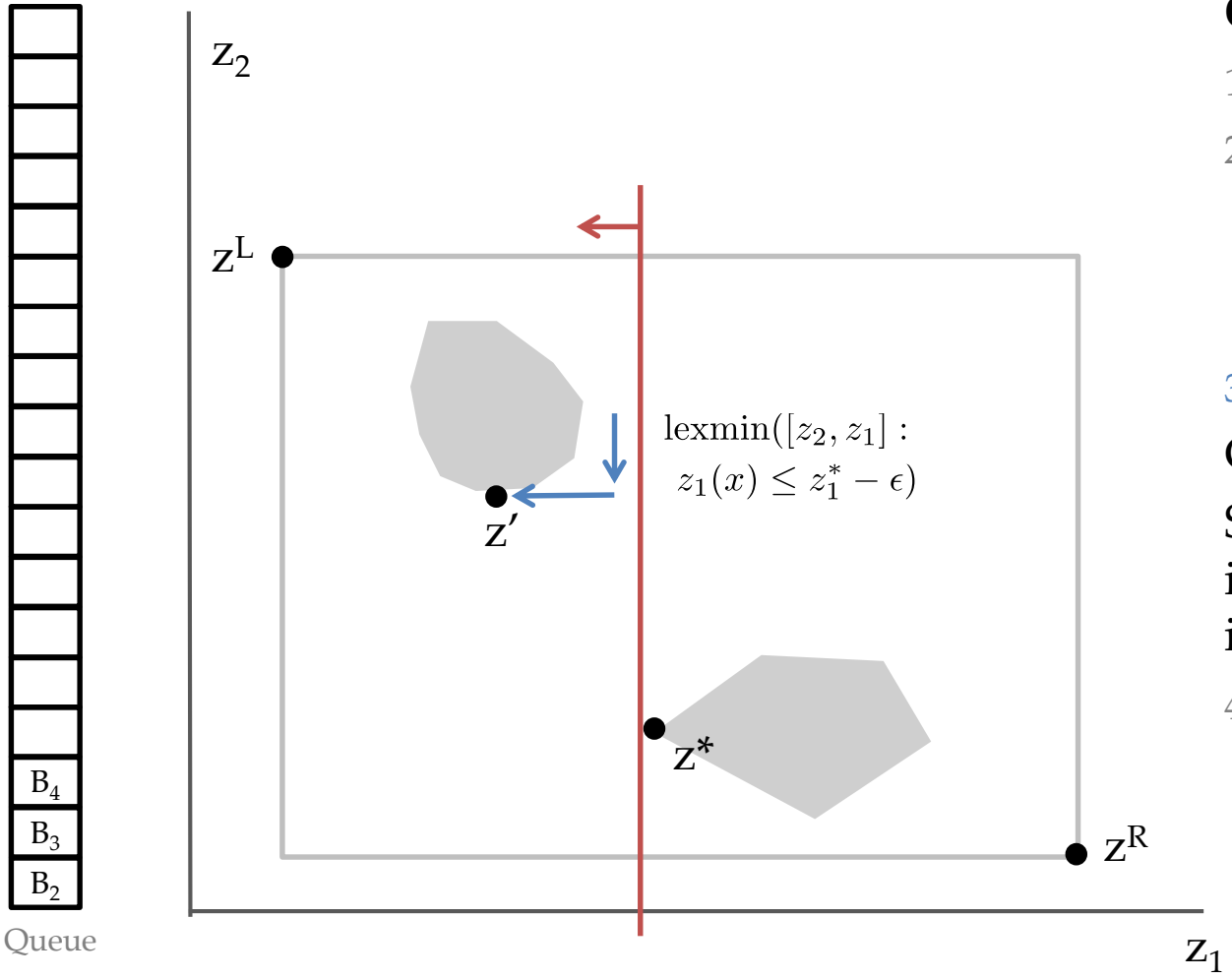


Outer Loop:

1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line.

Case 1:

Split crosses at a **vertical gap** in the frontier, so the solution is not on the split line.



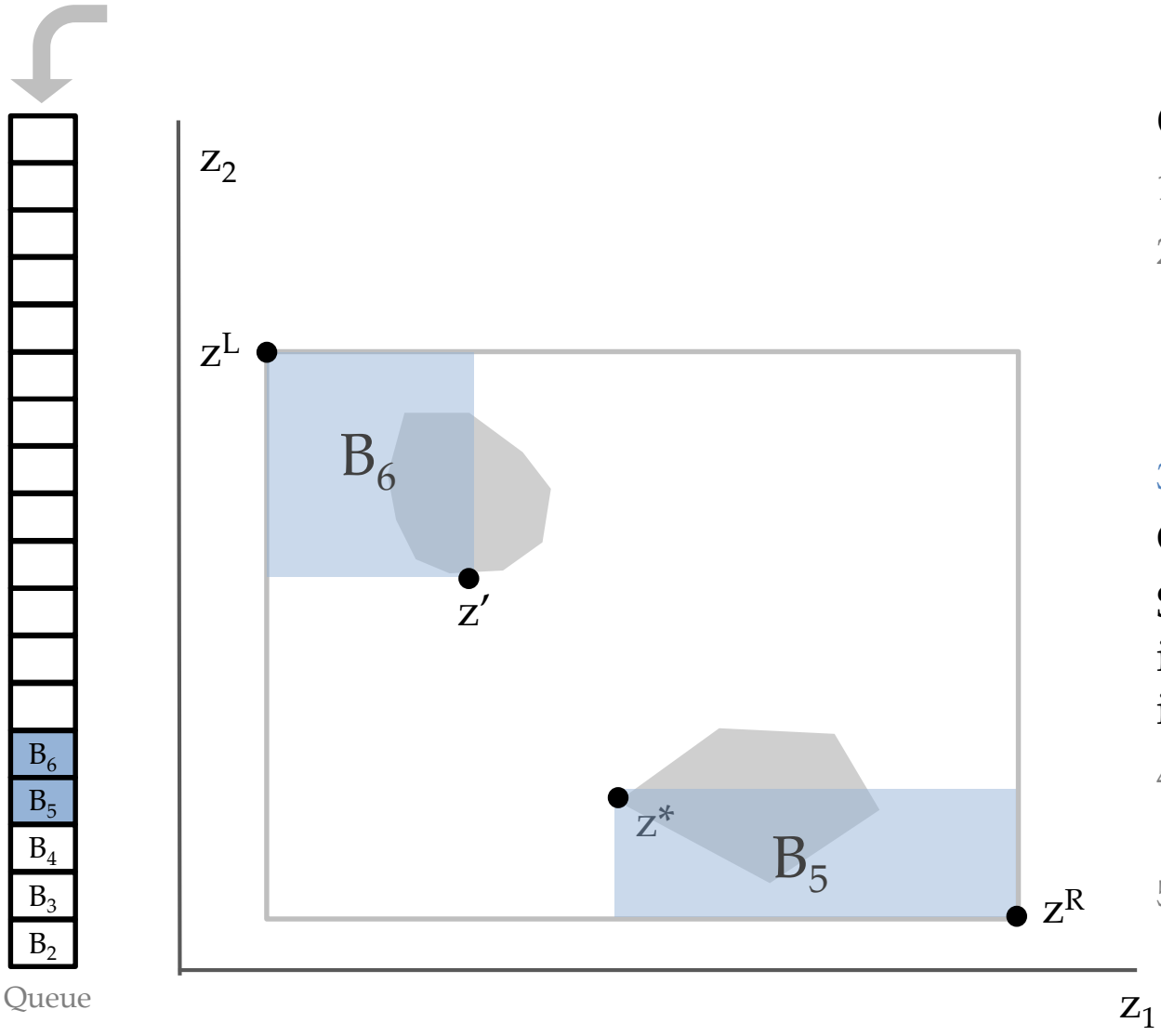
Outer Loop:

1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line.

Case 1:

Split crosses at a vertical gap in the frontier, so the solution is not on the split line.

4. Solve symmetric lexmin to the left of z*.



Outer Loop:

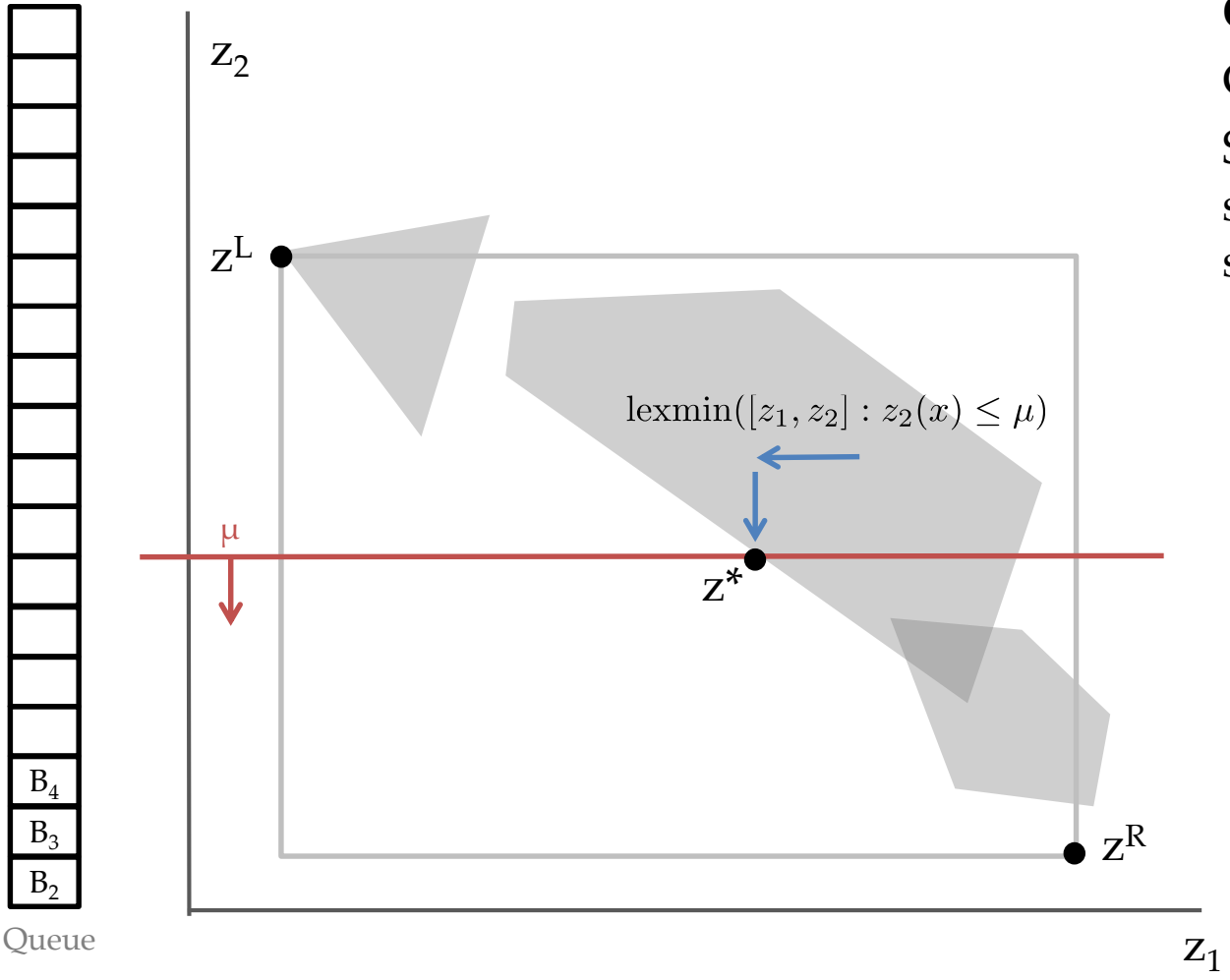
1. Retrieves box from queue.
2. Define a horizontal split line as a constraint (either crosses an ND line segment or vertical gap).
3. Lexmin under split line.

Case 1:

Split crosses at a vertical gap in the frontier, so the solution is not on the split line.

4. Solve symmetric lexmin to the left of z*.
5. Add two boxes to queue.

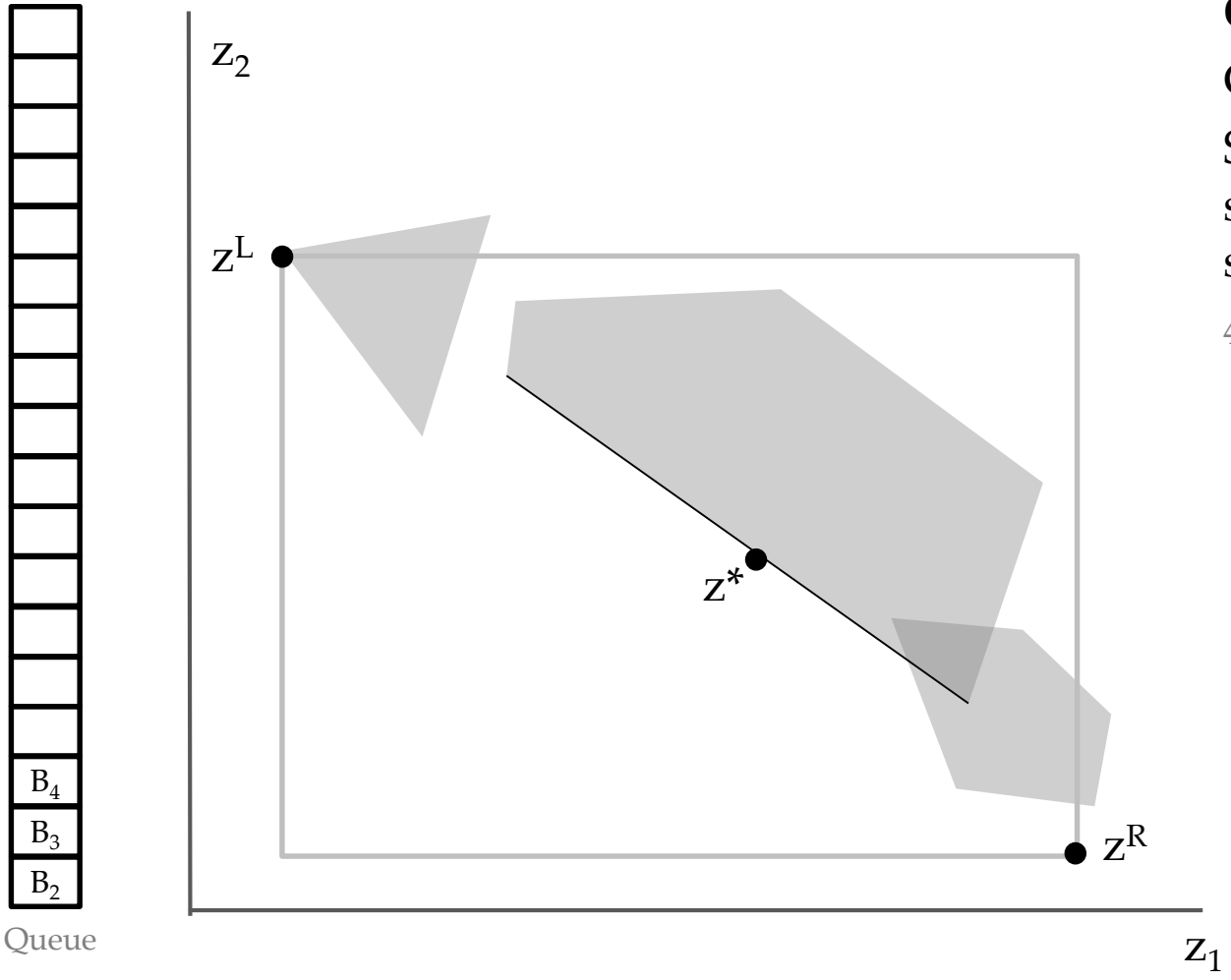
* This case follows the same procedure as Balanced Box Method for BOIP.



Outer Loop:

Case 2:

Split crosses at a ND line segment in the frontier, so the solution is on the split line.

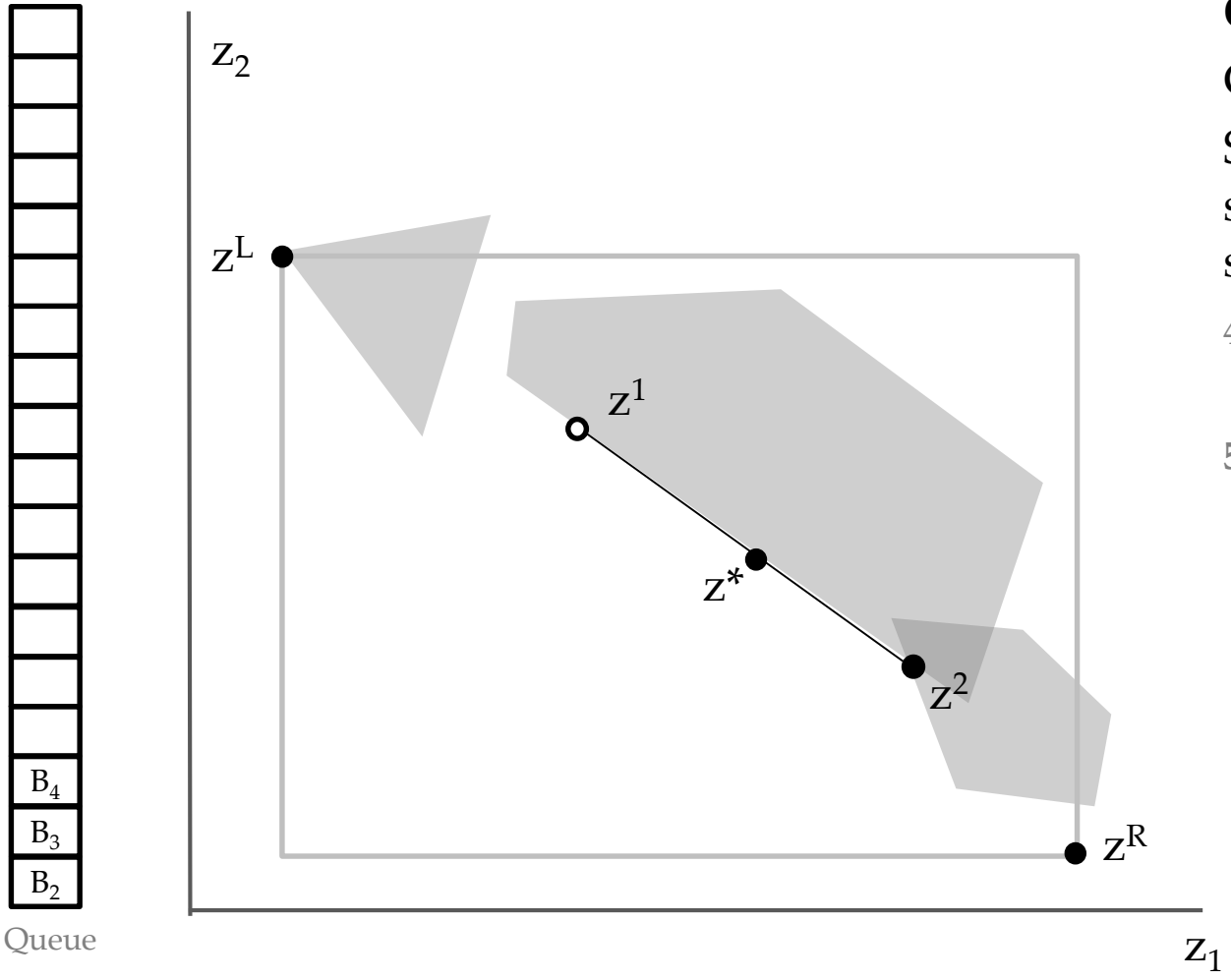


Outer Loop:

Case 2:

Split crosses at a ND line segment in the frontier, so the solution is on the split line.

