Computational Geometry

Lecture 4:
Linear Programming
or
Profit Maximization

Part I:
Introduction to Linear Programming

Philipp Kindermann

Winter Semester 2020

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2.

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2.

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2. For the production of x_1 units of P_1 and x_2 units of P_2, you're profit in € is:

$$G(x_1, x_2) = 30x_1 + 50x_2$$

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2. For the production of x_1 units of P_1 and x_2 units of P_2, you're profit in € is:

$$G(x_1, x_2) = 30x_1 + 50x_2$$

Three machines M_A, M_B and M_C produce the required components A, B and C for the products.

$$\begin{aligned} M_A &: \\ M_B &: \\ M_C &: \end{aligned}$$

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2. For the production of x_1 units of P_1 and x_2 units of P_2, you're profit in € is:

$$G(x_1, x_2) = 30x_1 + 50x_2$$

Three machines M_A, M_B and M_C produce the required components A, B and C for the products. The components are used in different quantities for the products

$$\begin{aligned} M_A: &\quad 4x_1 + 11x_2 \\ M_B: &\quad x_1 + x_2 \\ M_C: &\quad x_2 \end{aligned}$$

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2. For the production of x_1 units of P_1 and x_2 units of P_2, you're profit in € is:

$$G(x_1, x_2) = 30x_1 + 50x_2$$

Three machines M_A, M_B and M_C produce the required components A, B and C for the products. The components are used in different quantities for the products, and each machine requires some time for the production.

$$\begin{array}{lrcrcl} M_A: & 4x_1 & + & 11x_2 & \leq & 880 \\ M_B: & x_1 & + & x_2 & \leq & 150 \\ M_C: & & & x_2 & \leq & 60 \end{array}$$

Maximizing Profits

You're the boss of a small company that produces two products P_1 and P_2. For the production of x_1 units of P_1 and x_2 units of P_2, you're profit in € is:

$$G(x_1, x_2) = 30x_1 + 50x_2$$

Three machines M_A, M_B and M_C produce the required components A, B and C for the products. The components are used in different quantities for the products, and each machine requires some time for the production.

$$\begin{aligned} M_A: &\quad 4x_1 + 11x_2 \leq 880 \\ M_B: &\quad x_1 + x_2 \leq 150 \\ M_C: &\quad x_2 \leq 60 \end{aligned}$$

Which choice of (x_1, x_2) maximizes the profit?

Solution

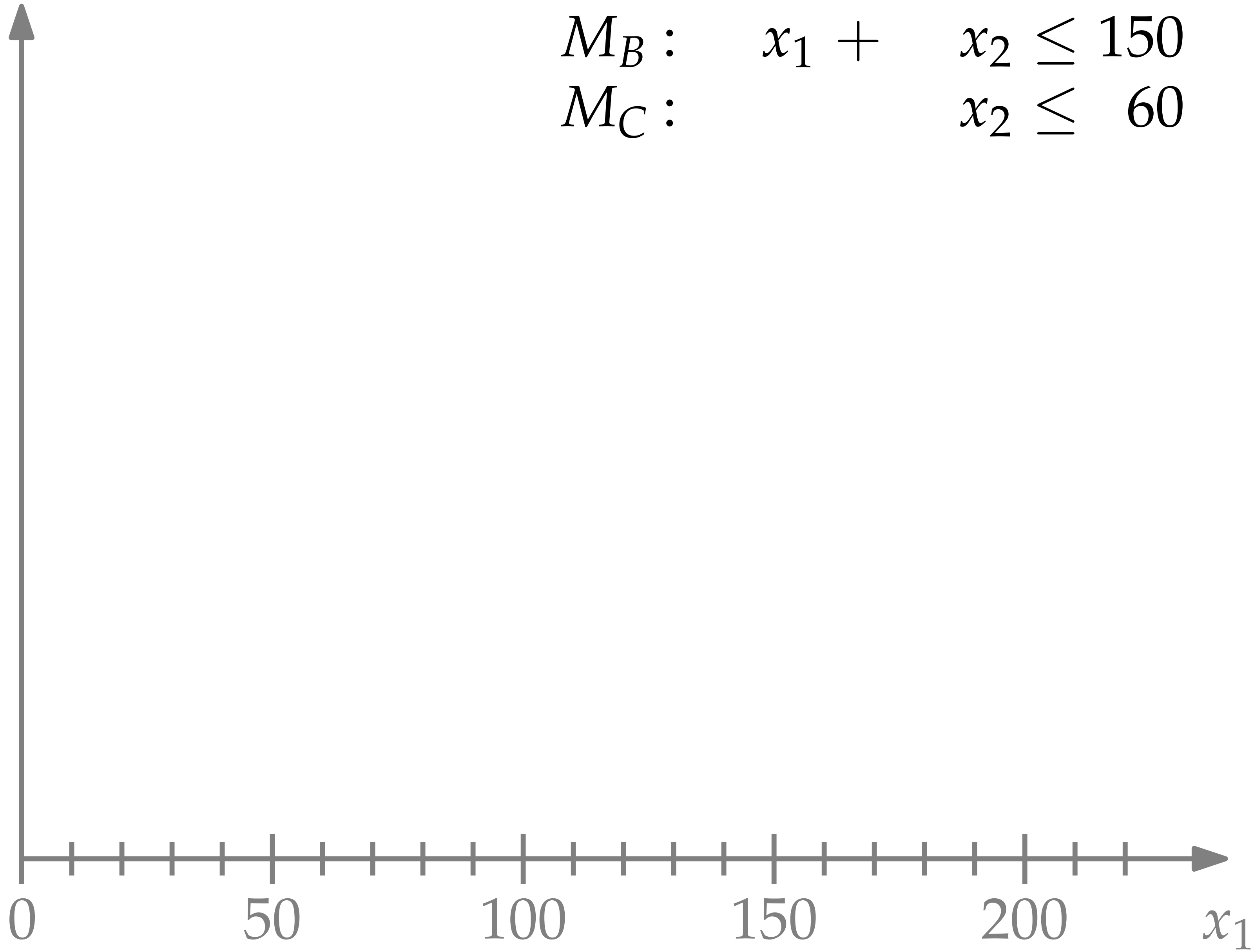
Linear constraints:

$$
\begin{aligned}
M_A: \quad 4x_1 + 11x_2 &\le 880 \\
M_B: \quad x_1 + x_2 &\le 150 \\
M_C: \quad x_2 &\le 60
\end{aligned}
$$

Solution

Linear constraints:

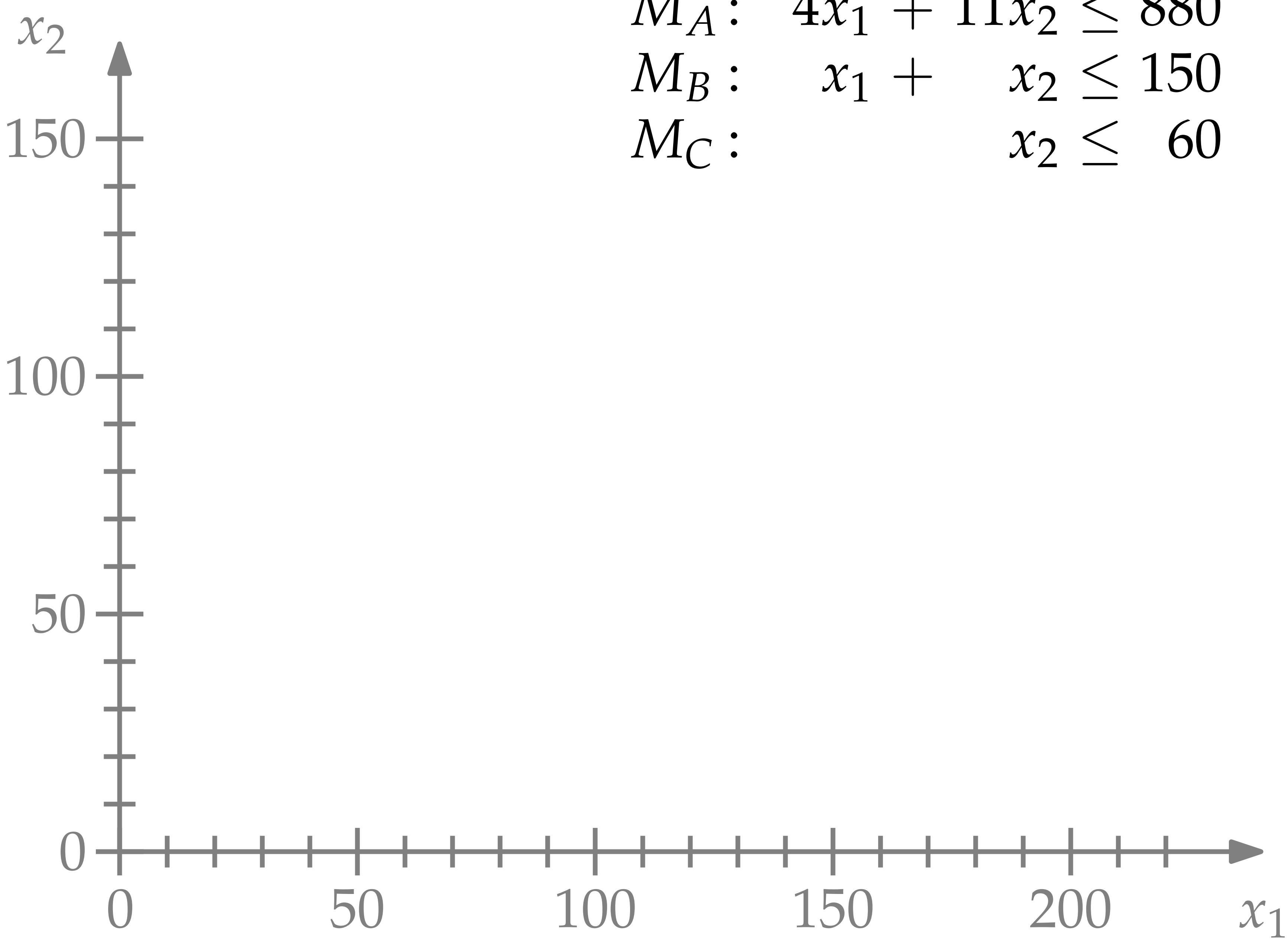
$$
\begin{aligned}
M_A: &\quad 4x_1 + 11x_2 \leq 880 \\
M_B: &\quad x_1 + x_2 \leq 150 \\
M_C: &\quad x_2 \leq 60
\end{aligned}
$$



Solution

Linear constraints:

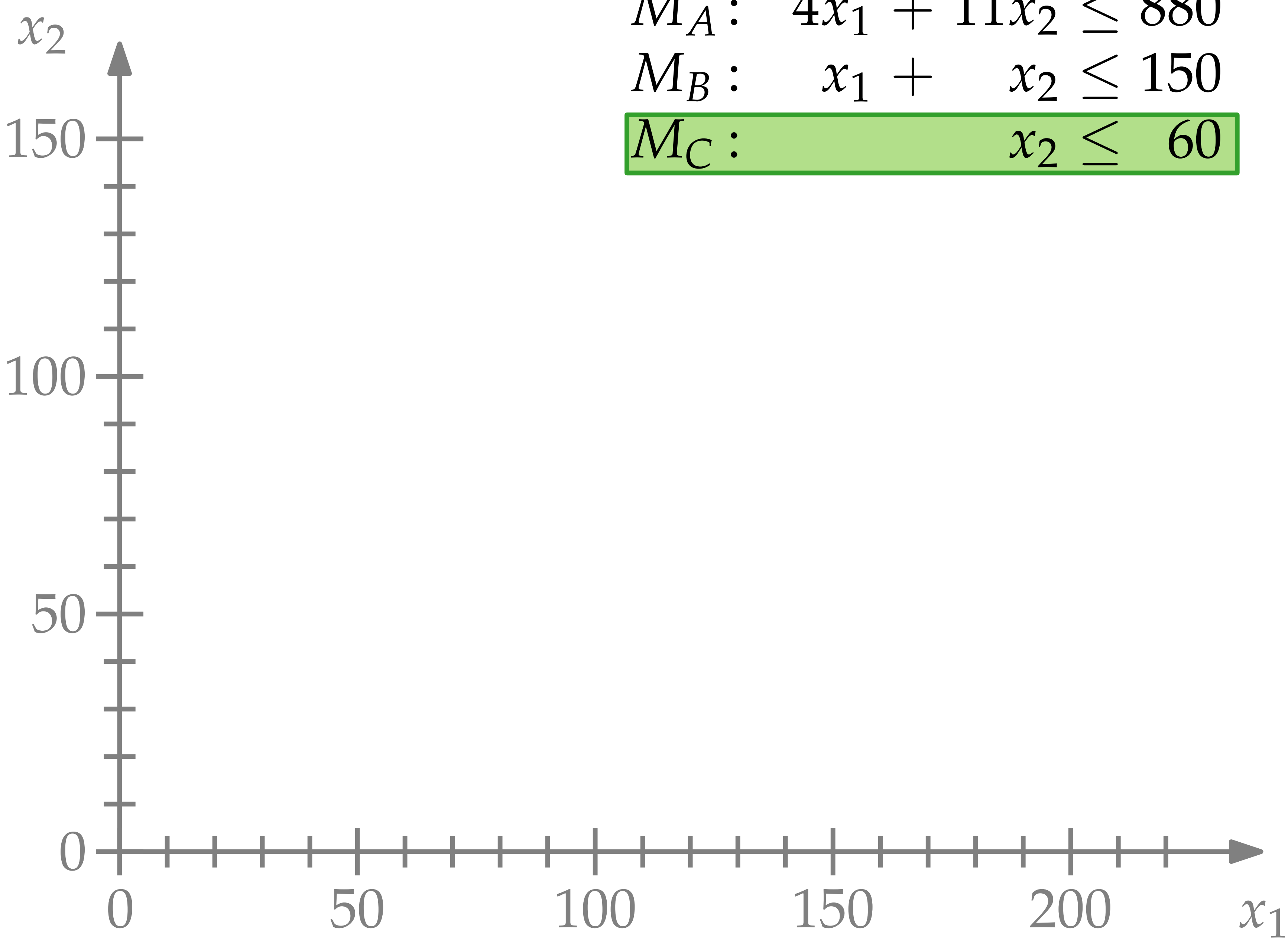
$$
\begin{aligned}
M_A:&\quad 4x_1 + 11x_2 \leq 880\\
M_B:&\quad x_1 + x_2 \leq 150\\
M_C:&\quad x_2 \leq 60
\end{aligned}
$$