4. Perform **Line Generation** subroutine.

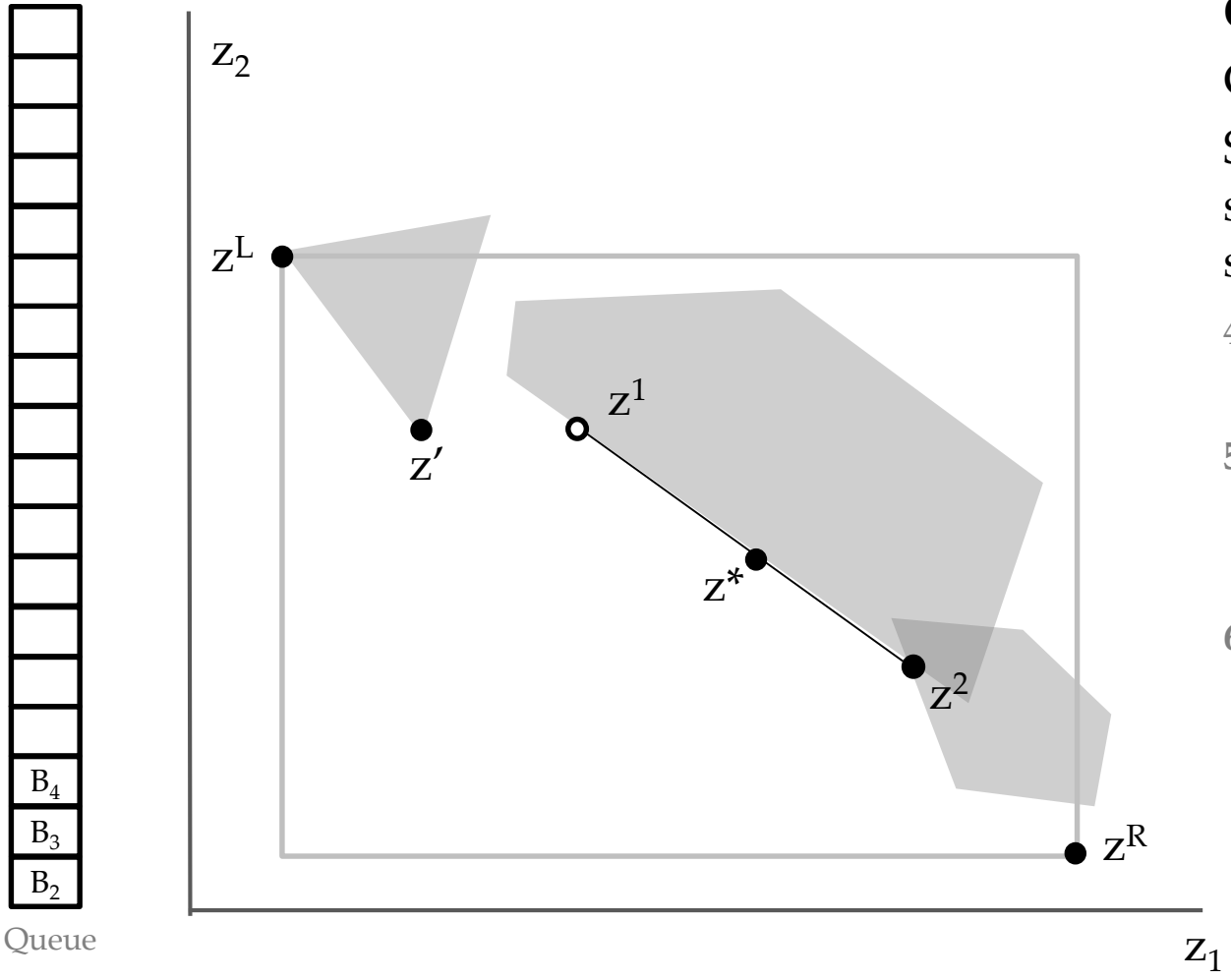


Outer Loop:

Case 2:

Split crosses at a ND line segment in the frontier, so the solution is on the split line.

4. Perform **Line Generation** subroutine.
5. **Inner Loop** refines line segment and returns ND portion.

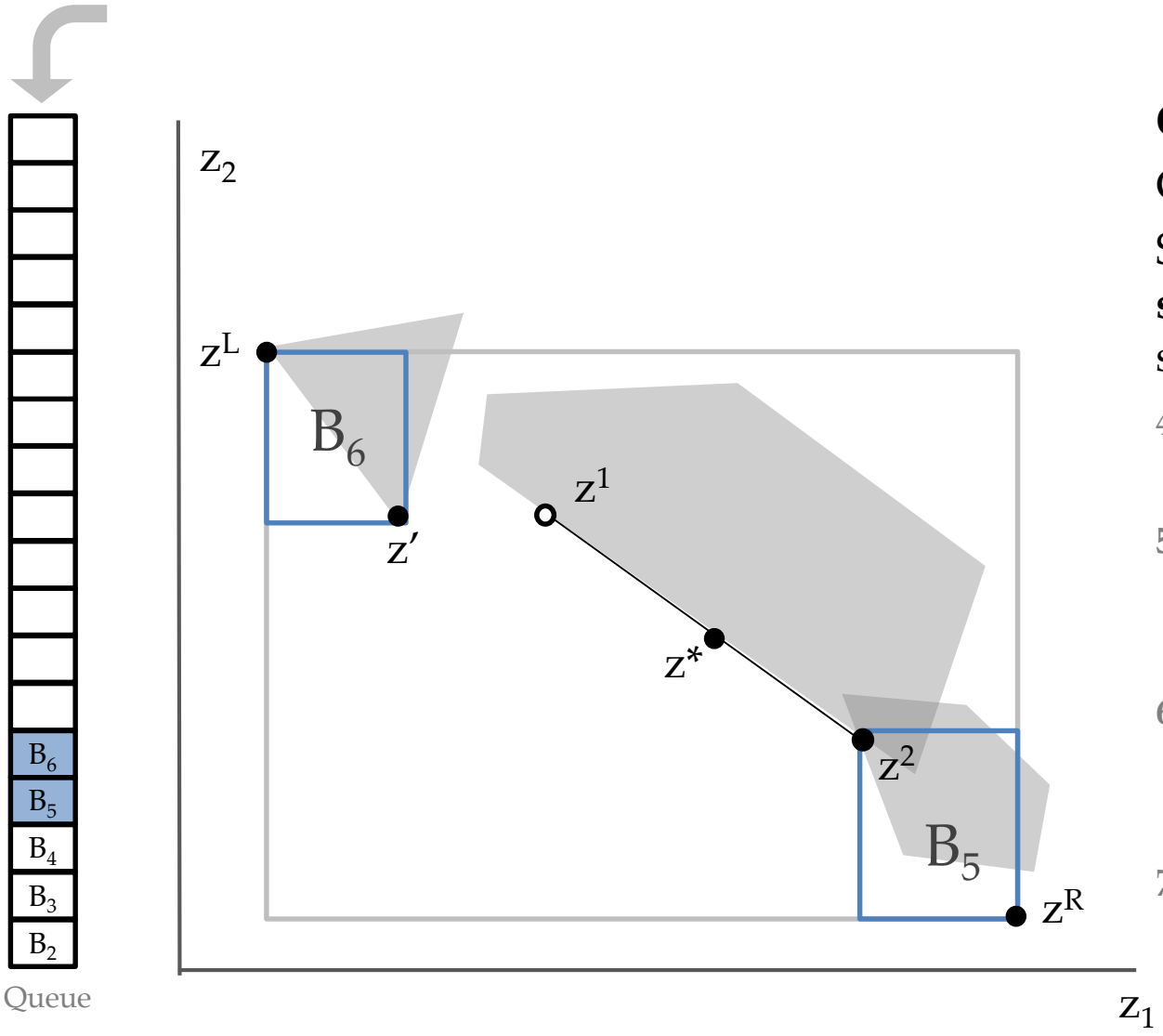


Outer Loop:

Case 2:

Split crosses at a ND line segment in the frontier, so the solution is on the split line.

4. Perform **Line Generation** subroutine.
5. **Inner Loop** refines line segment and returns ND portion.
6. **Inner Loop** also returns the NDP that dominates any open endpoint.

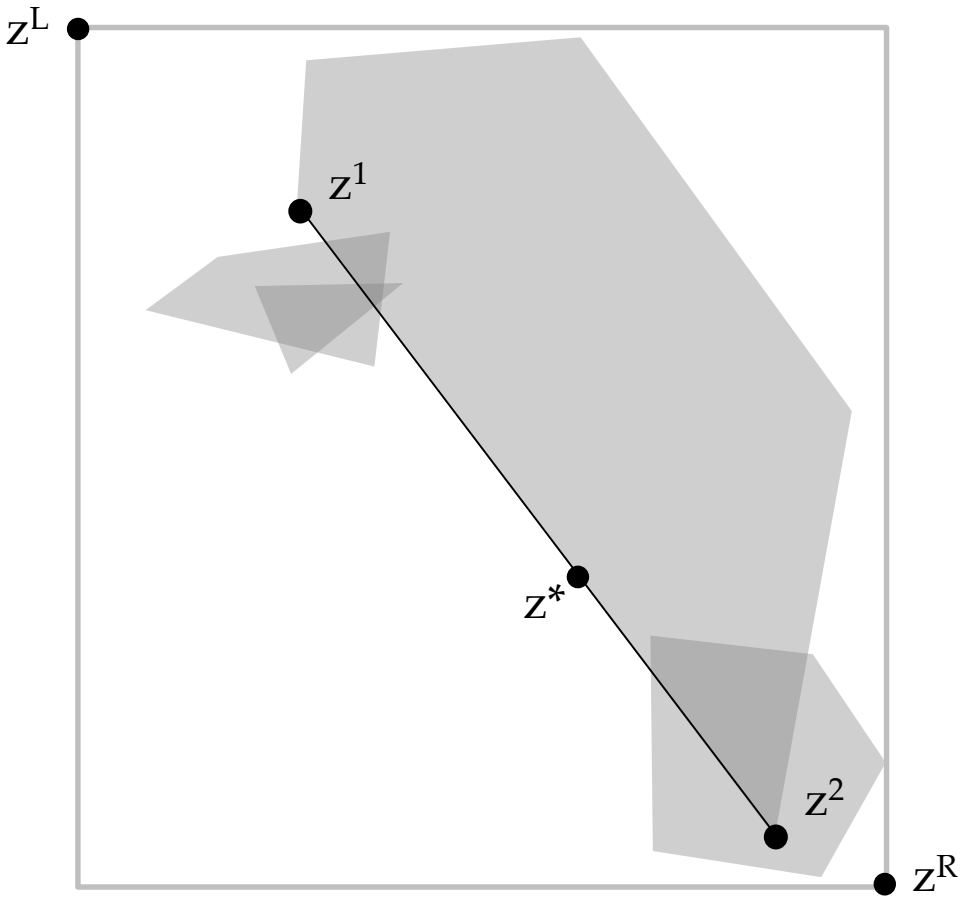


Outer Loop:

Case 2:

Split crosses at a **ND line segment** in the frontier, so the solution is on the split line.

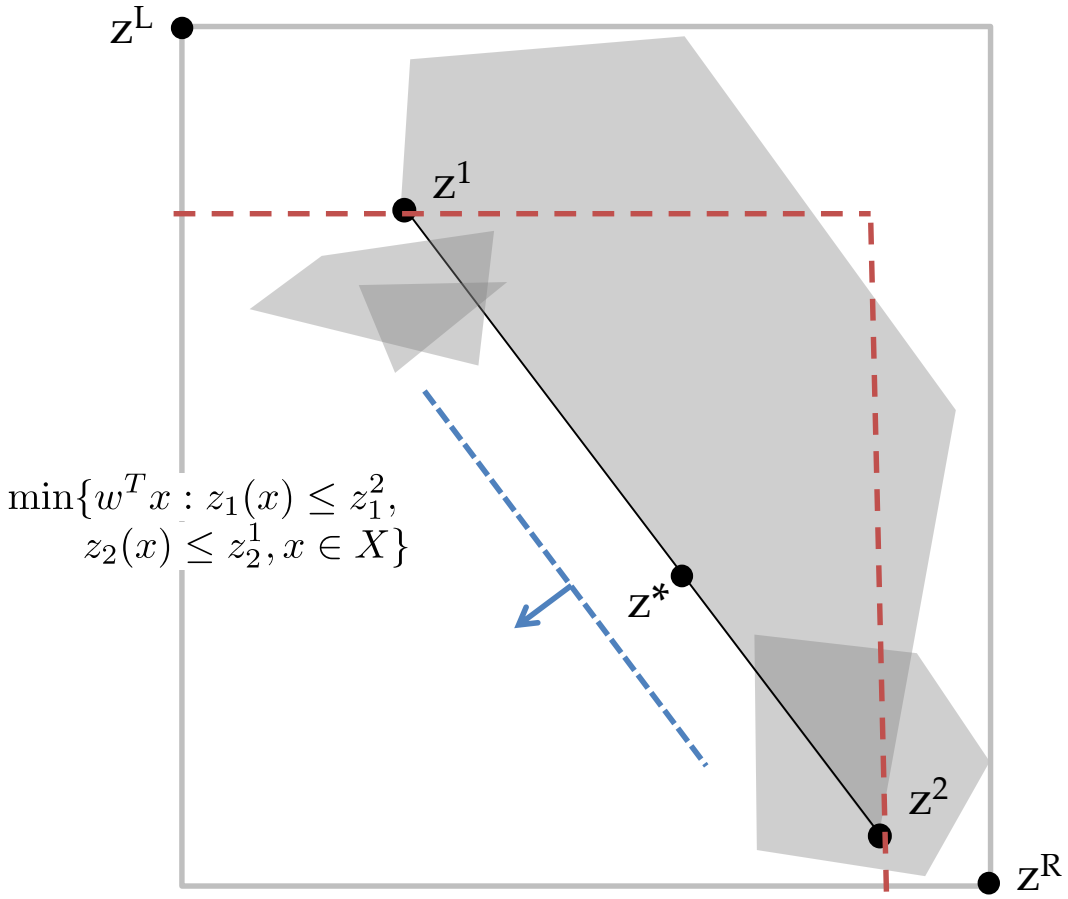
4. Perform Line Generation subroutine.
5. **Inner Loop** refines line segment and returns ND portion.
6. **Inner Loop** also returns the NDP that dominates any open endpoint.
7. Add $L(z^1,z^2)$ to the approximation, and add two boxes to queue.



Inner Loop:

1. **Line Generation** provides a full line segment from the integer frontier s.t.

$$z^* \in L(z^1, z^2)$$

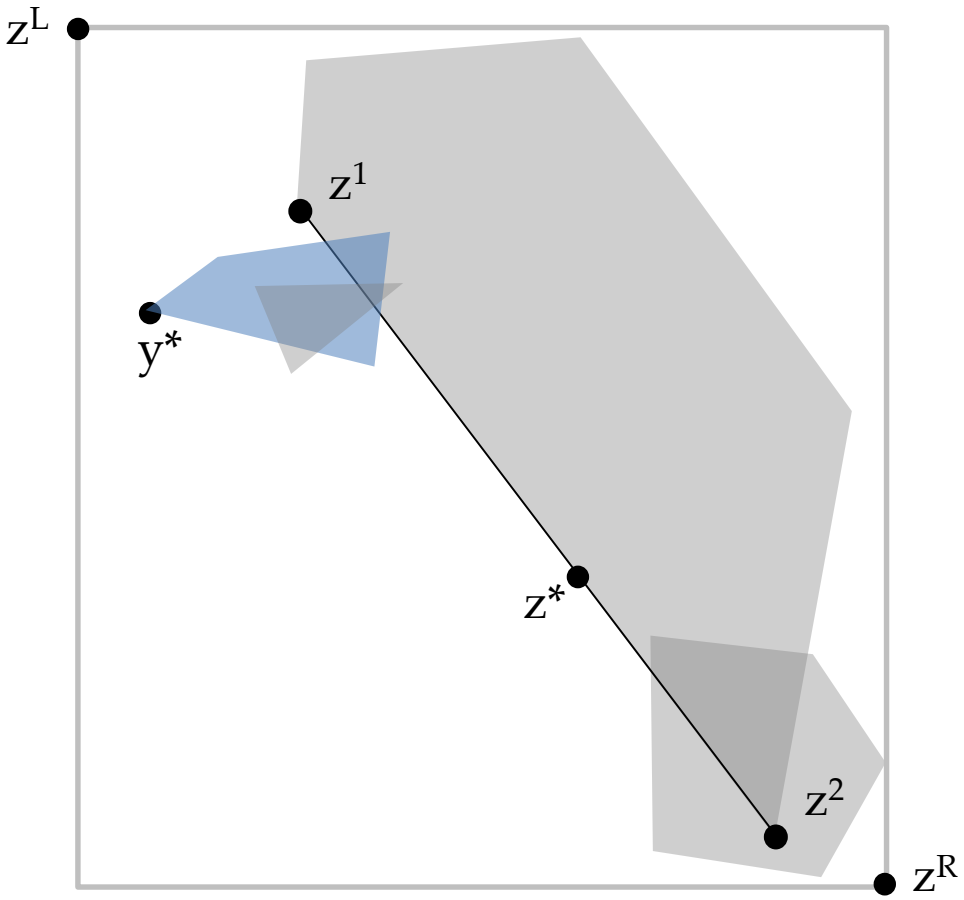


Inner Loop:

1. **Line Generation** provides a full line segment from the integer frontier s.t.

$$z^* \in L(z^1, z^2)$$

2. Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)



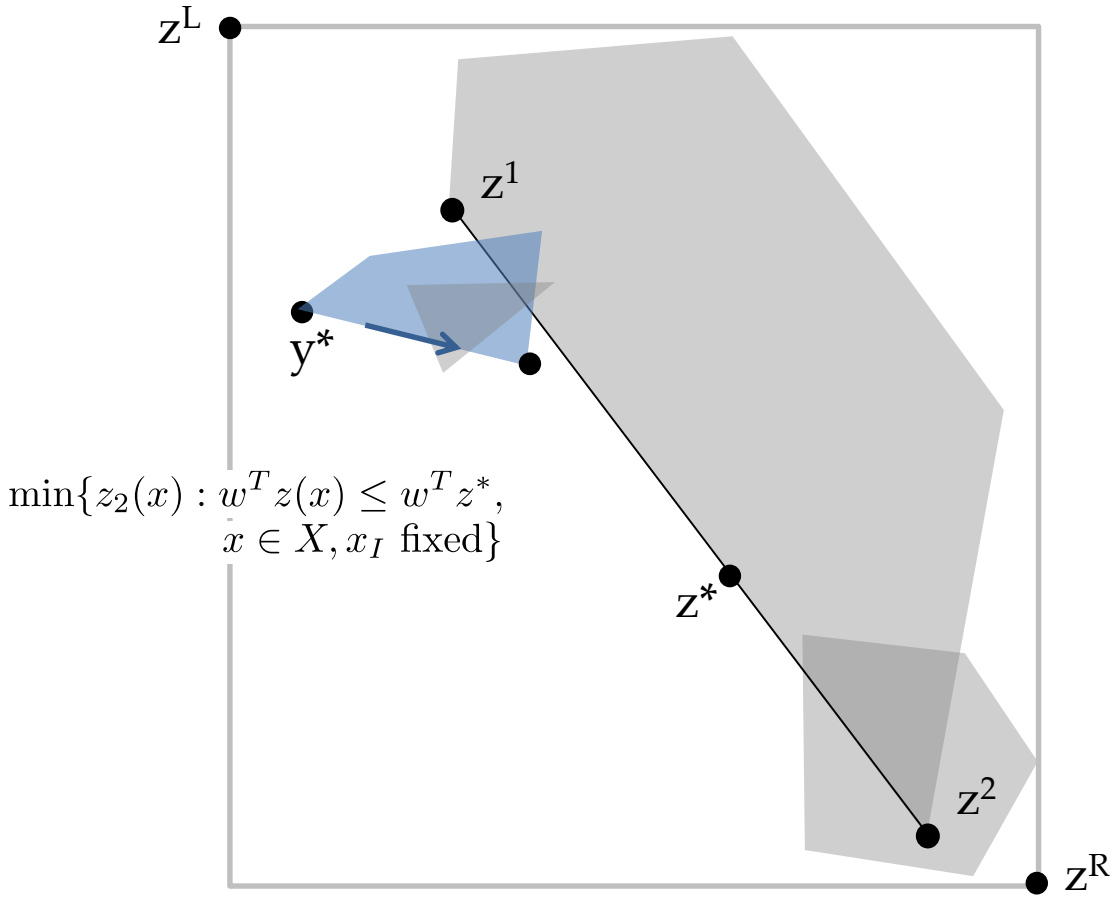
Inner Loop:

1. **Line Generation** provides a full line segment from the integer frontier s.t.

$$z^* \in L(z^1, z^2)$$

2. Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

Case 1: NDP y* dominates line segment.