Solution

Linear constraints:

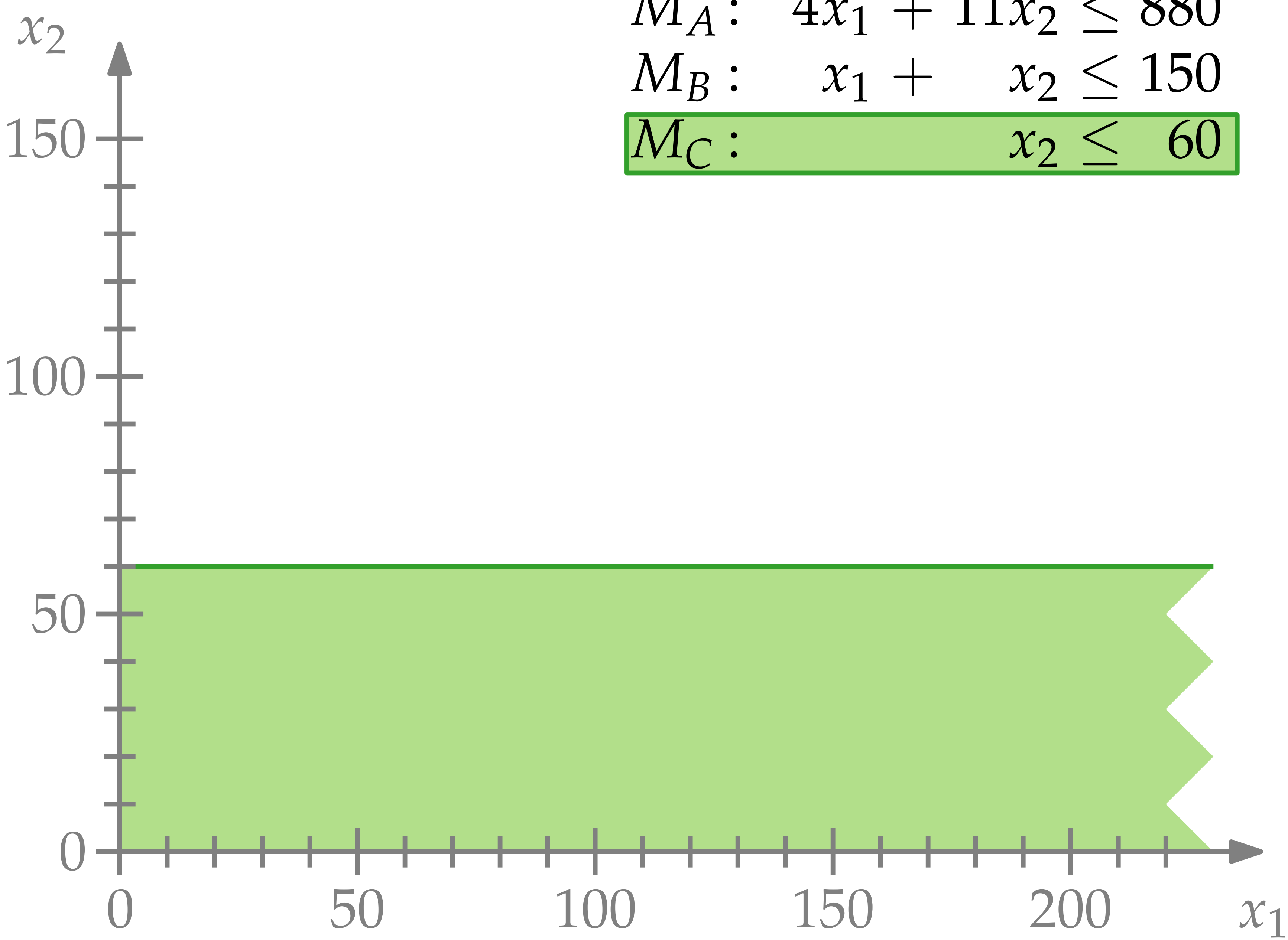
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$



Solution

Linear constraints:

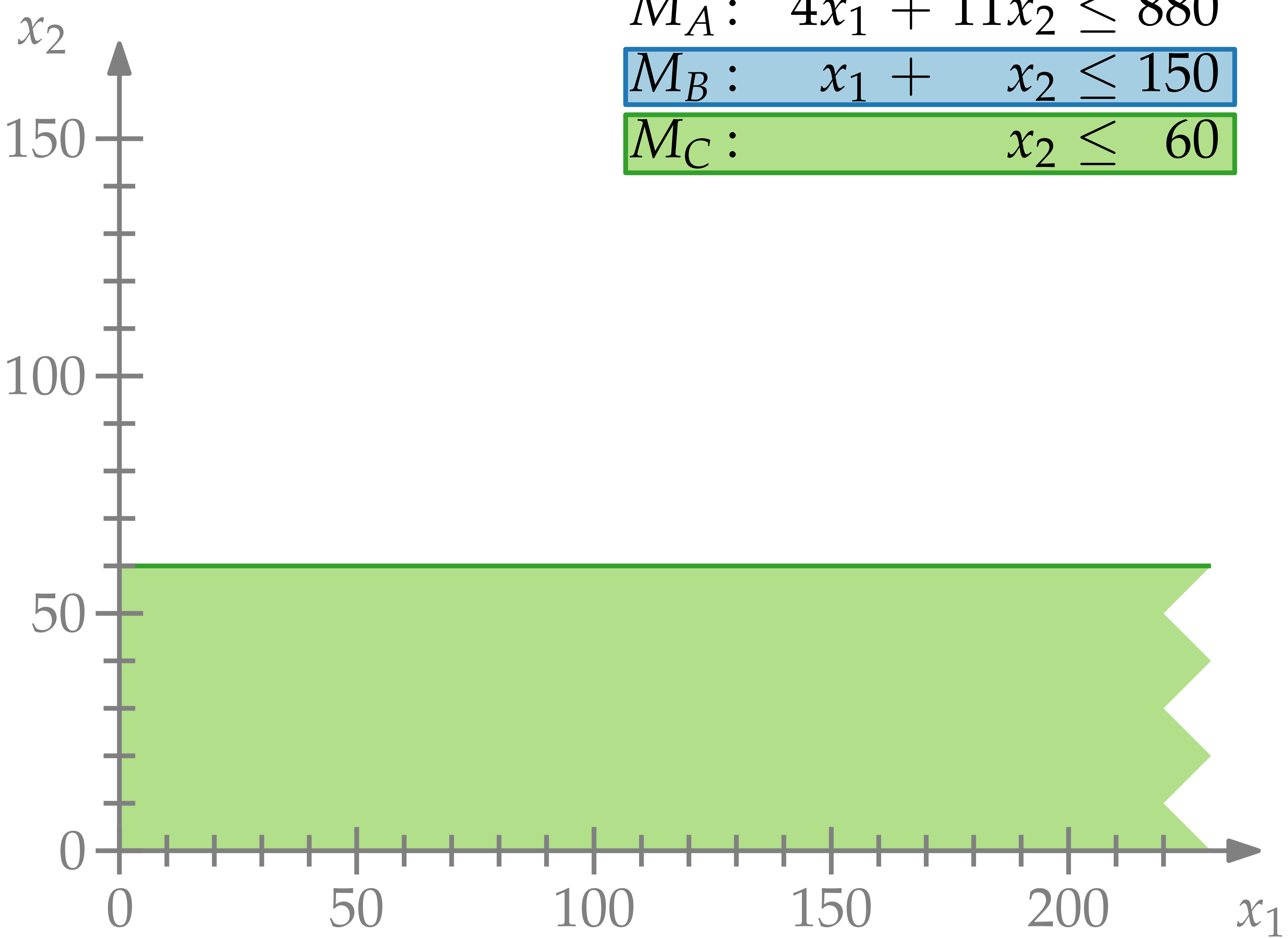
$$
\begin{aligned}
M_A:&\quad 4x_1 + 11x_2 \le 880\\
M_B:&\quad x_1 + x_2 \le 150\\
M_C:&\quad x_2 \le 60
\end{aligned}
$$