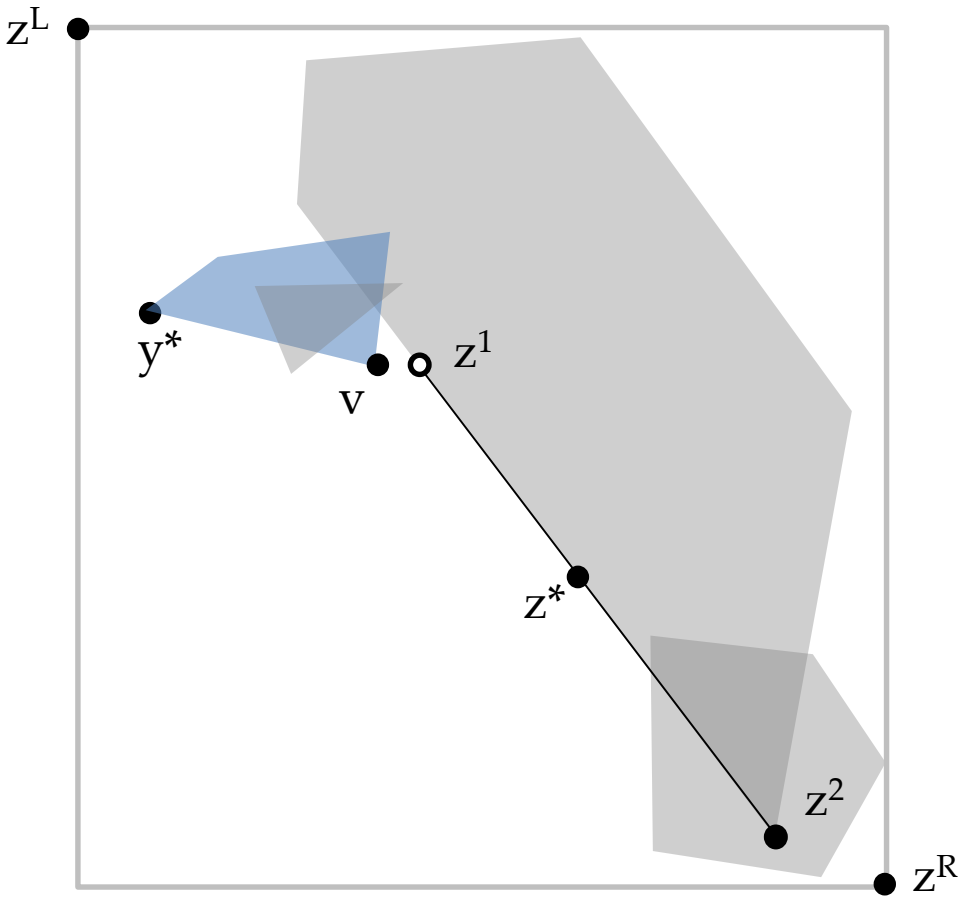


Inner Loop:

1. **Line Generation** provides a full line segment from the integer frontier s.t.
$$z^* \in L(z^1, z^2)$$
2. Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

Case 1: NDP y* dominates line segment

1. Traverse the integer frontier of the NDP towards z* (LP).

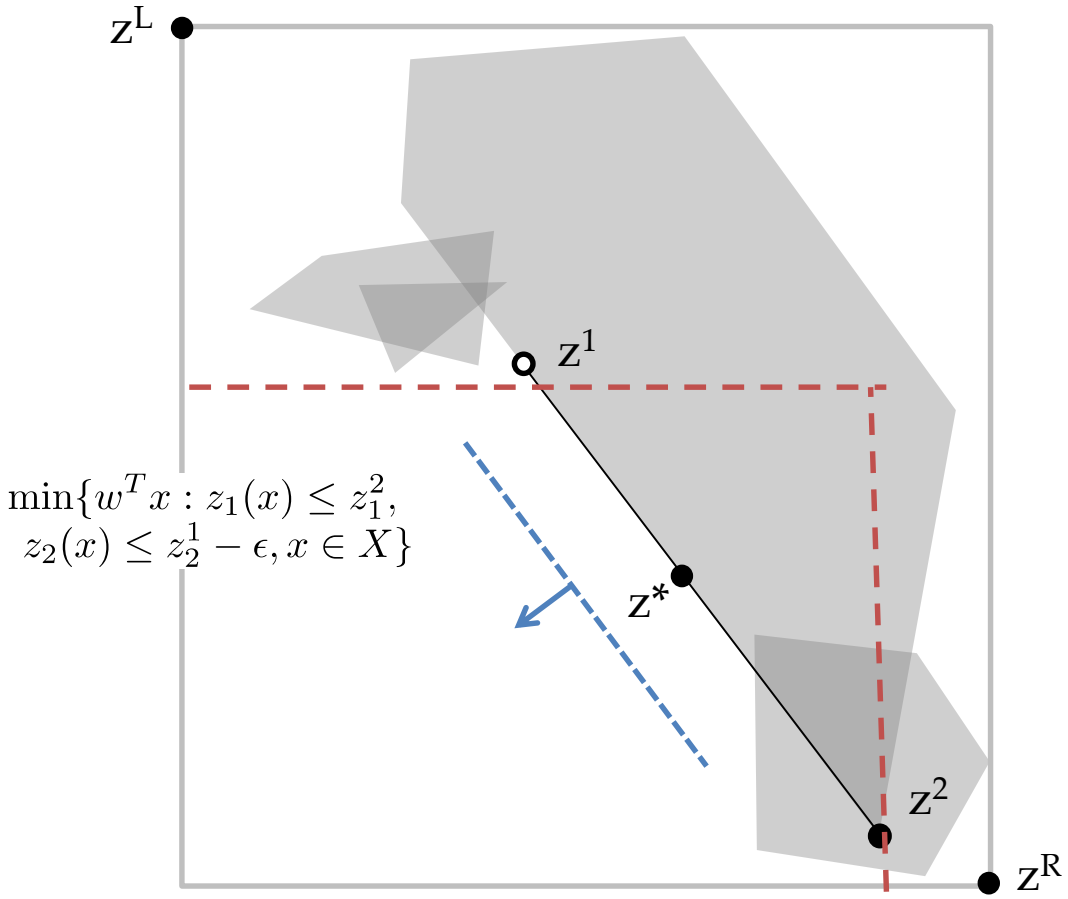


Inner Loop:

1. **Line Generation** provides a full line segment from the integer frontier s.t.
$$z^* \in L(z^1, z^2)$$
2. Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

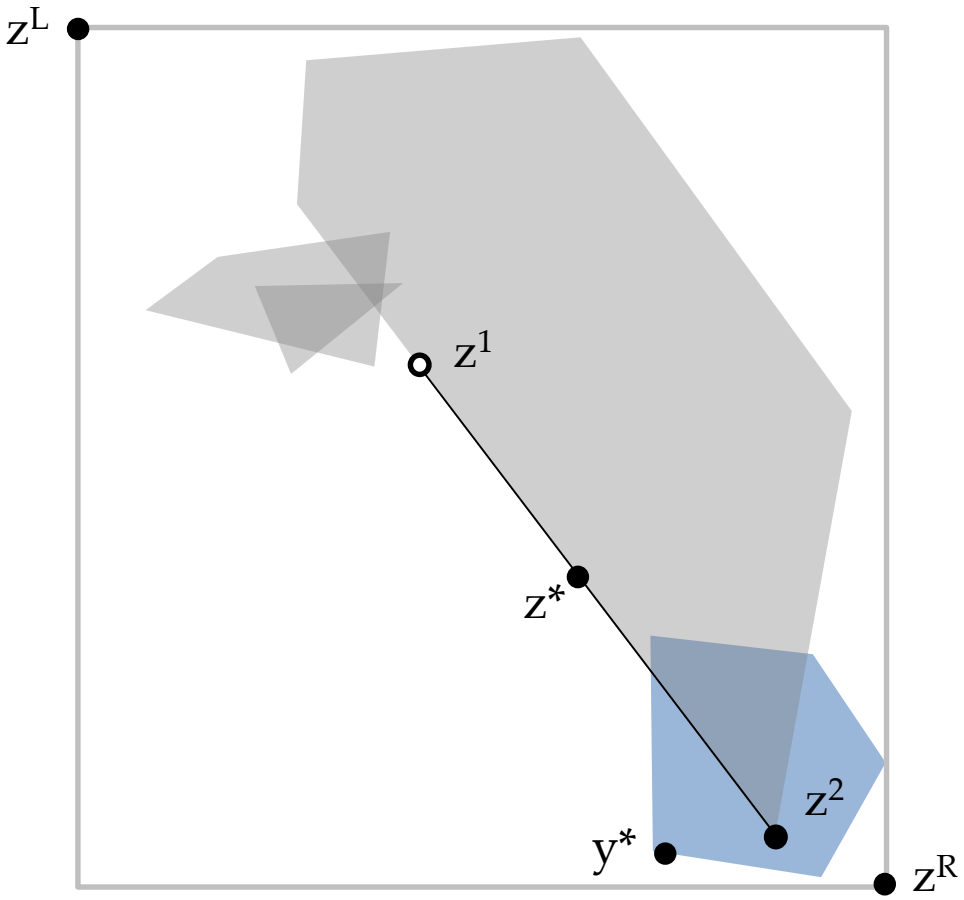
Case 1: NDP y* dominates line segment

1. Traverse the integer frontier of the NDP towards z* (LP).
2. Update endpoint location based on v. If $v \in L(z^1, z^2)$, then endpoint is closed, else endpoint is open.



Inner Loop:

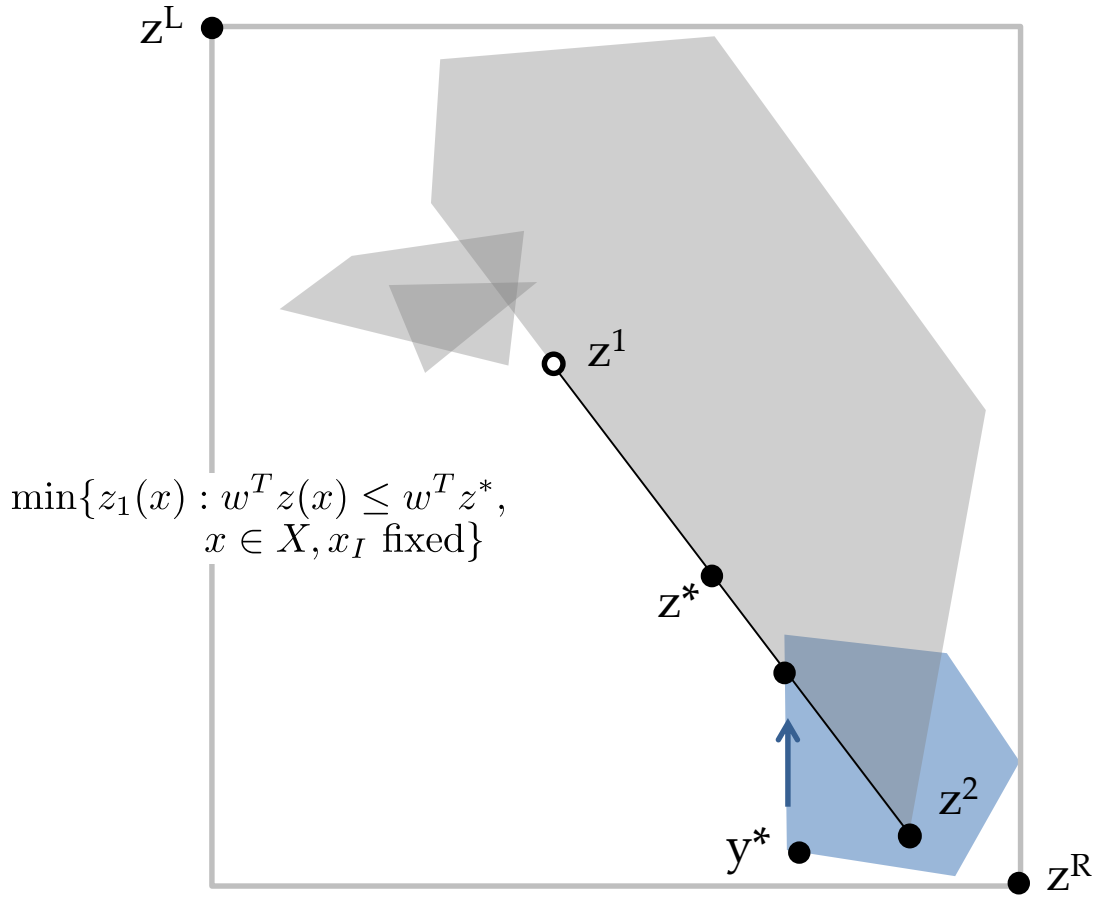
Repeat: Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)



Inner Loop:

Repeat: Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

Case 1: NDP y* dominates line segment

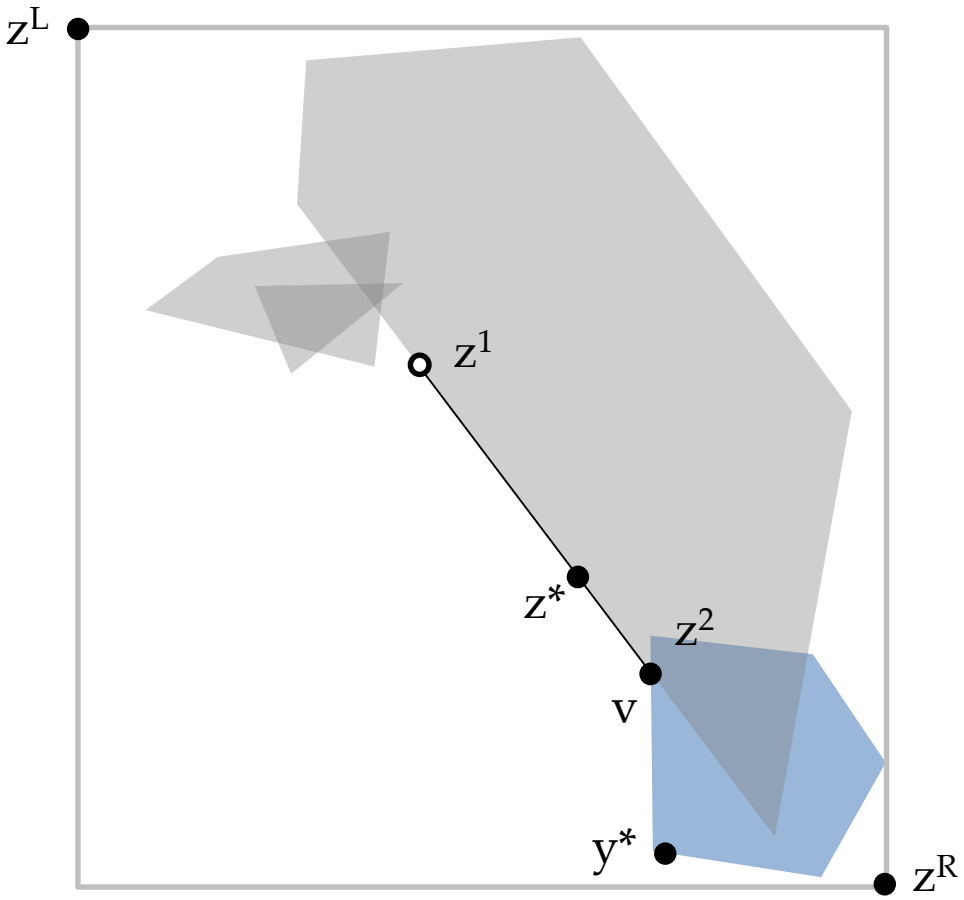


Inner Loop:

Repeat: Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

Case 1: NDP y* dominates line segment

1. Traverse the integer frontier of the NDP towards z* (LP).

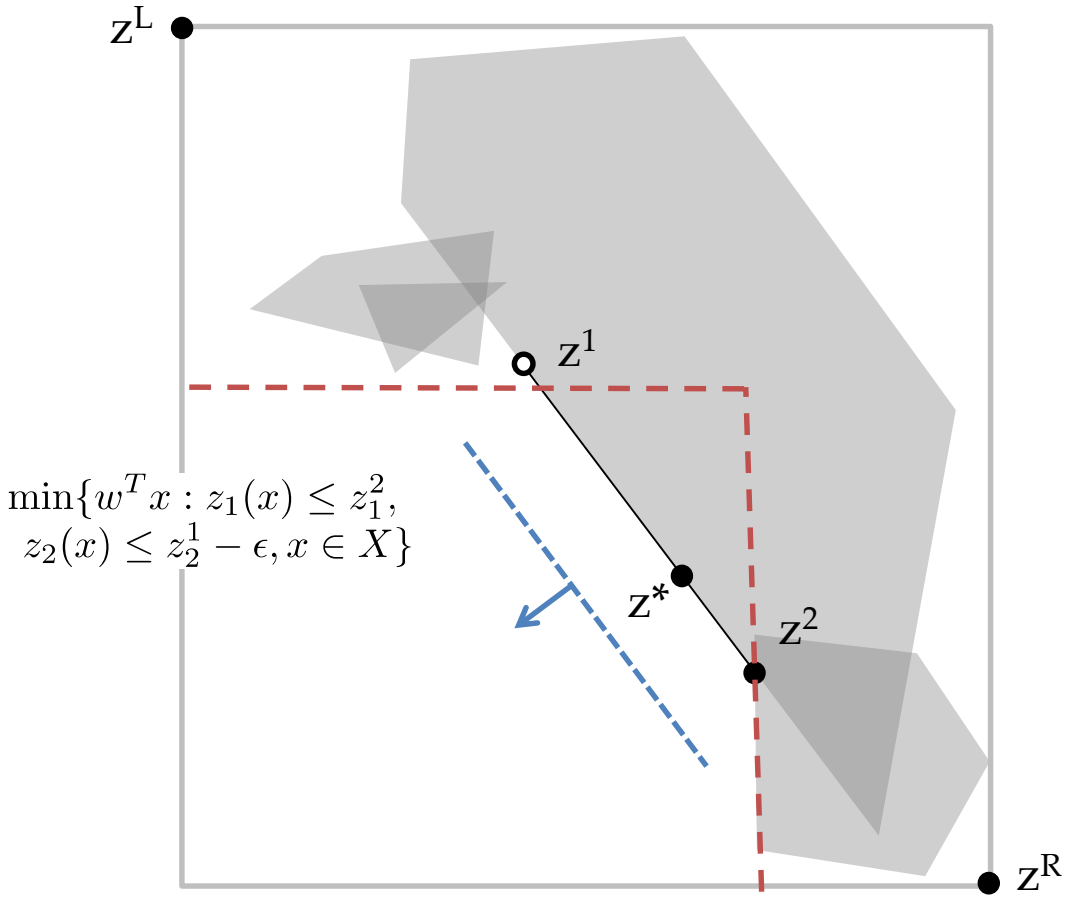


Inner Loop:

Repeat: Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

Case 1: NDP y* dominates line segment

1. Traverse the integer frontier of the NDP towards z* (LP).
2. Update endpoint location based on v. If $v \in L(z^1, z^2)$, then endpoint is closed, else endpoint is open.



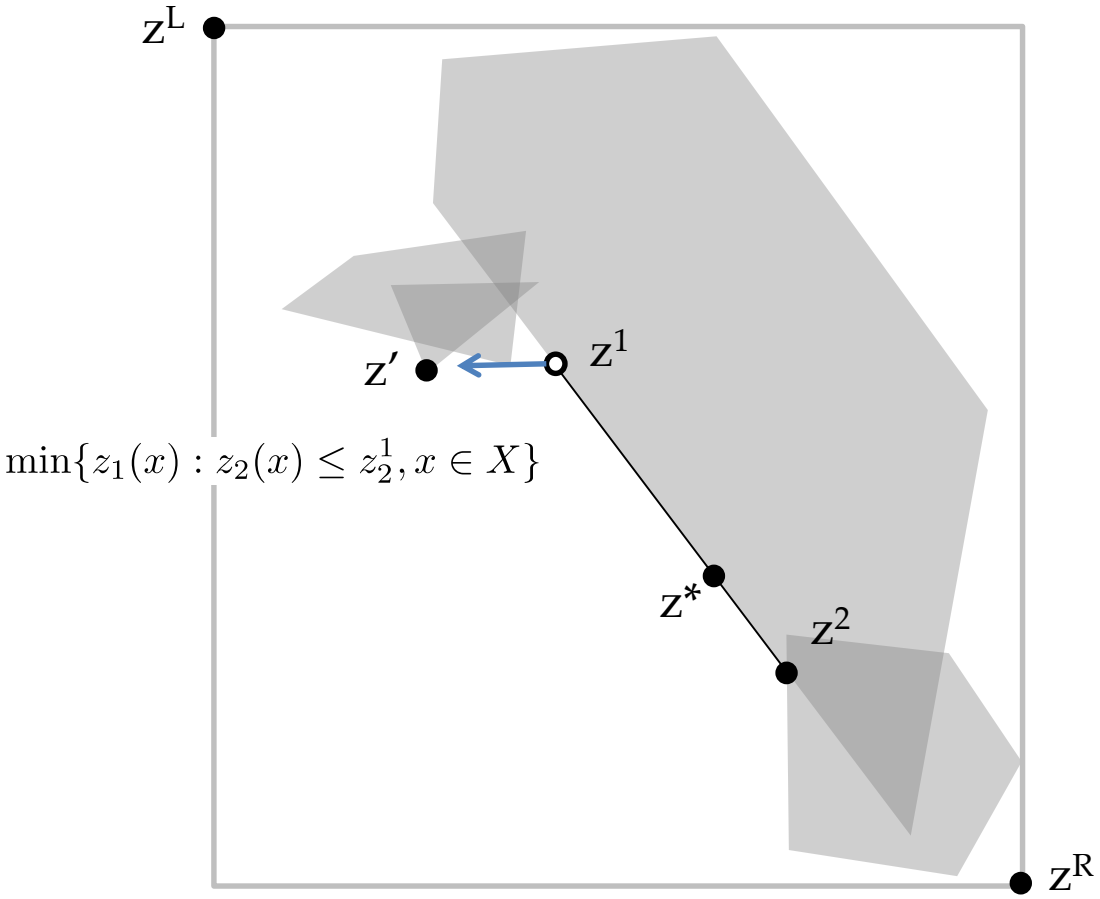
Inner Loop:

Repeat: Solve scalarization IP w.r.t. gradient vector of $L(z^1, z^2)$ to find NDP that dominates it. (Two cases.)

Case 2: NDP y* does NOT dominate line segment.

END LOOP:

$L(z^1, z^2)$ is nondominated



Inner Loop:

Once L(z1,z2) is found to be nondominated:

For each open endpoint, solve an IP to find the NDP that dominates it.

Return to Outer Loop: L(z^1,z^2) and the NDPs that dominate z^1, z^2

* Note that each of the NDPs y* are "forgotten" by the basic method: motivation for the recursive and enhanced methods.

Complexity Results: Basic Method

n = number of NLS in the *strict interior* of a box

g = number of vertical gaps in a box ($g \leq n+1$)

Lexicographic IPs

$$\hat{L}(n) = 3n + 2$$

- Only Outer Loop solves lexicographic IPs
- 1 lexmin solved for each NLS
- 2 lexmin solved for each vertical gap
- $n + 2g \leq 3n + 2$

Single-Objective IPs

$$\hat{S}(n) = \frac{n(n+1)}{2} + 2(n-1)$$

- Only Inner Loop solves single-objective IPs
- At most n scalarized IPs solved to refine line segment
- At most 2 single-objective IPs solved for open endpoints
- Result follows by induction

Complexity Results: Recursive Method

Inner Loop is defined recursively.

Requires (simple) routine to prevent cycling.

Lexicographic IPs

$\hat{L}(n) = 3n + 2$

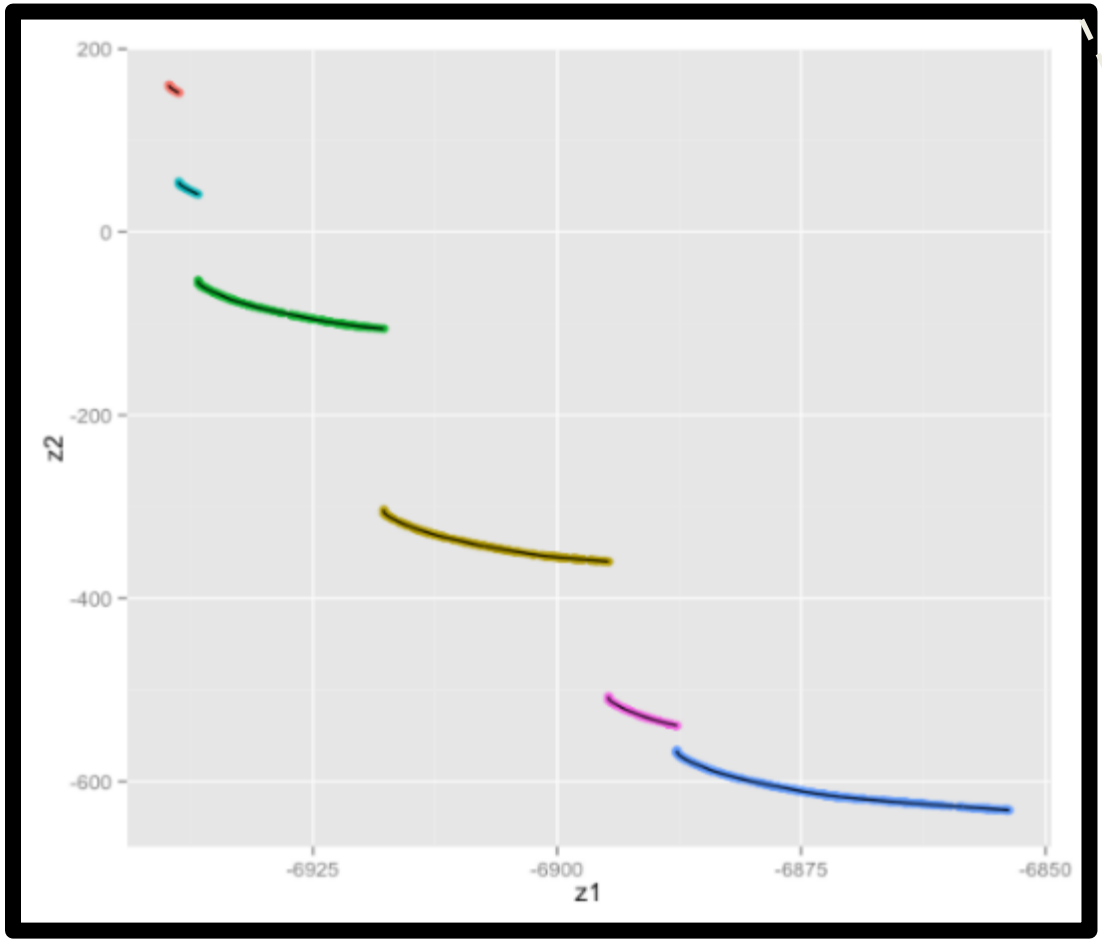
- Same as Basic Method

Single-Objective IPs

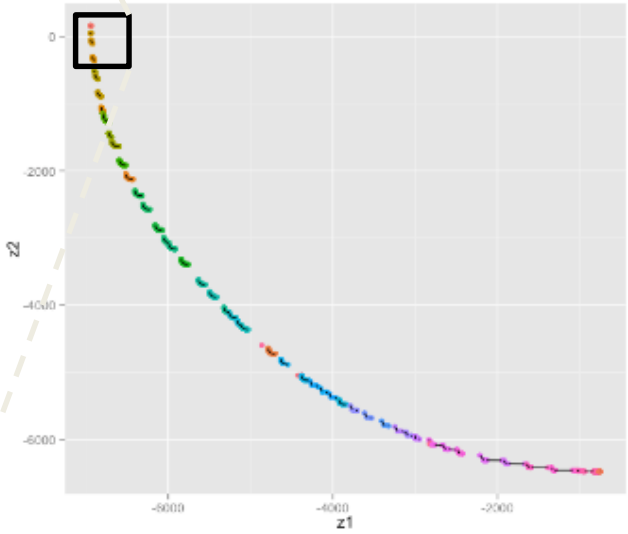
$\hat{S}(n) = 2n - 1$

- Every scalarization either finds an NDP or confirms a line segment is nondominated
- At most 2 scalarization IPs for all NLS, except the first one (first NDP must be found by lexicographic IP)

Enhancement: Same Integer Solution

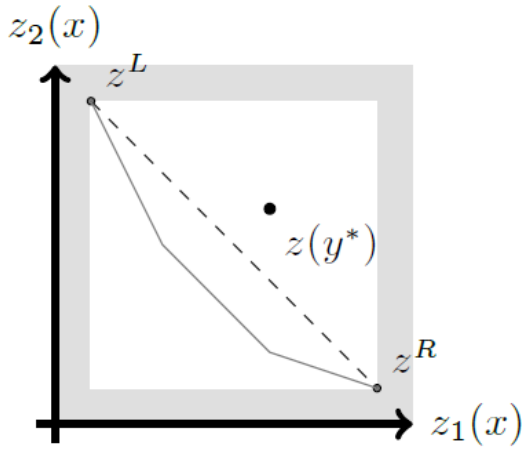


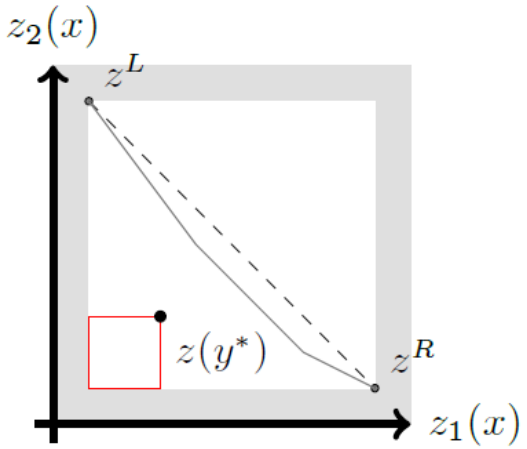
typical nondominated frontier of test instances used in the literature

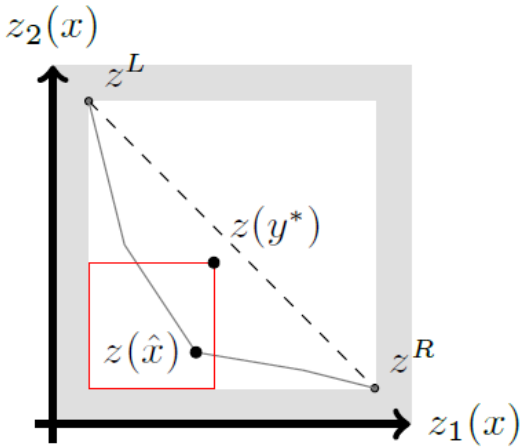


Enhancement: Same Integer Solution

- Treat a box with corner NDPs from solutions with same integer part differently
- Dichotomic search with fixed integer part
  - Requires linear program solves
- Verify using no-good constraint: $\sum_{j:\, x_j^*=0} x_j + \sum_{j:\, x_j^*=1} (1 - x_j) \geq 1$







Observations

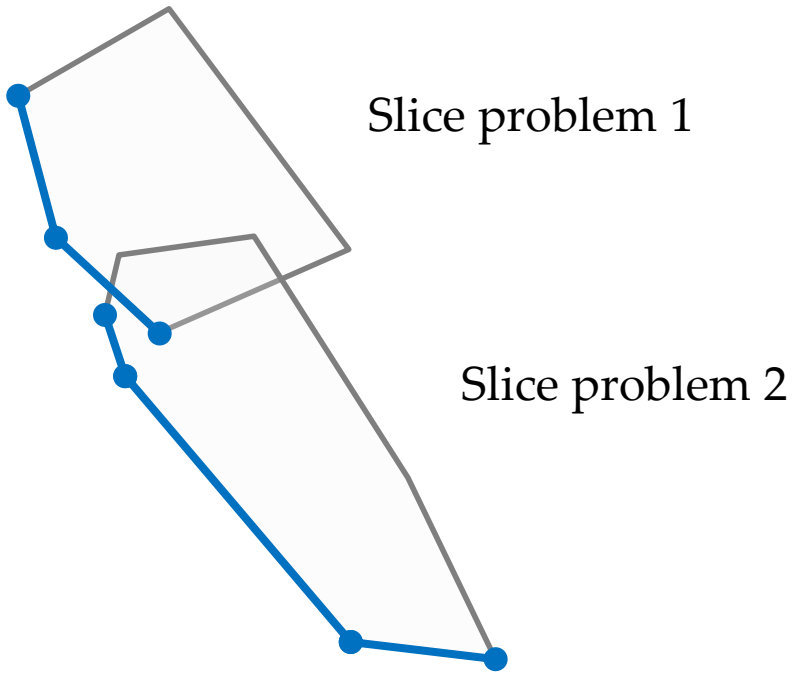
- Advantages of Balanced Box and Boxed-Line Methods
  - Approximation
    - Using a priority queue and always exploring the box with the largest unexplored area, the methods quickly obtain a high-quality approximation of the nondominated frontier
  - Parallelization
    - The methods parallelize naturally, as each of the boxes in the priority queue can be explored independently

Epsilon Tabu Constraint Method

Soylu and Yildiz

Epsilon Tabu Constraint Method

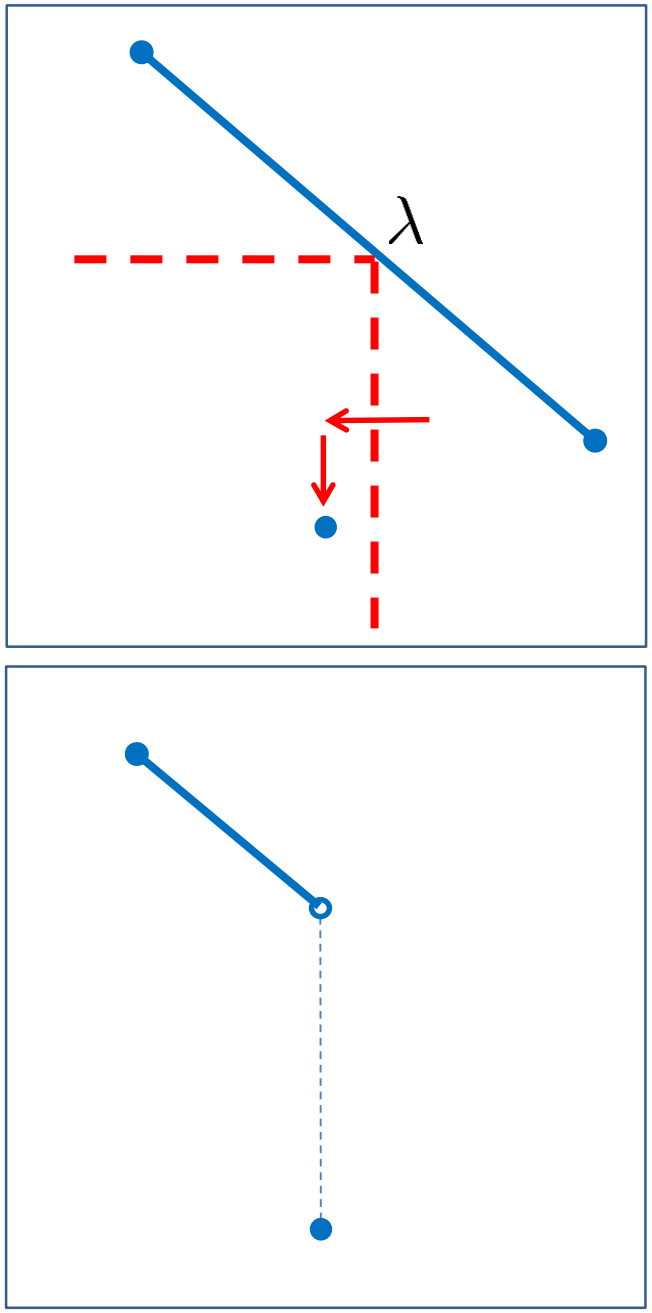
Identify consecutive slice problems



ε Tabu Constraint Method

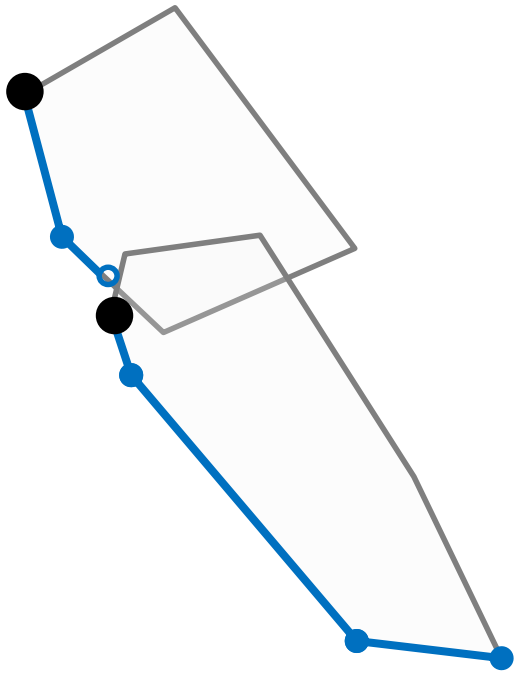
Verifying line segment

$$
\begin{aligned}
\hat{z} = \text{lexmin} \quad & (z_1(x), z_2(x)) \\
\text{s.t.} \quad & z_1(x) \leq \lambda z_1^i + (1-\lambda) z_1^{i+1} \\
& z_2(x) \leq \lambda z_2^i + (1-\lambda) z_2^{i+1} \\
& x_I \neq x_I^0 \\
& x \in \mathbb{X} \\
& 0 \leq \lambda \leq 1
\end{aligned}
$$