Solution

Linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$



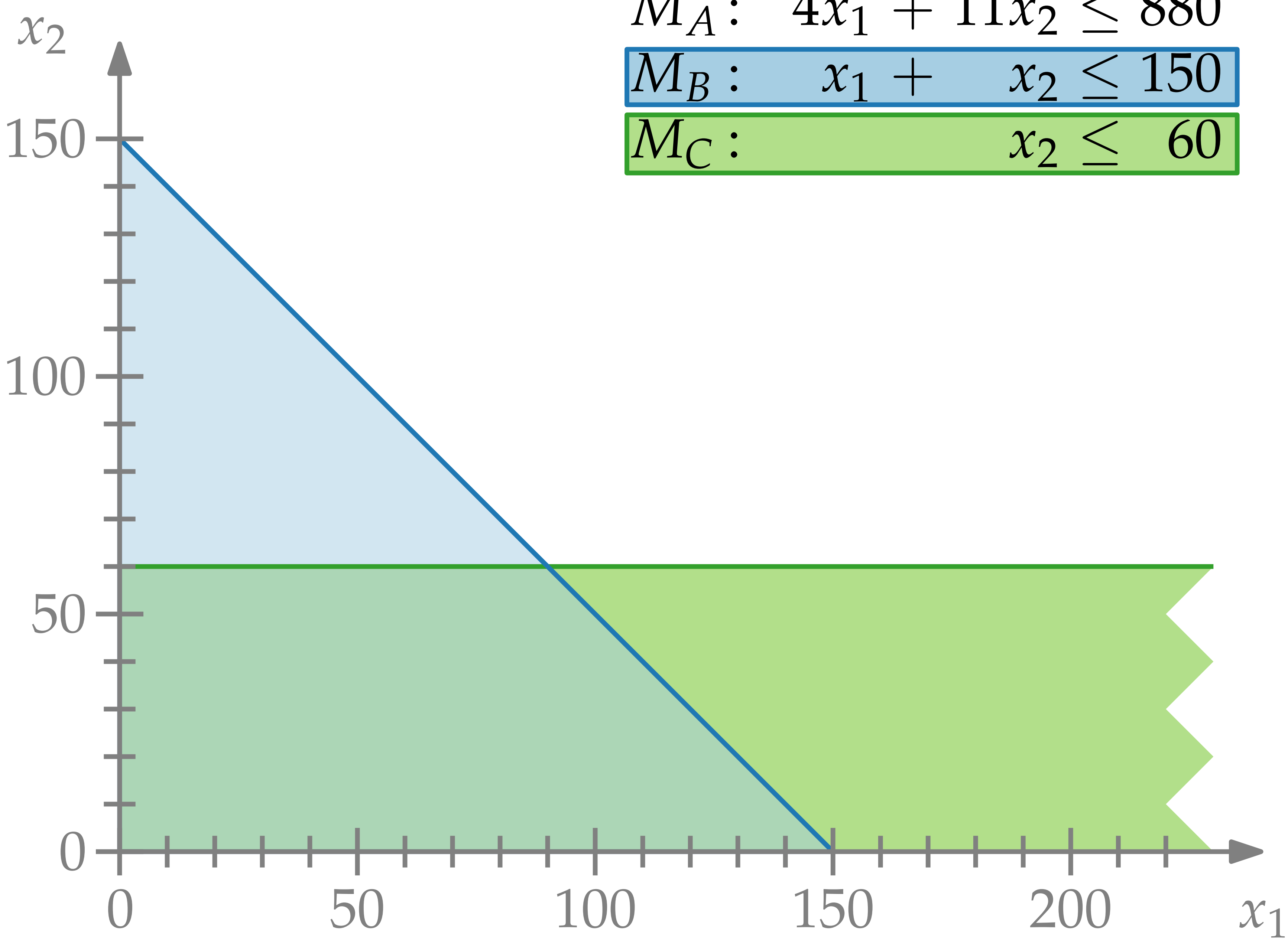
Solution

Linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$

$$M_B: \quad x_1 + x_2 \leq 150$$