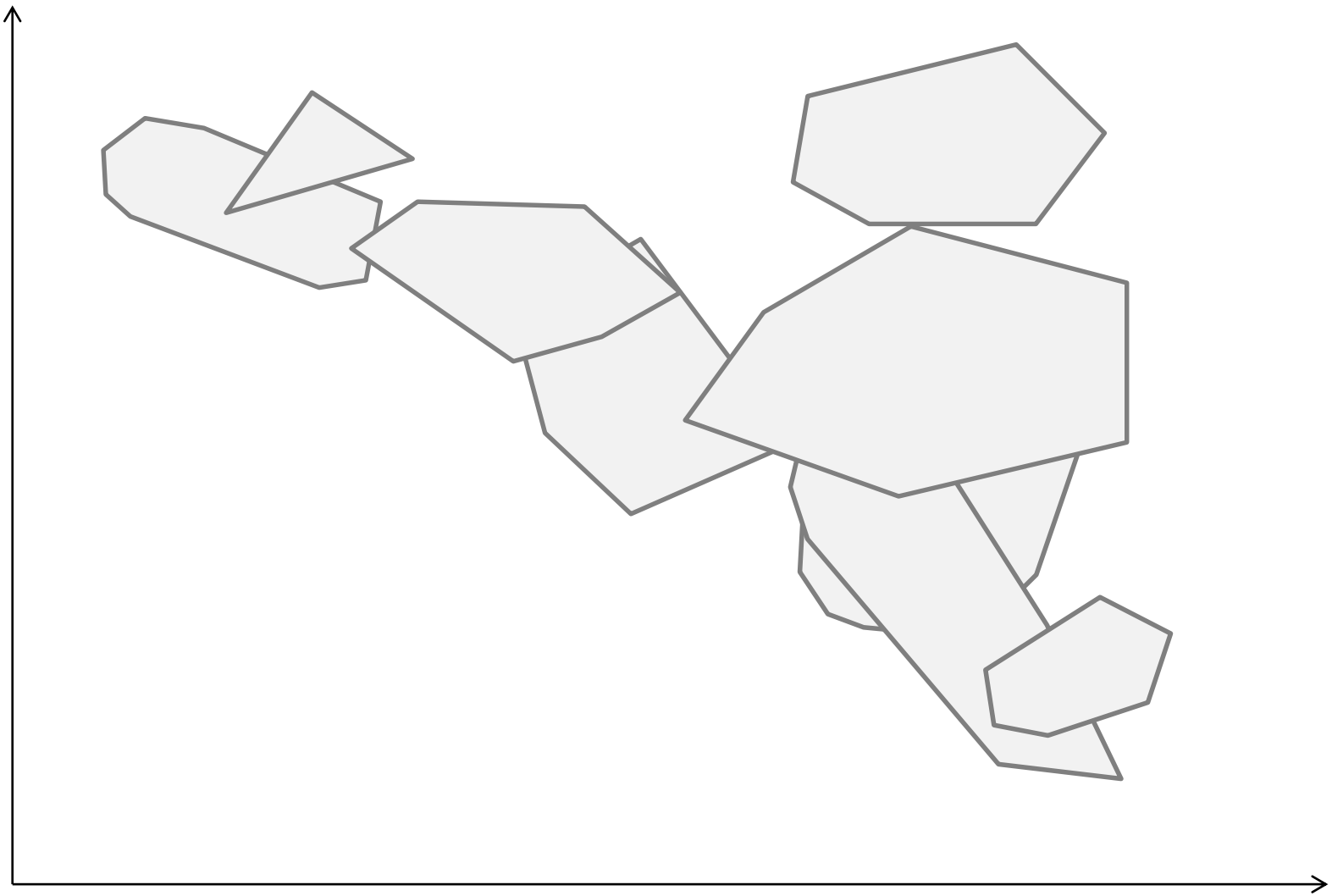
Epsilon Tabu Method

Identify consecutive slice problems

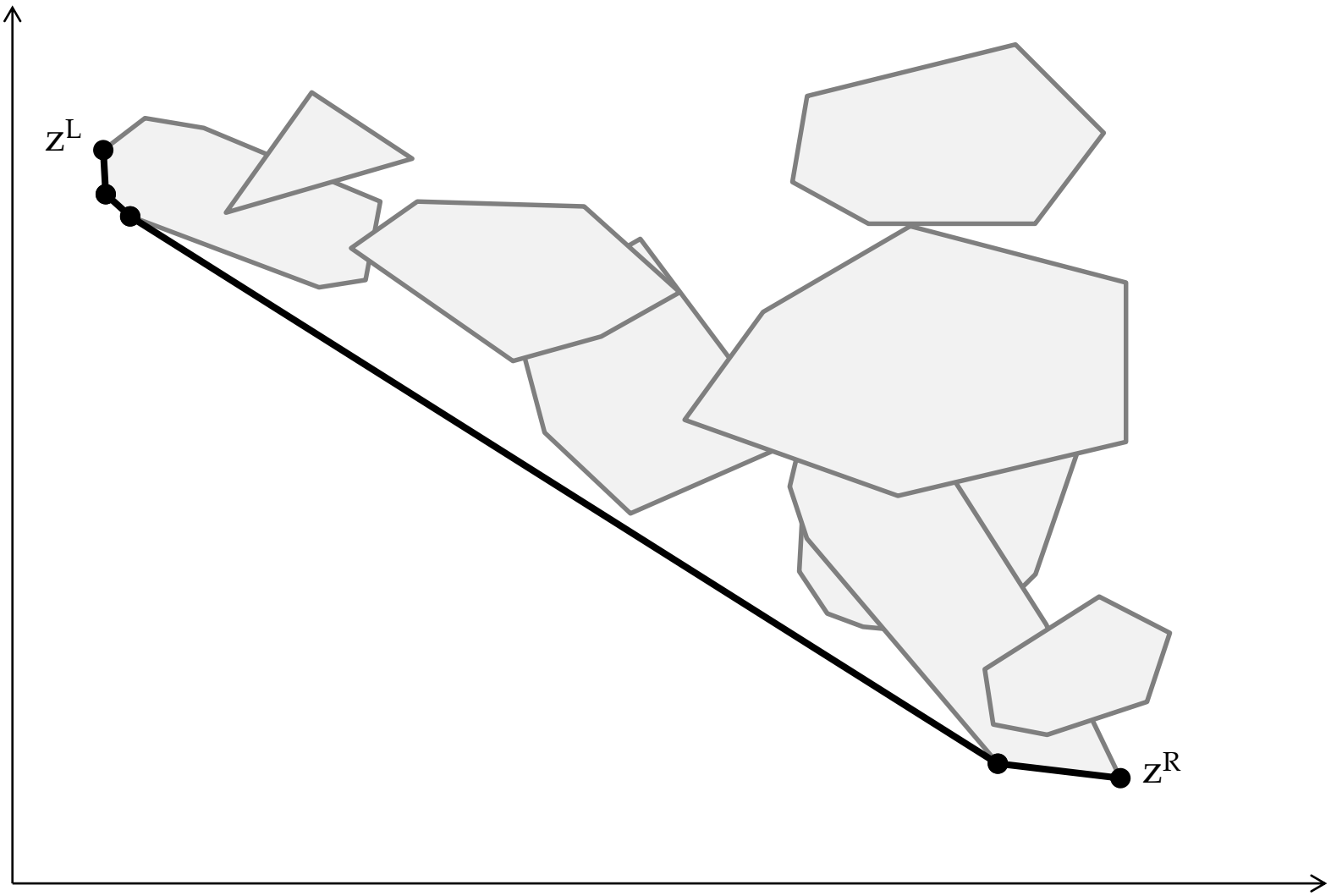


Search-and-Remove Method

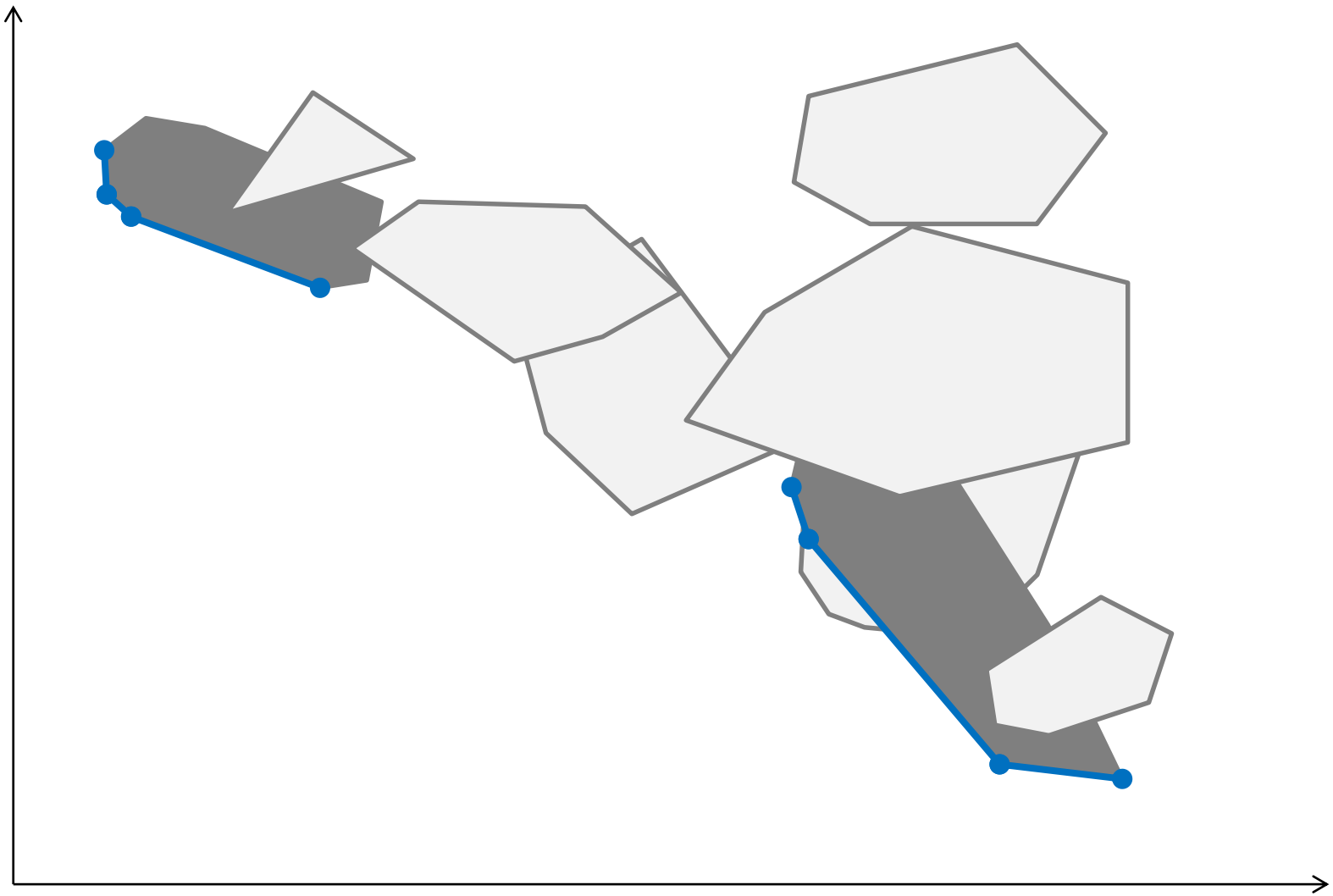
Soylu



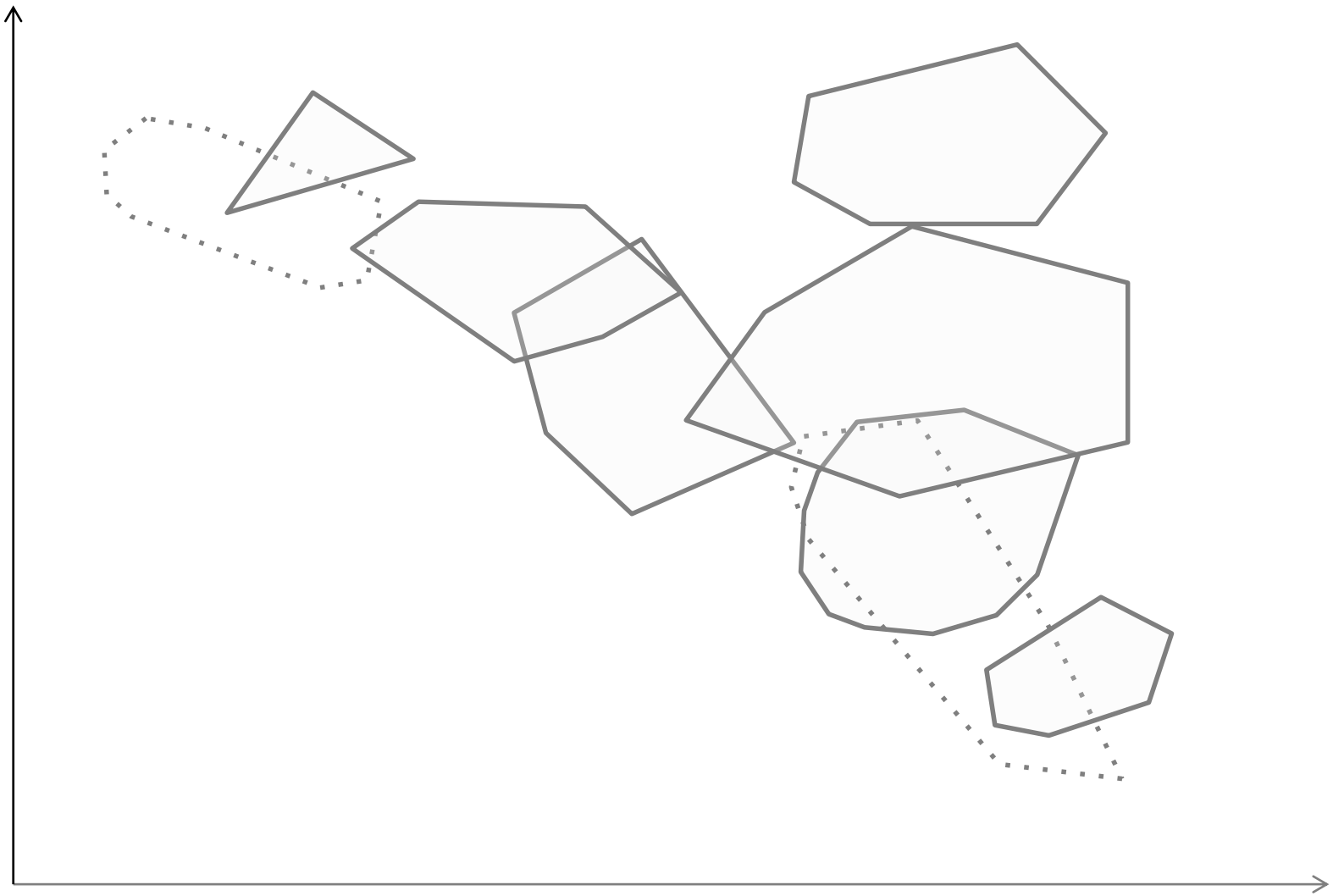
Iteration 1: Dichotomic Search



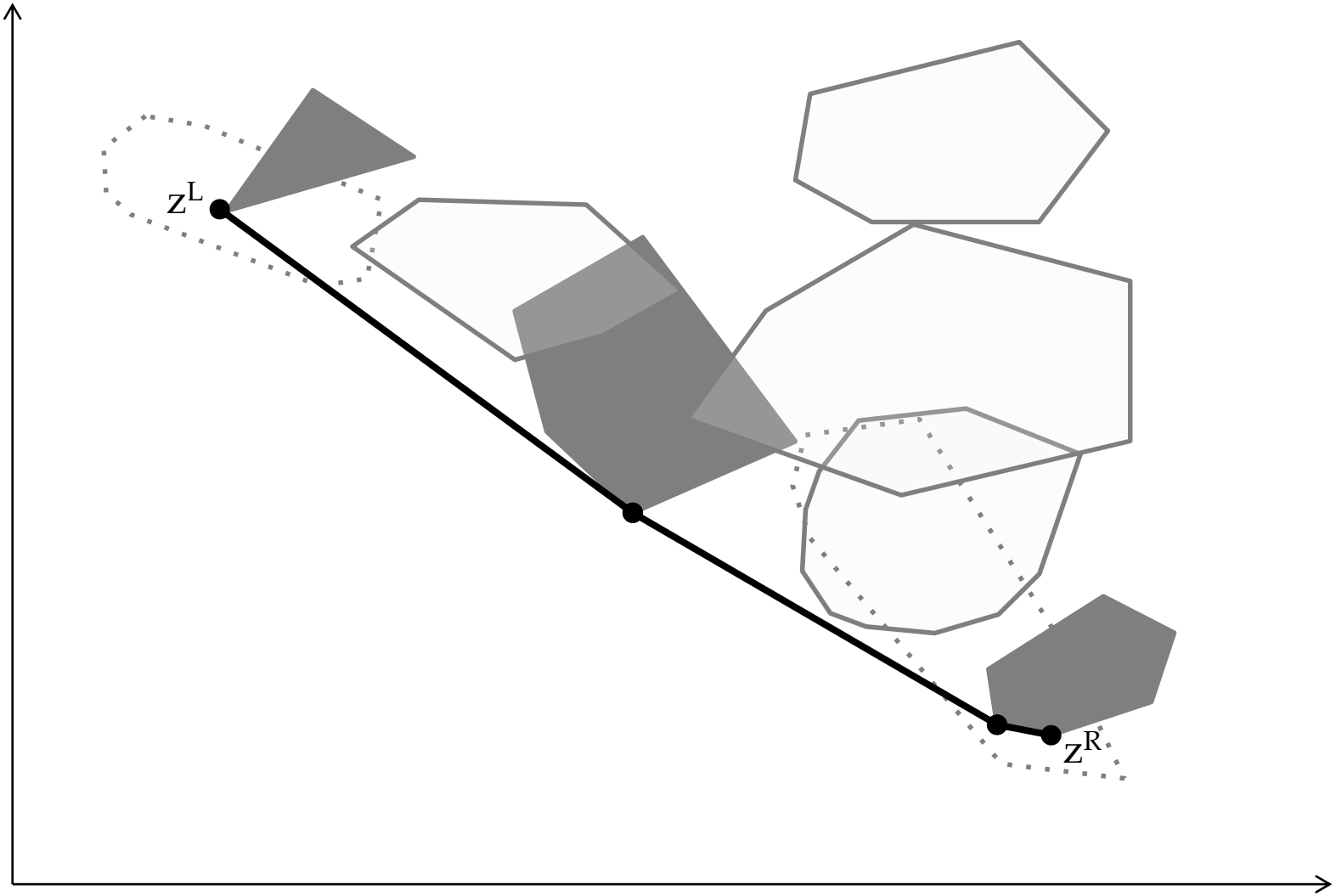
Iteration 1: Solve Slice Problems



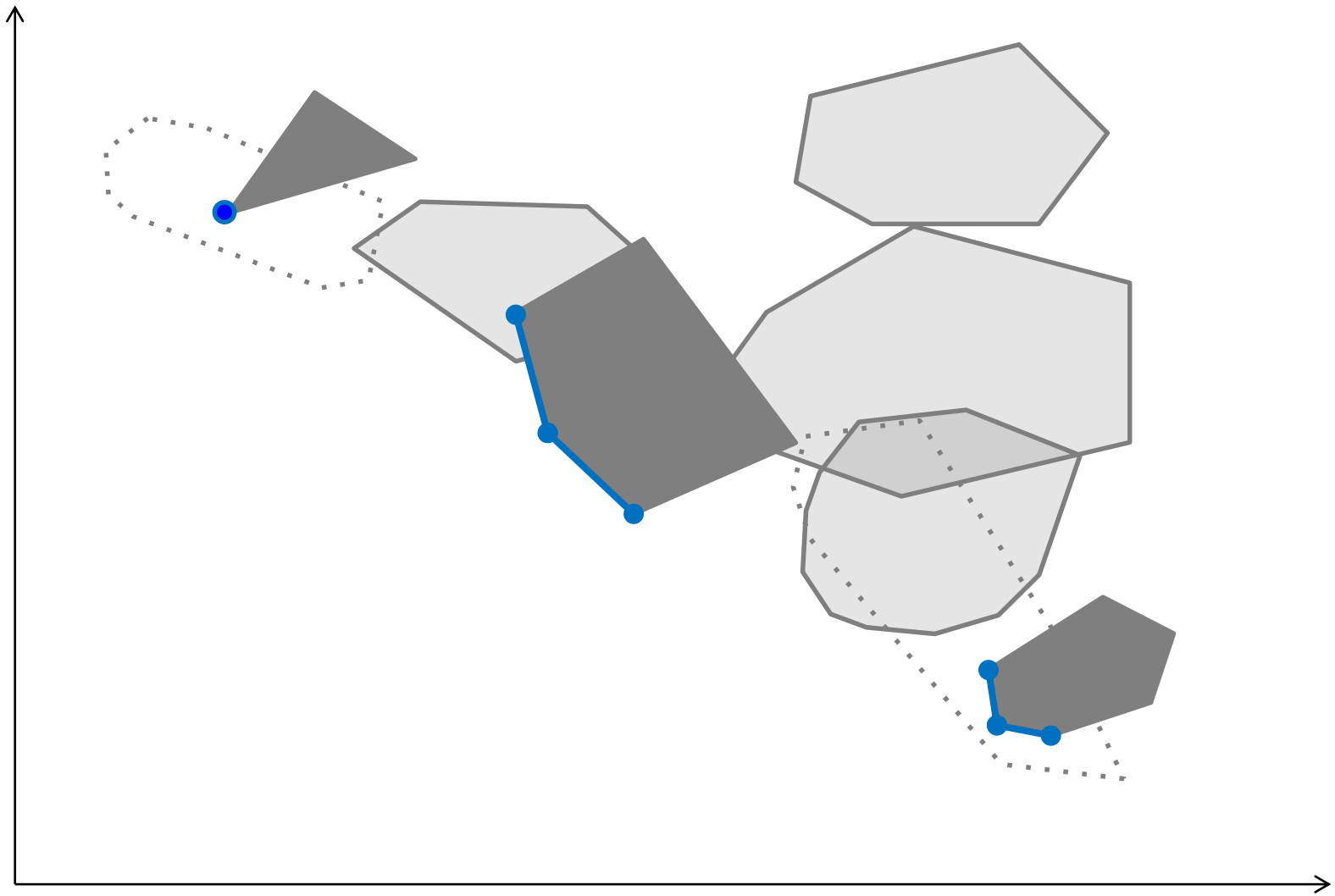
Iteration 1: Add Tabu Constraints



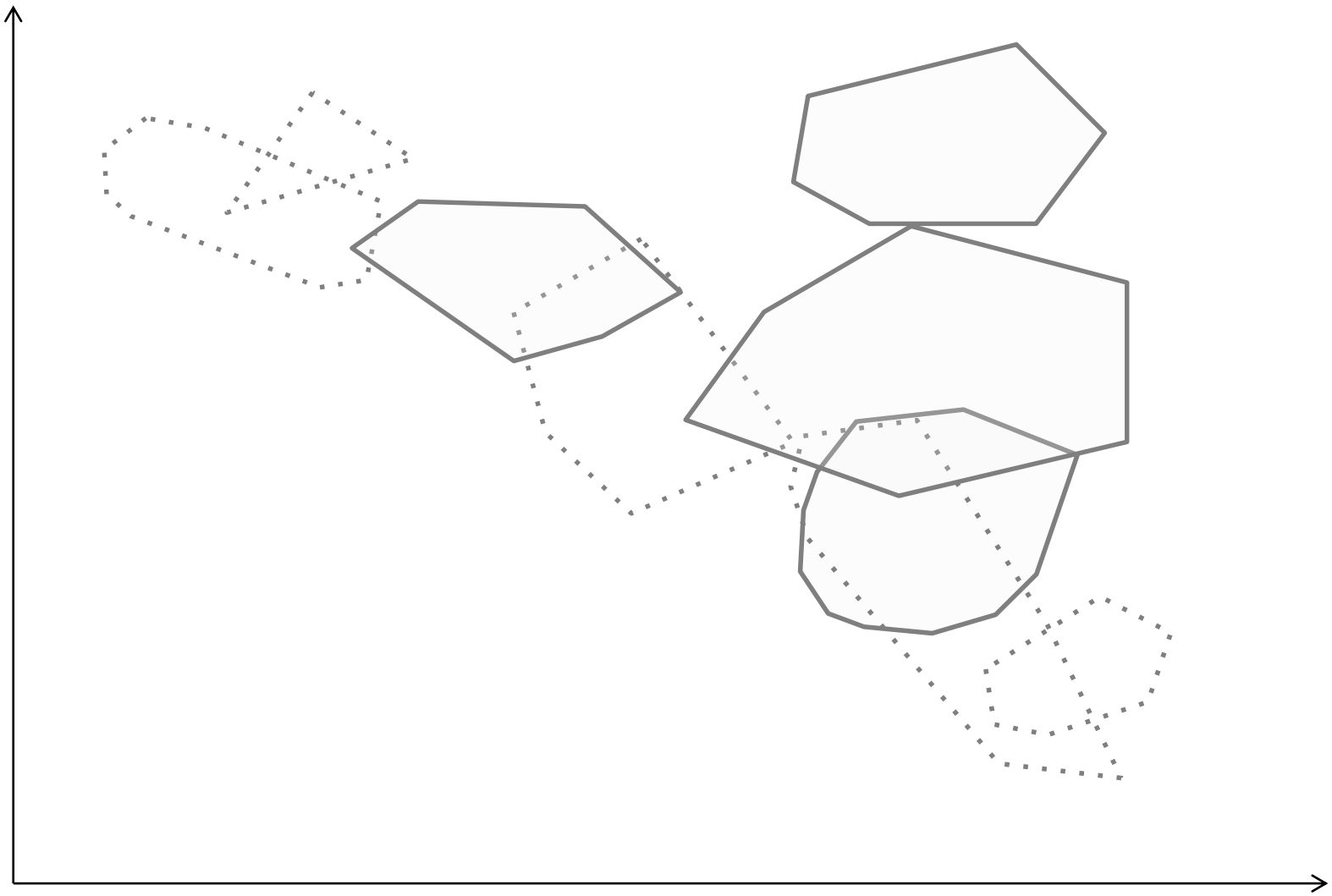
Iteration 2: Dichotomic Search



Iteration 2: Solve Slice Problems



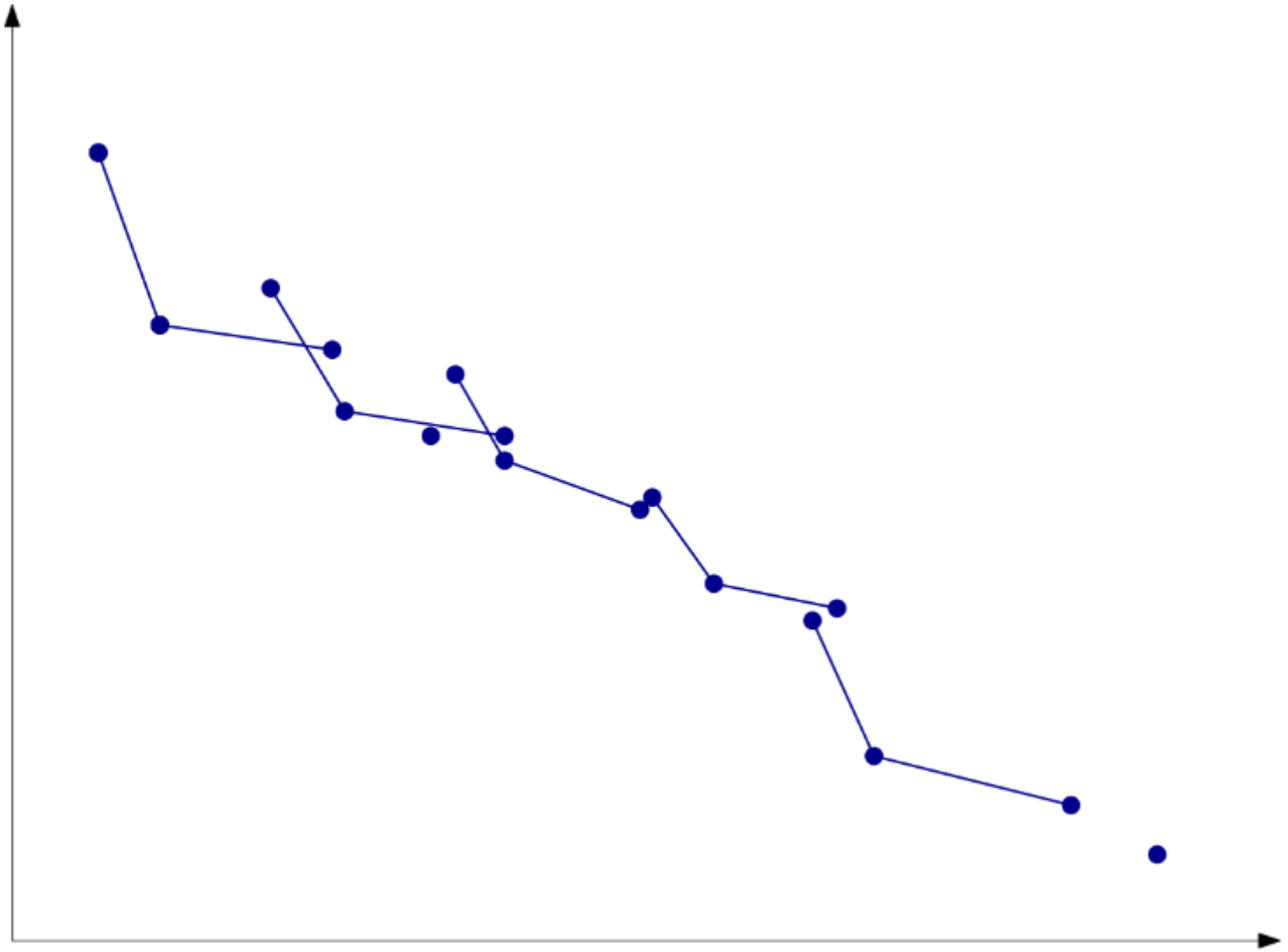
Iteration 2: Add Tabu Constraints



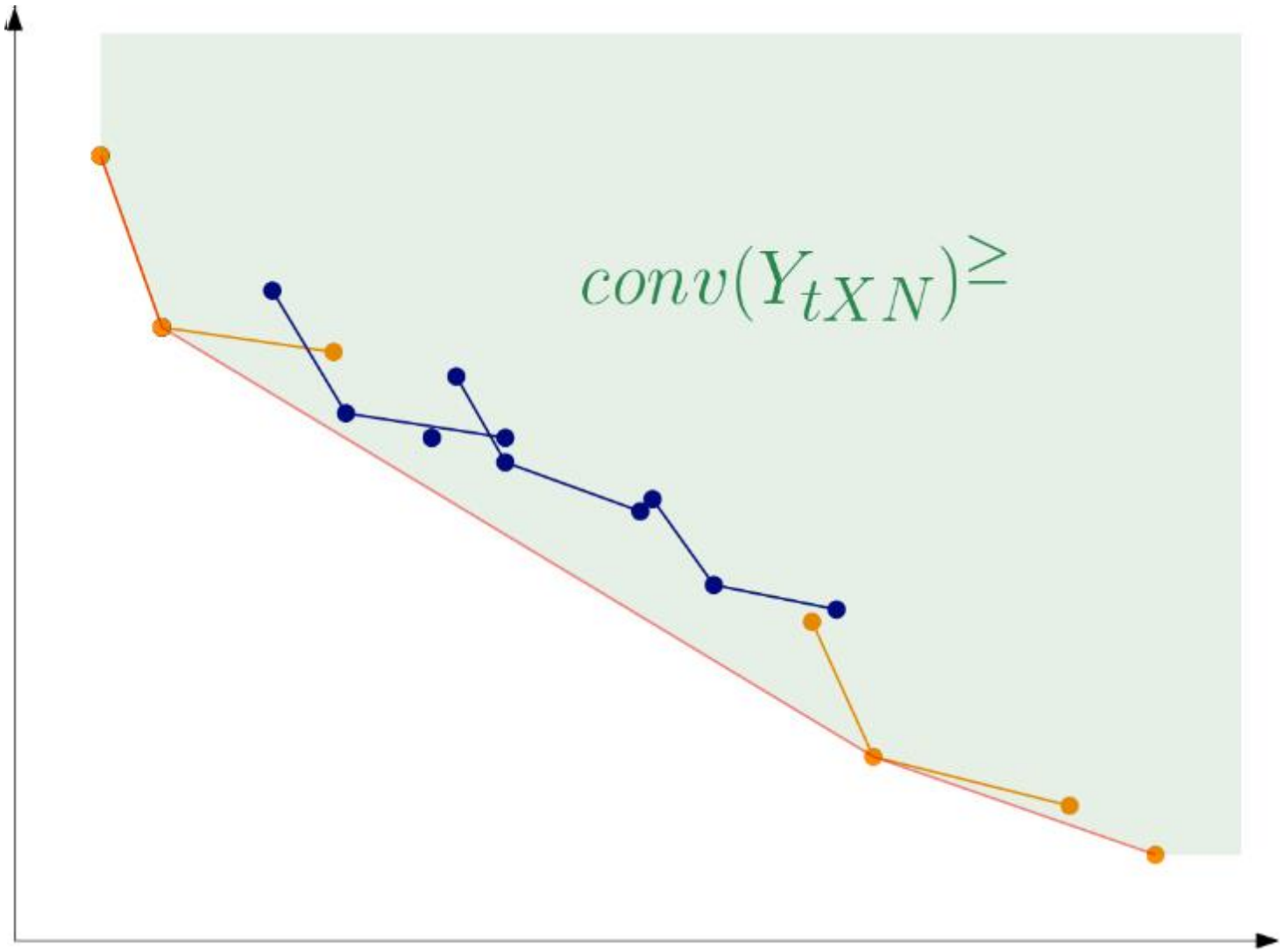
Stopping Conditions

- Infeasibility
- Bound sets

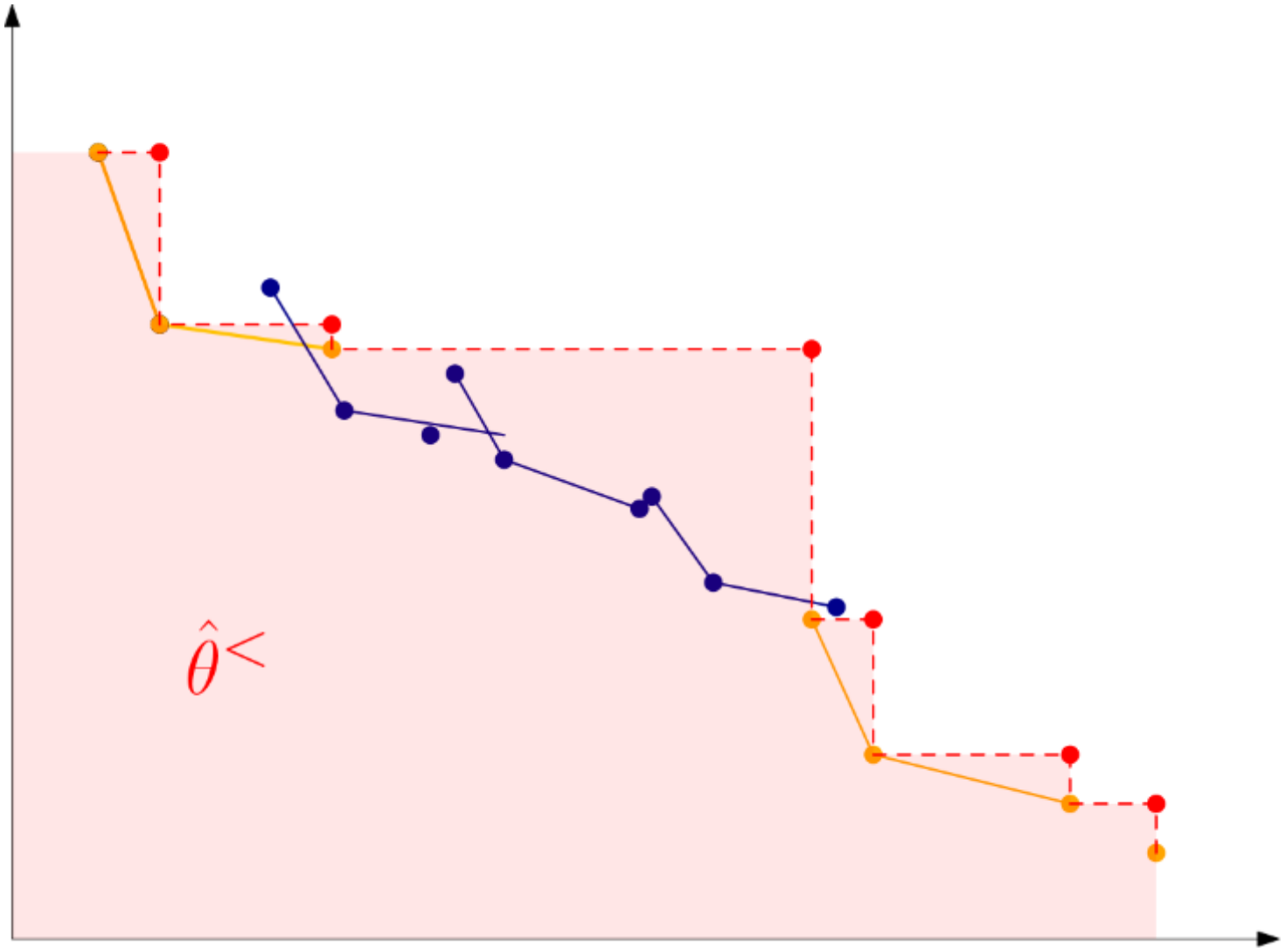
Bound sets



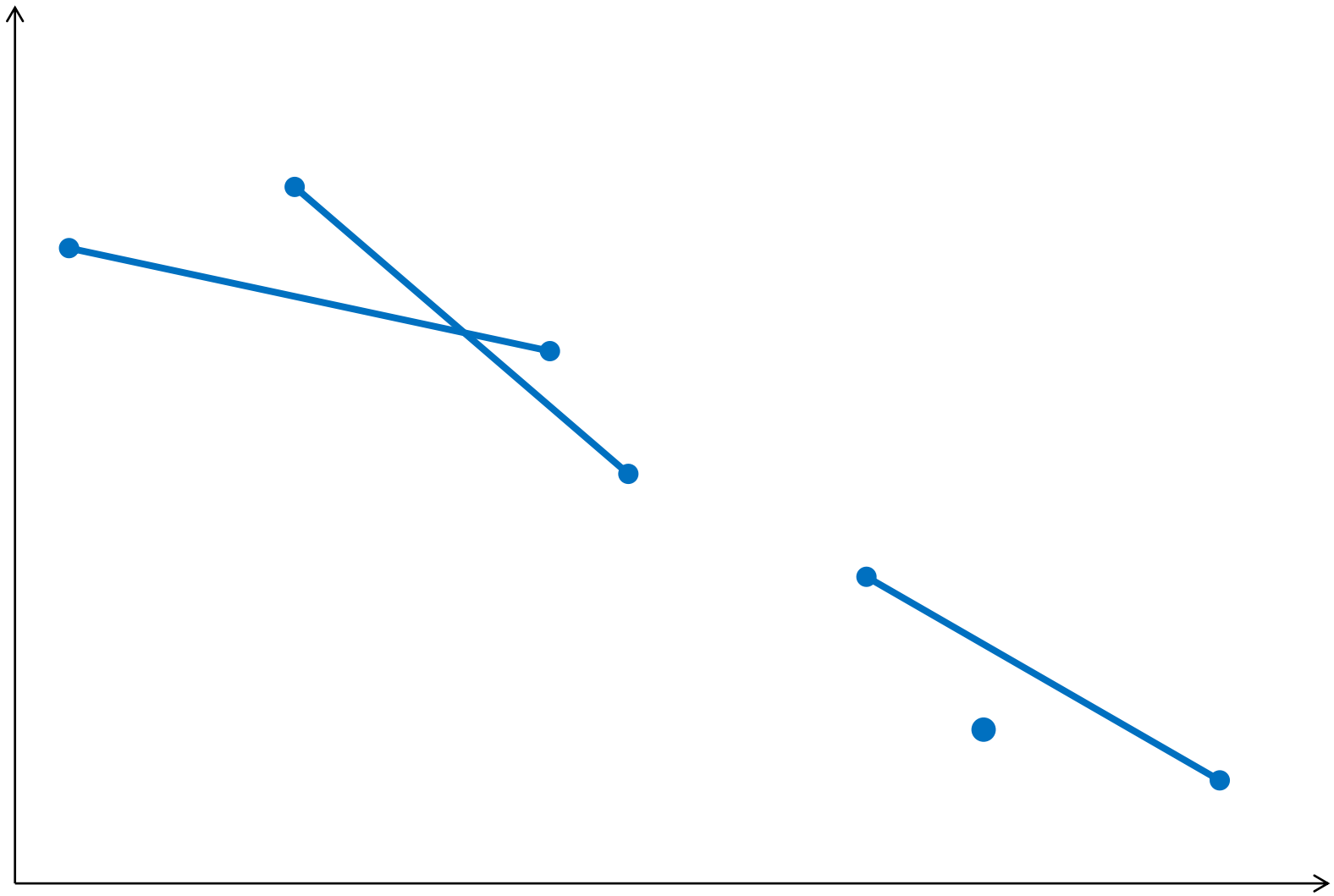
Lower bound set



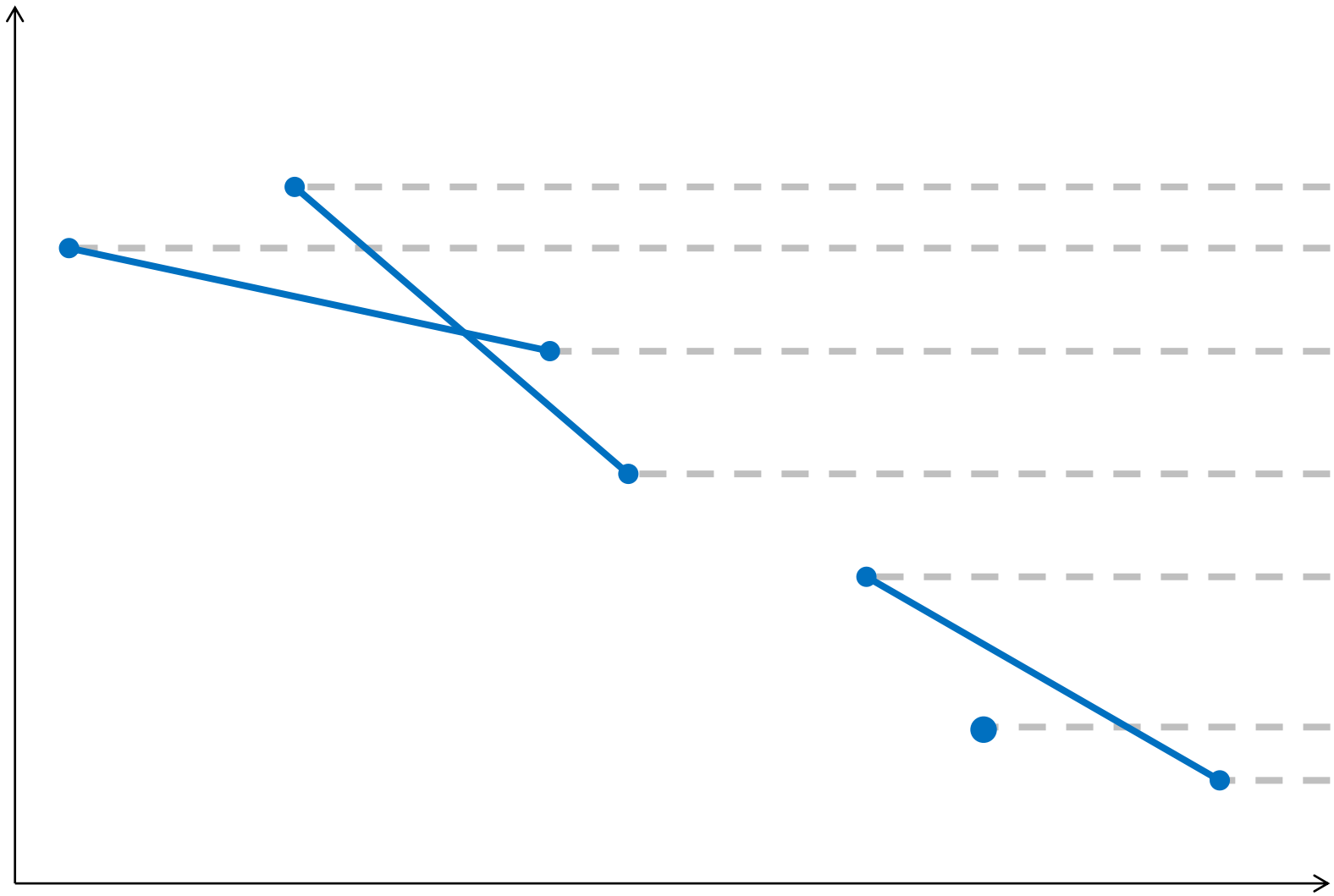
Upper bound set



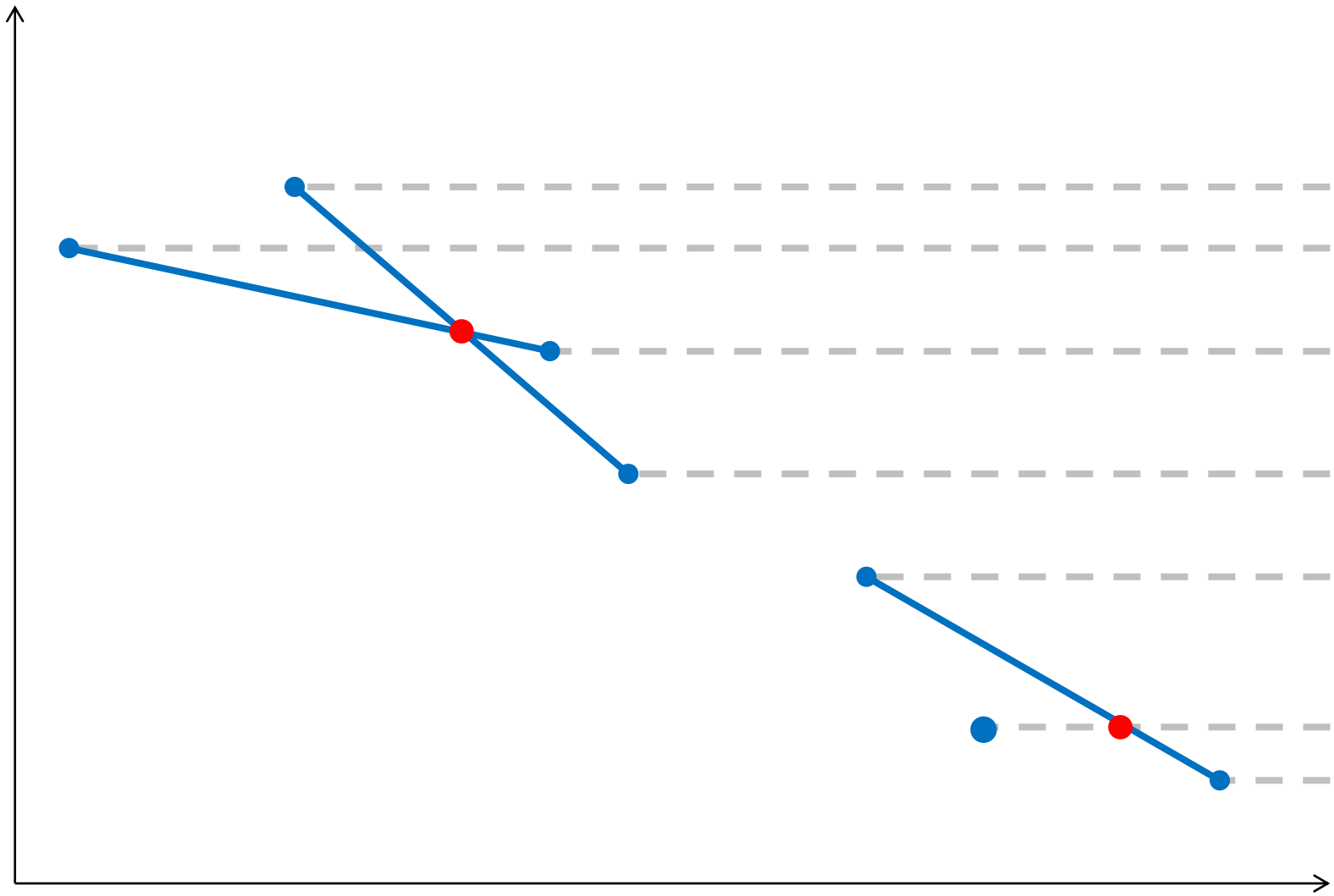
Construct NDF from list of slices



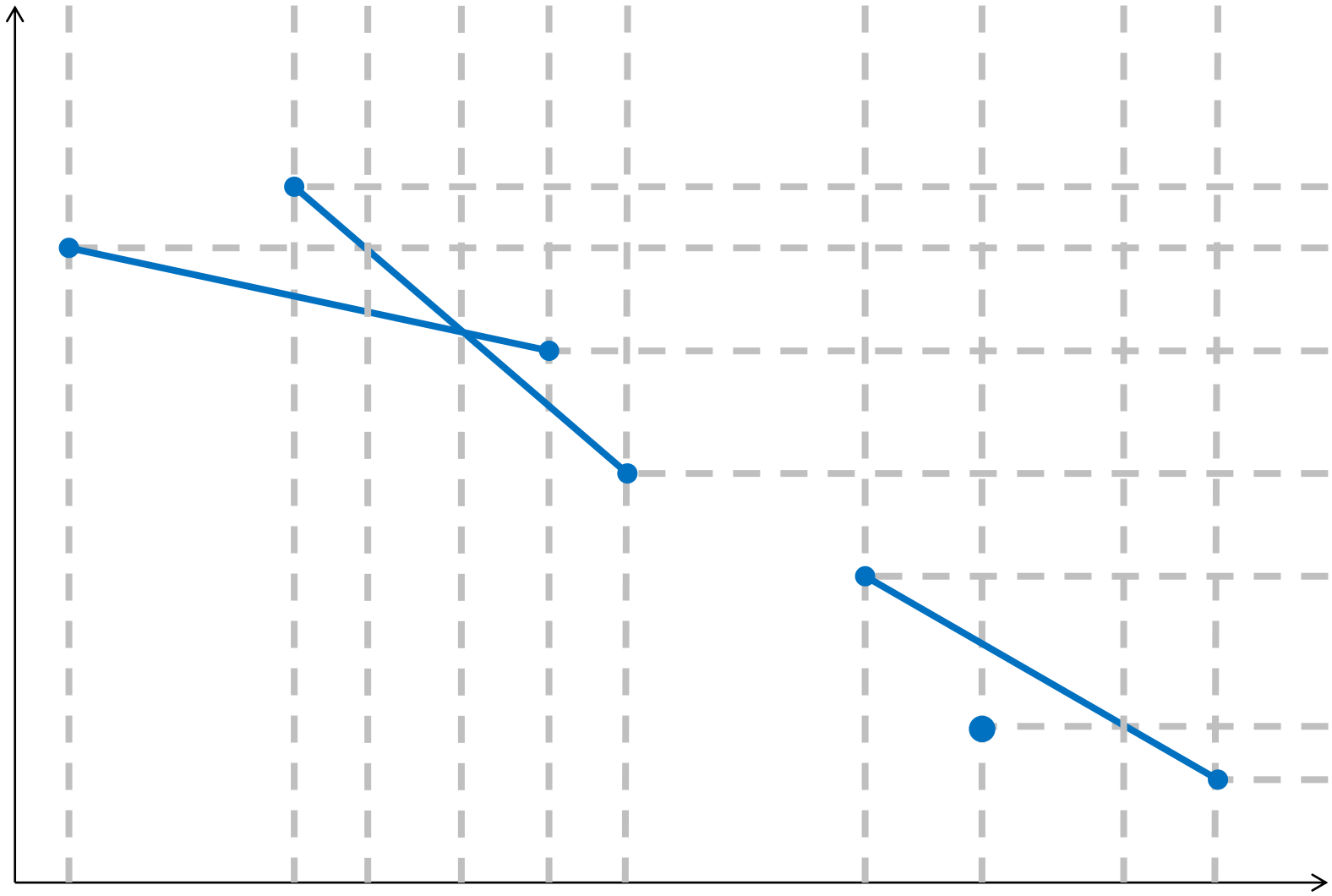
Add horizontal auxiliary lines



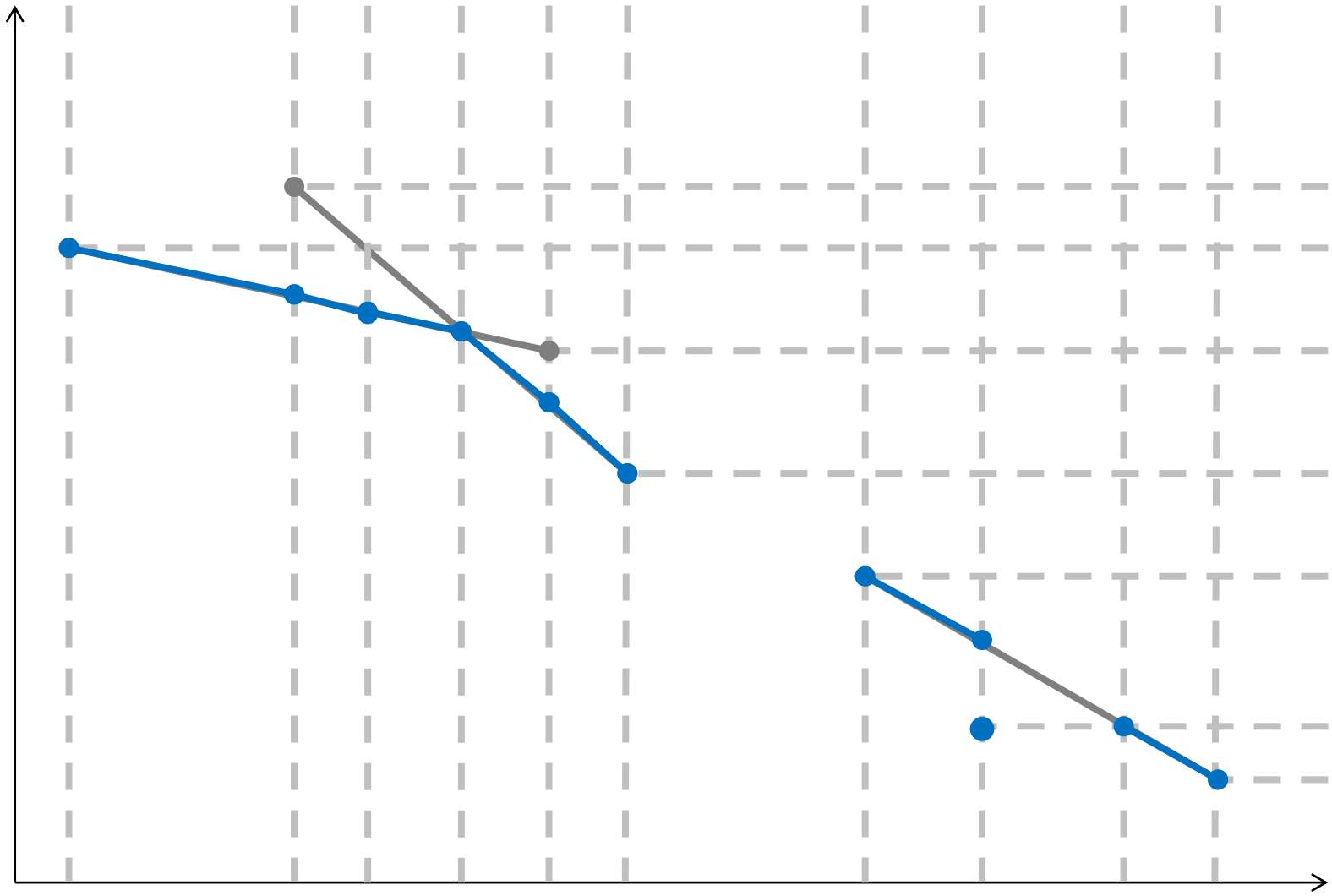