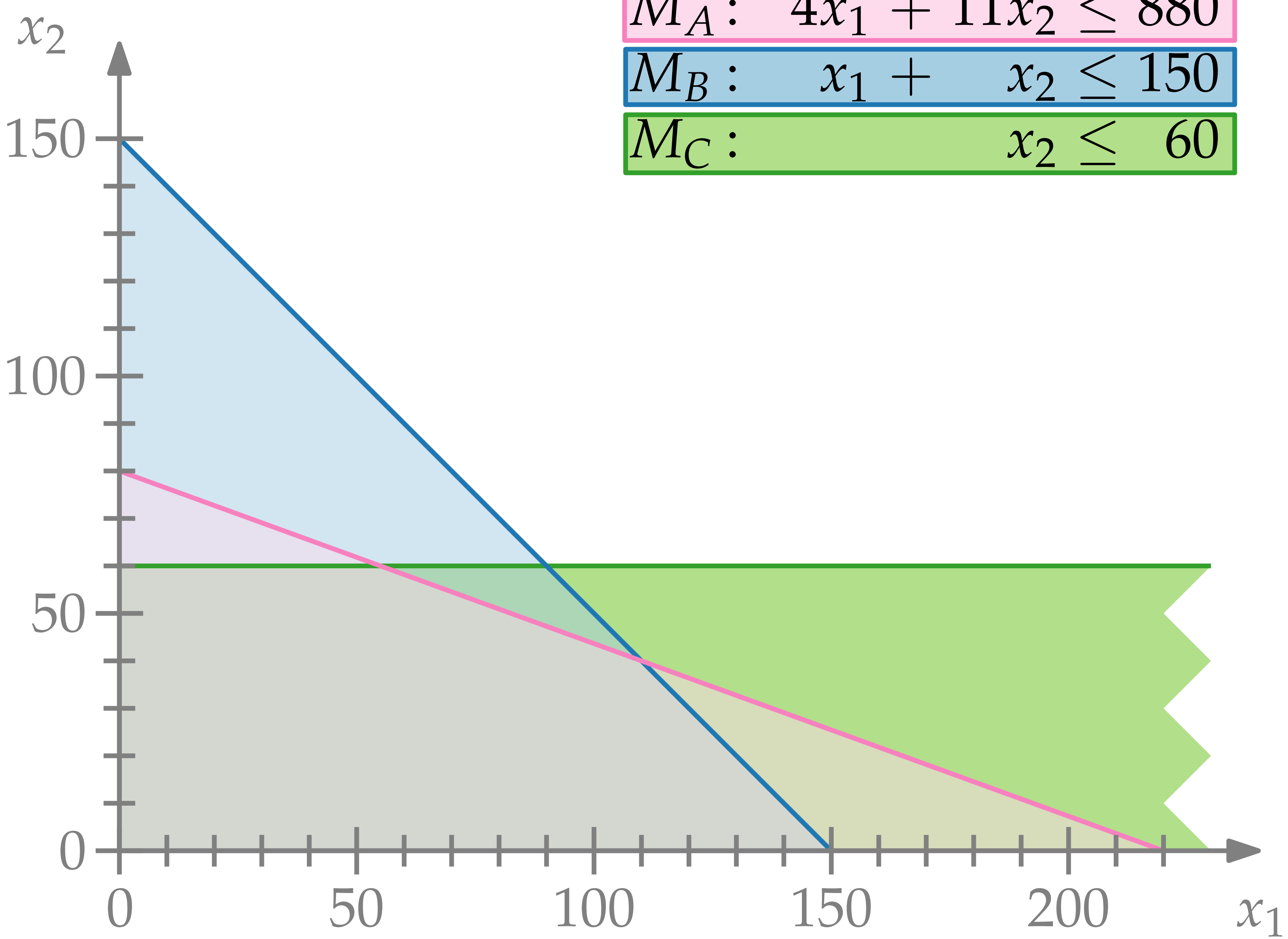
$$M_C: \quad x_2 \leq 60$$



Solution

Linear constraints:

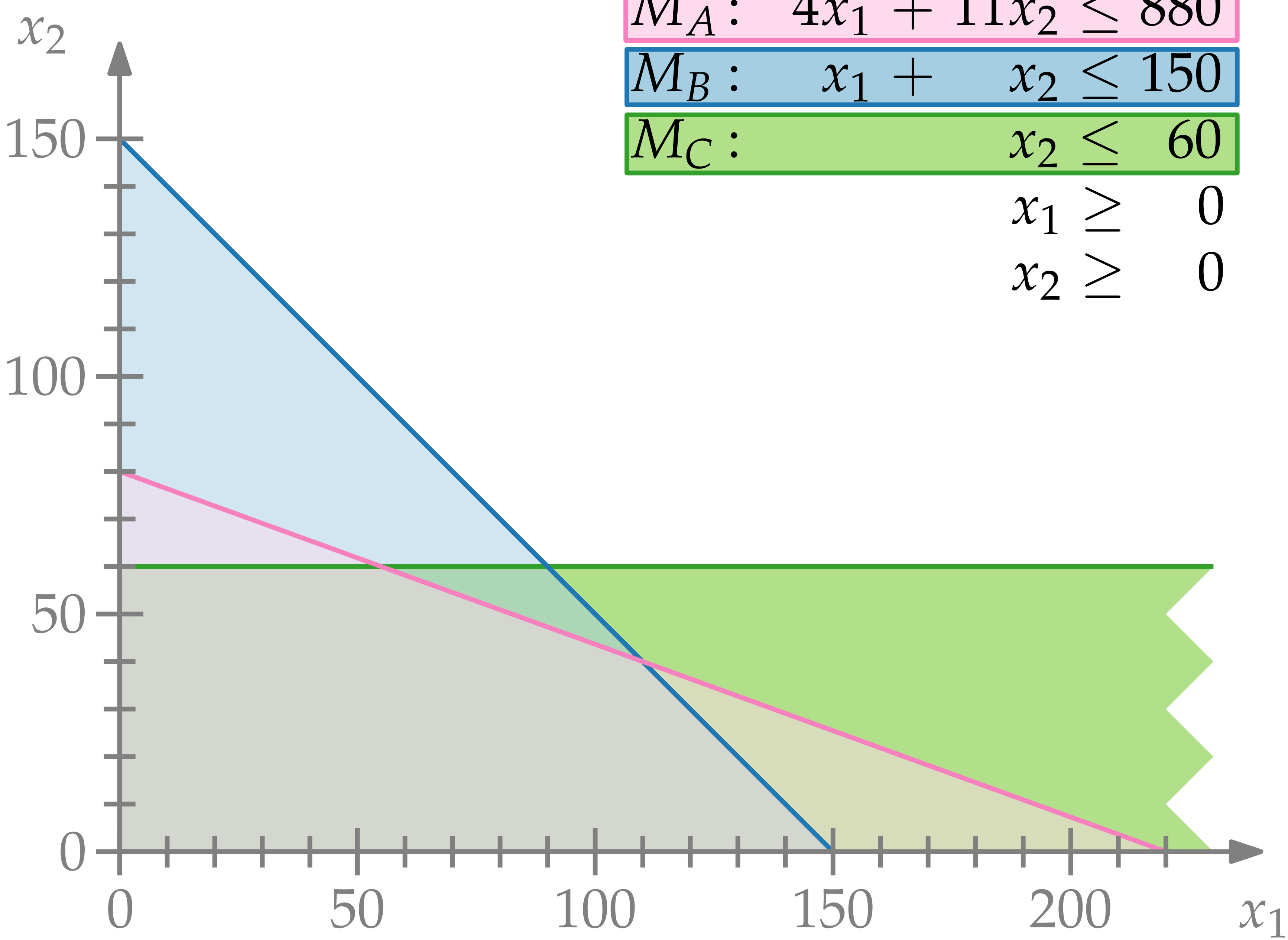
$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$