Compute all intersection points



Determine intervals along the z_1 axis



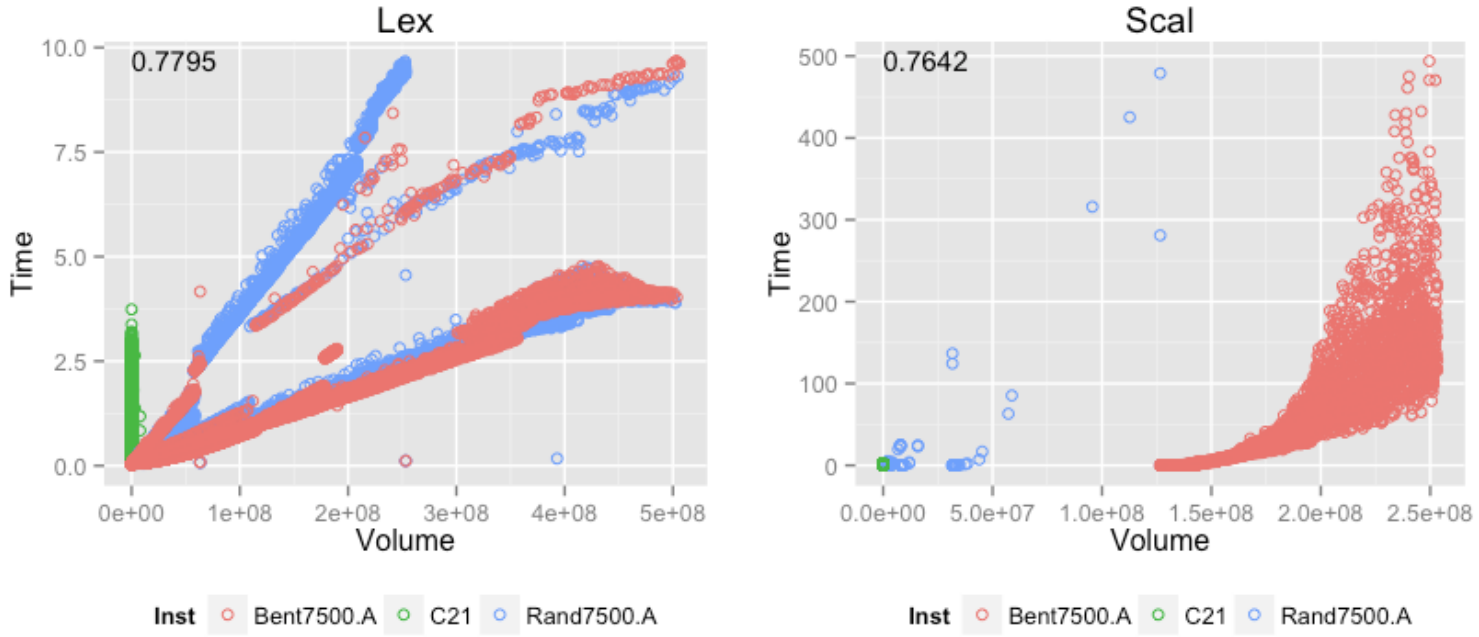
Find lowest line segment per interval



Boxed-Line Method: PureLex Variant

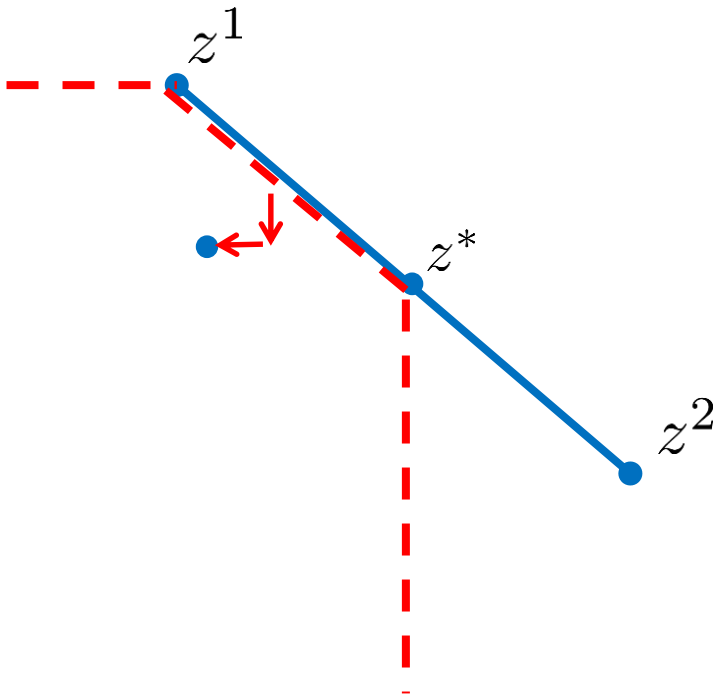
Observations

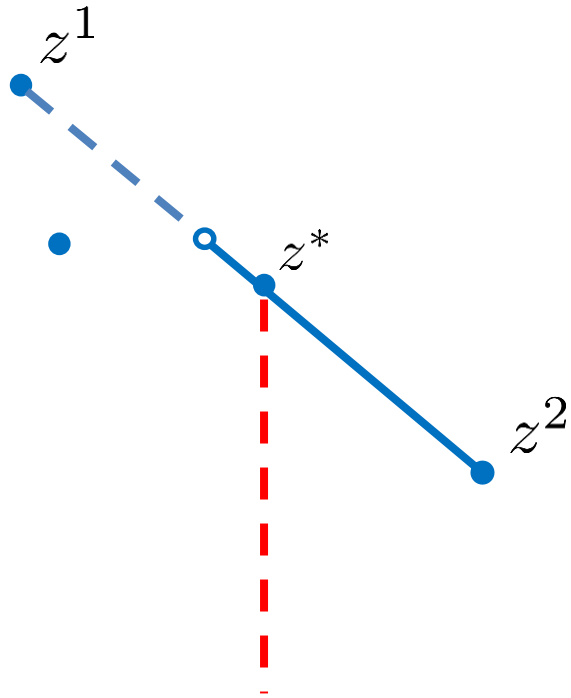
- Computational experiments reveal:
  - The larger the box (in criterion space), the harder it is to solve the single objective IPs
  - The time it takes to solve Scalarized IPs is more sensitive to the size of the box than the time it takes to solve Lexicographic IPs.



Boxed-Line Method: PureLex Variant

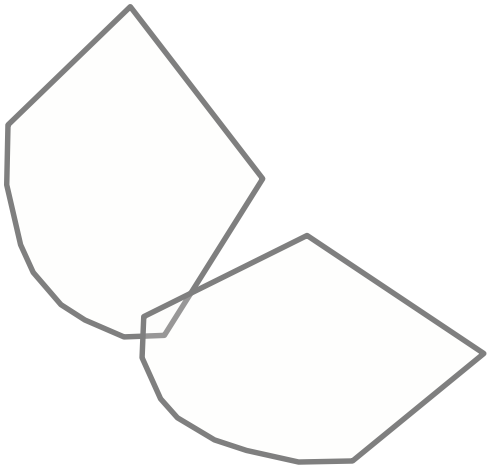
Correcting generated line using Lexmin



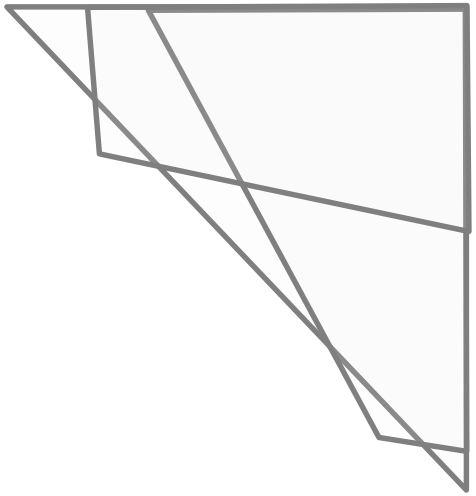


Computation Studies

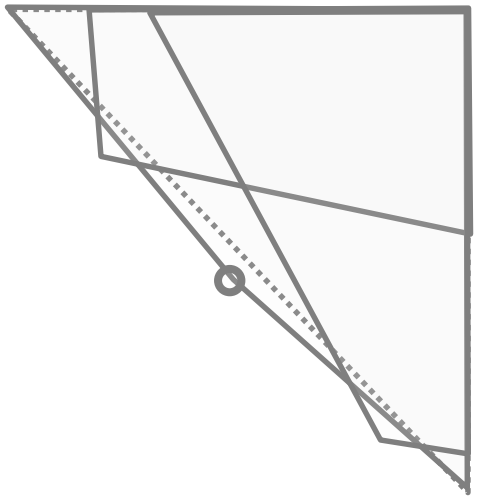
Instances



Historical Instances
- C160 & C320 (5 each)
- ~3500 & 17500 NDPs
- Few pareto slices
- Each slice contributes many, small segments



Aligned Instances
- Rand5000 (5)
- 3n NDPs & n+1 slices
- Most slices contribute at most 2 segments



Bent Instances
- Bent1000 (3)
- 3n NDPs & n+1 slices
- Scalarized IPs are very hard

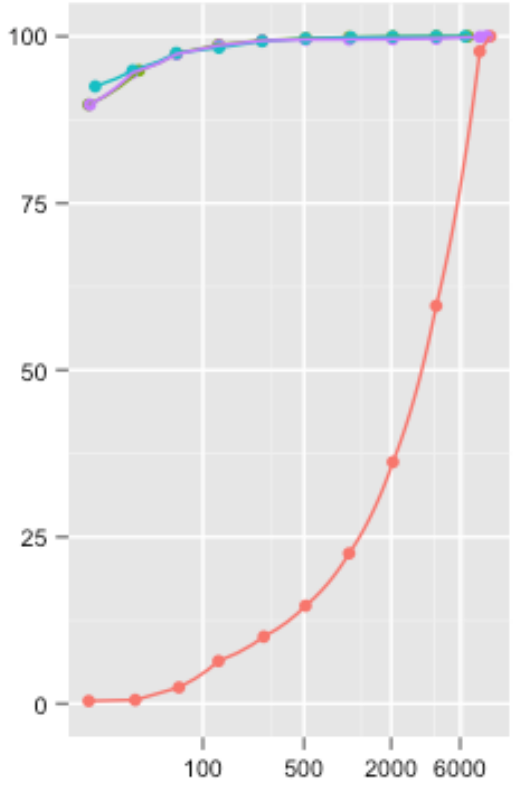
Algorithm	Ins	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	21	16850	294	37665.5	33307.6	4339.7	53514	17790	40	17894	-	171370	17803	15926	17766
	22	19778	410	42045.3	36927.4	5095.7	61766	20540	23	20663	-	191611	20510	18074	20476
	23	17319	343	38995.1	34570.3	4409.4	53859	17895	26	18043	-	171514	17884	15776	17871
	24	19898	460	46967.1	41632.8	5312.7	62338	20706	34	20892	-	200619	20696	17867	20682
	25	13682	337	24450.1	21268.9	3171.0	42196	14024	27	14121	-	130934	13994	12130	13964
Avg.		17505.4	368.8	38024.6	33541.4	4465.7	54734.6	18191	30	18322.6	-	173209.6	18177.4	15954.6	18151.8
Multibox	21	16850	294	38456.5	34019.1	4419.0	53903	17920	40	18023	-	172549	17933	16204	17896
	22	19778	410	42792.9	37590.8	5180.2	62343	20732	23	20856	-	193363	20702	18447	20668
	23	17319	343	38420.5	34061.9	4345.0	54357	18062	26	18207	-	173084	18051	16080	18038
	24	19902	460	49236.4	43622.3	5592.0	62974	20920	34	21100	-	202579	20911	18243	20896
	25	13680	337	24774.0	21540.7	3223.1	42626	14169	26	14262	-	132214	14139	12416	14109
Avg.		17505.8	368.8	38736.1	34167	4551.9	55240.6	18360.6	29.8	18489.6	-	174757.8	18347.2	16278	18321.4
NoGood	21	15699	294	6106.2	4890.5	1211.5	6598	1741	40	1849	1227	43443	2956	1227	1722
	22	18840	410	7788.7	6297.5	1485.3	8613	2287	23	2421	1595	52662	3850	1595	2233
	23	16449	343	6977.4	5607.1	1366.8	7459	1982	26	2131	1338	46627	3309	1338	1961
	24	18546	460	9535.6	7899.2	1630.3	10070	2672	34	2884	1808	56219	4468	1808	2679
	25	13239	337	5329.1	4296.6	1029.2	6578	1753	26	1853	1193	37911	2916	1193	1700
Avg.		16554.6	368.8	7147.4	5798.2	1344.6	7863.6	2087	29.8	2227.6	1432.2	47372.4	3499.8	1432.2	2059
Recursive	21	16831	294	37201.4	32767.3	4417.0	52297	16979	-	18339	-	170040	16971	15611	16955
	22	19763	410	41650	36417.7	5210.1	60312	19600	-	21112	-	189879	19568	17830	19536
	23	17315	343	38684.6	34149.4	4521.7	52585	17042	-	18501	-	170082	17030	15546	17018
	24	19890	460	46196.8	40725.8	5449.5	60571	19576	-	21419	-	198351	19566	17509	19552
	25	13667	337	24635	21314.5	3310.4	40899	13143	-	14613	-	129295	13113	11834	13083
Avg.		17493.2	368.8	37673.6	33074.9	4581.7	53332.8	17268	-	18796.8	-	171529.4	17249.6	15666	17228.8

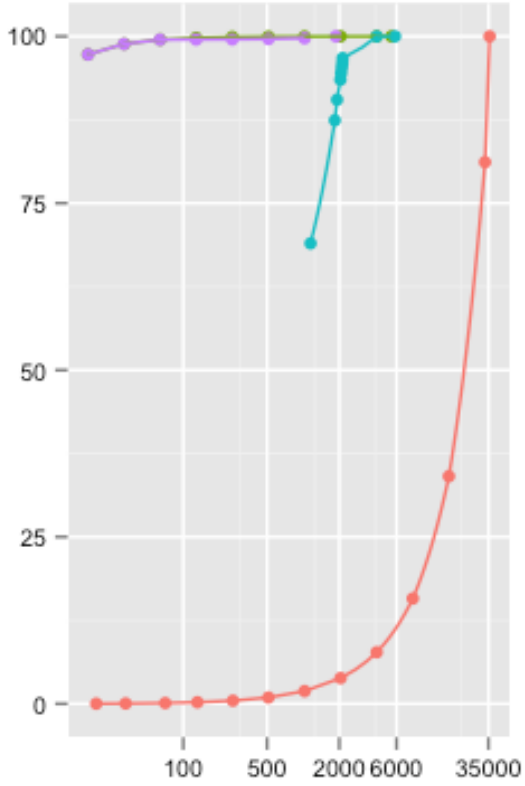
Comparison between the different algorithms for historical instances, class C320.

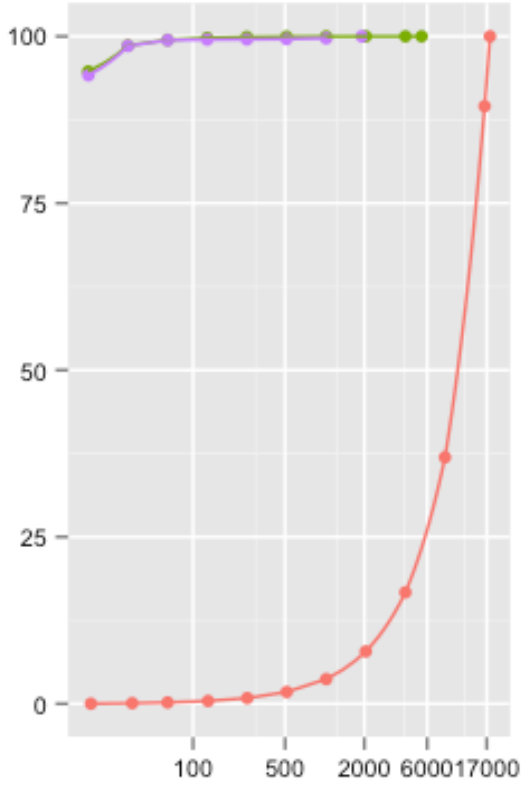
Algorithm	Ins	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	A	15002	5001	3996.1	1803.6	1875.2	52010	14653	398	22306	-	104487	14613	3259	14567
	B	15002	5001	3853.1	1687.7	1849.7	51738	14561	493	22123	-	103984	14519	3154	14475
	C	15002	5001	3981.6	1850.5	1819.9	51695	14544	518	22089	-	103755	14495	3193	14446
	D	15002	5001	3842.0	1699.0	1824.9	51744	14601	449	22093	-	104201	14563	3153	14521
	E	15002	5001	3948.9	1767.4	1864.2	51745	14574	474	22123	-	104228	14542	3116	14504
Avg.		15002	5001	3924.3	1761.6	1846.8	51786.4	14586.6	466.4	22146.8	-	104131	14546.4	3175	14502.6
Multibox	A	15002	5001	4022.5	1700.4	1987.3	53949	16329	339	20952	-	113616	16303	84	16221
	B	15002	5001	3925.9	1638.6	1952.6	53197	16036	430	20695	-	112354	16020	78	15943
	C	15002	5001	4040.1	1708.4	1996.3	53351	16124	428	20675	-	112264	16101	77	15989
	D	15002	5001	3909.1	1565.3	2003.1	53695	16248	368	20831	-	113210	16241	66	16140
	E	15002	5001	4068.5	1736.9	1997.6	53522	16176	387	20783	-	112744	16163	76	16057
Avg.		15002	5001	3993.2	1669.9	1987.4	53542.8	16182.6	390.4	20787.2	-	112837.6	16165.6	76.2	16070
NoGood	A	15002	5001	4454.9	2278.3	1852.4	53523	16157	326	20820	63	113195	16218	63	16064
	B	15002	5001	4909.8	2623.9	1976.3	53881	16322	386	20749	102	113185	16397	102	16191
	C	15002	5001	4267.5	1969.3	1973.5	53239	16081	422	20592	63	112113	16126	63	15954
	D	15002	5001	4334.7	2025.1	1985.4	53682	16209	373	20798	93	113018	16277	93	16092
	E	15002	5001	4642.9	2328.5	1982.3	53552	16187	391	20704	83	112541	16239	83	16065
Avg.		15002	5001	4522.0	2245.0	1954.0	53575.4	16191.2	379.6	20732.6	80.8	112810.4	16251.4	80.8	16073.2
Recursive	A	15002	5001	2861.8	1383.9	1184.6	29711	3638	-	22435	-	72937	3616	1	3594
	B	15002	5001	2788.4	1301.3	1192.8	29587	3614	-	22359	-	72826	3594	1	3574
	C	15004	5001	2827.3	1350.4	1184.6	29538	3597	-	22344	-	72824	3578	1	3559
	D	15002	5001	2788.5	1306.2	1187.8	29664	3630	-	22404	-	72986	3608	1	3586
	E	15002	5001	2851.8	1367.3	1189.4	29641	3689	-	22263	-	72909	3664	1	3639
Avg.		15002.4	5001	2823.6	1341.8	1187.8	29628.2	3633.6	-	22361	-	72896.4	3612	1	3590.4

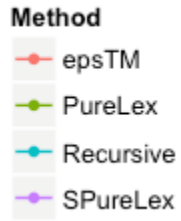
Comparison between the different algorithms for new instances, $n = 5000$.

Approximation: Explored Area

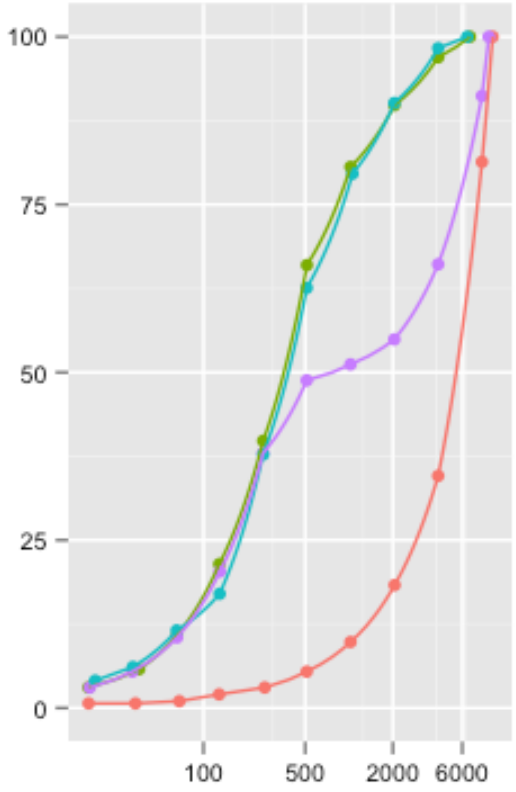


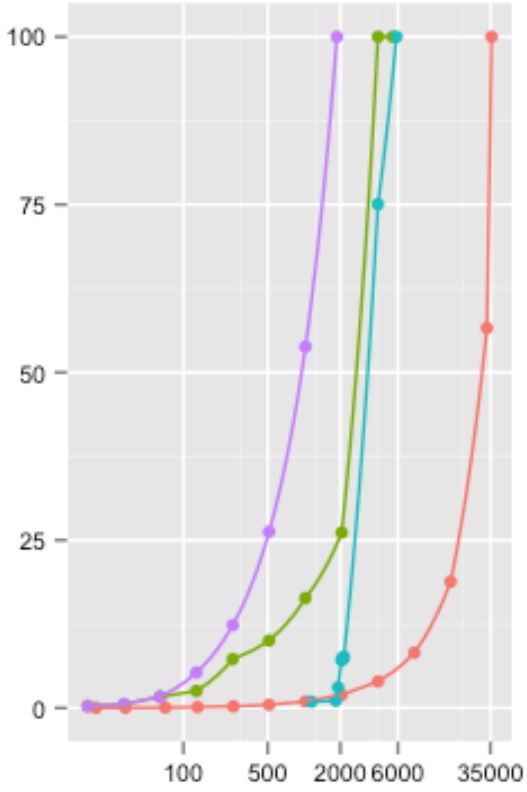


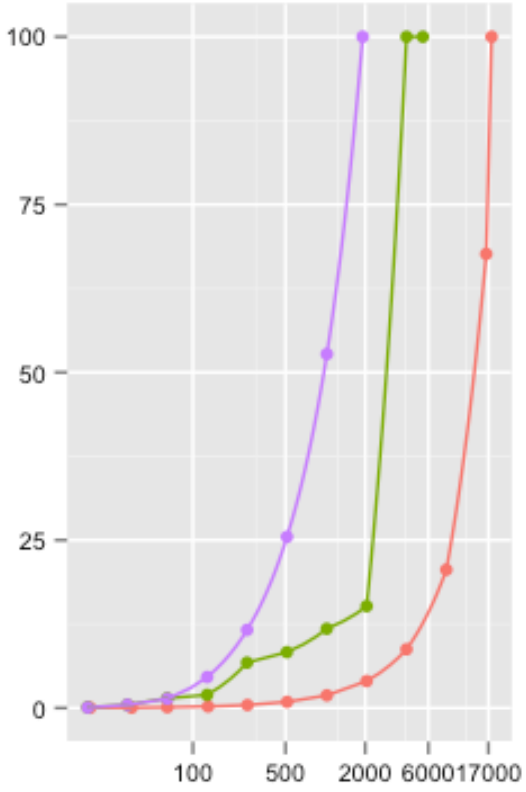


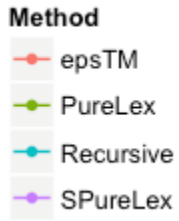


Approximation: Number of slices

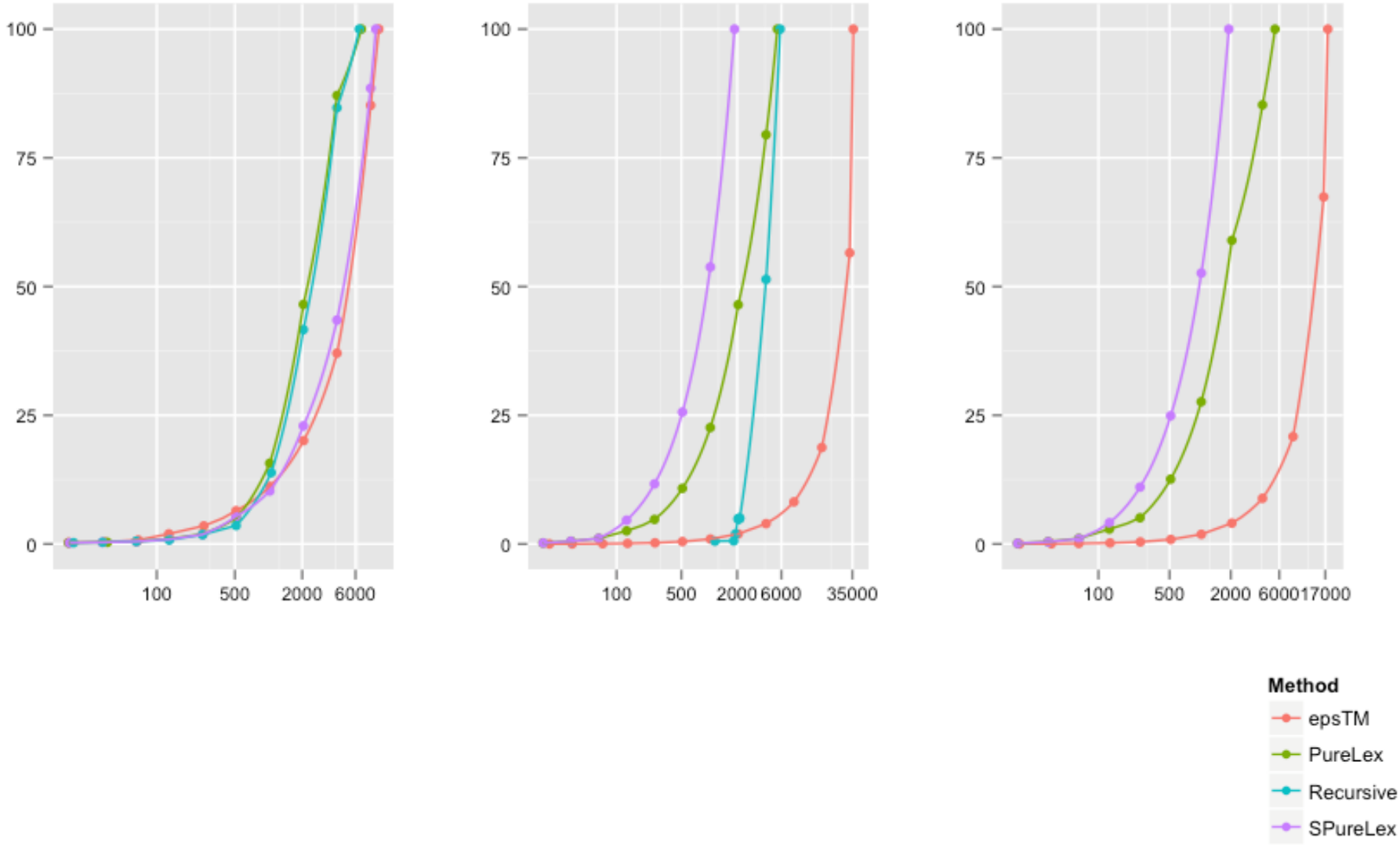








Approximation: Length of nondominated line segments



Future Research

Future Research

- Creating additional instances for testing biobjective mixed integer programming algorithms
- Designing and implementing an algorithm for triobjective mixed integer programming
- Exploring how to (best) reuse information from previous integer programming solves
- Developing methods that “reduce” the number of objective functions

THANK YOU
MERCI
BEDANKT