Solution

Linear constraints:

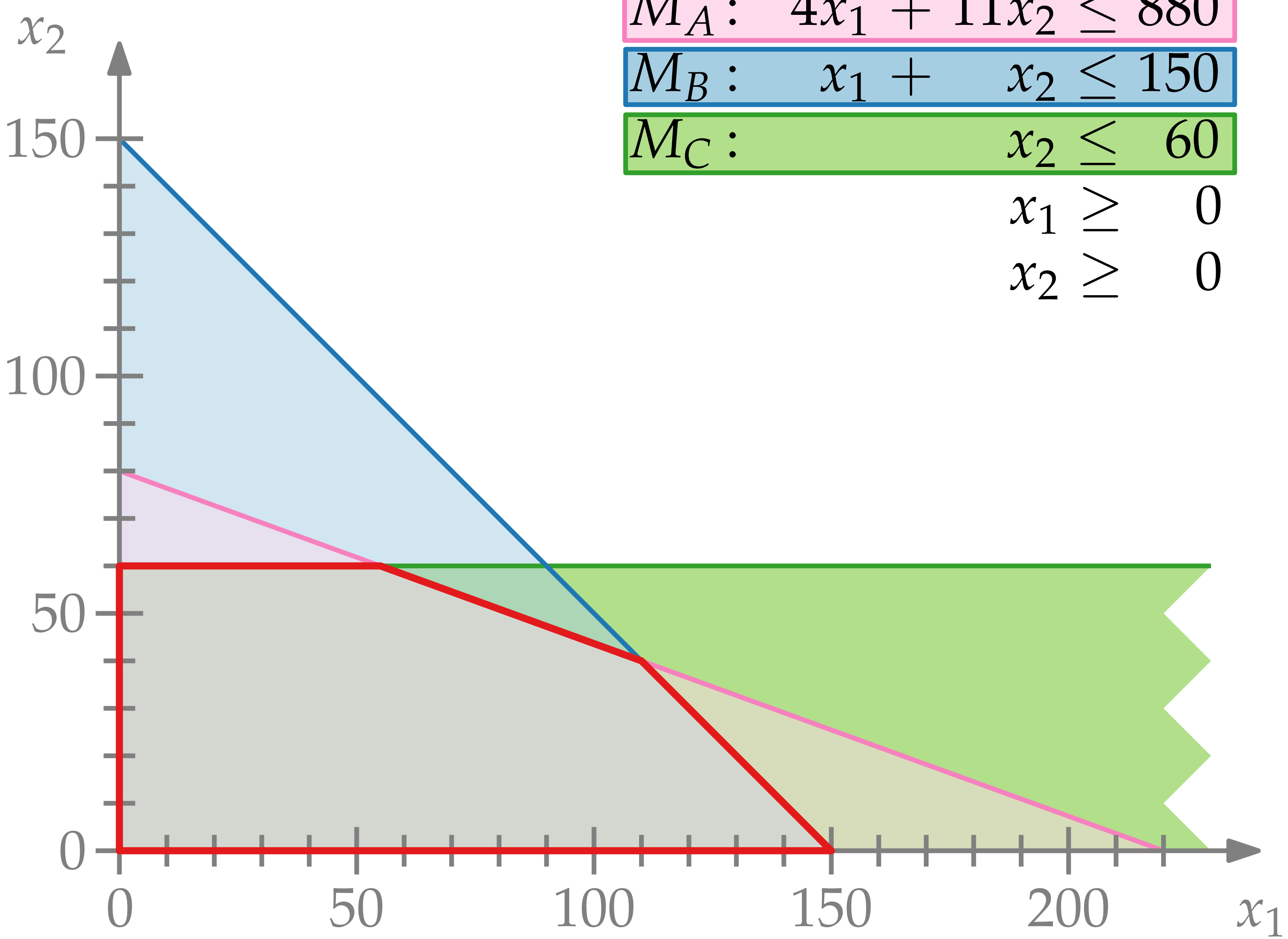
$$\begin{aligned}
M_A: \quad 4x_1 + 11x_2 &\leq 880 \\
M_B: \quad x_1 + x_2 &\leq 150 \\
M_C: \quad x_2 &\leq 60 \\
x_1 &\geq 0 \\
x_2 &\geq 0
\end{aligned}$$



Solution

Linear constraints:

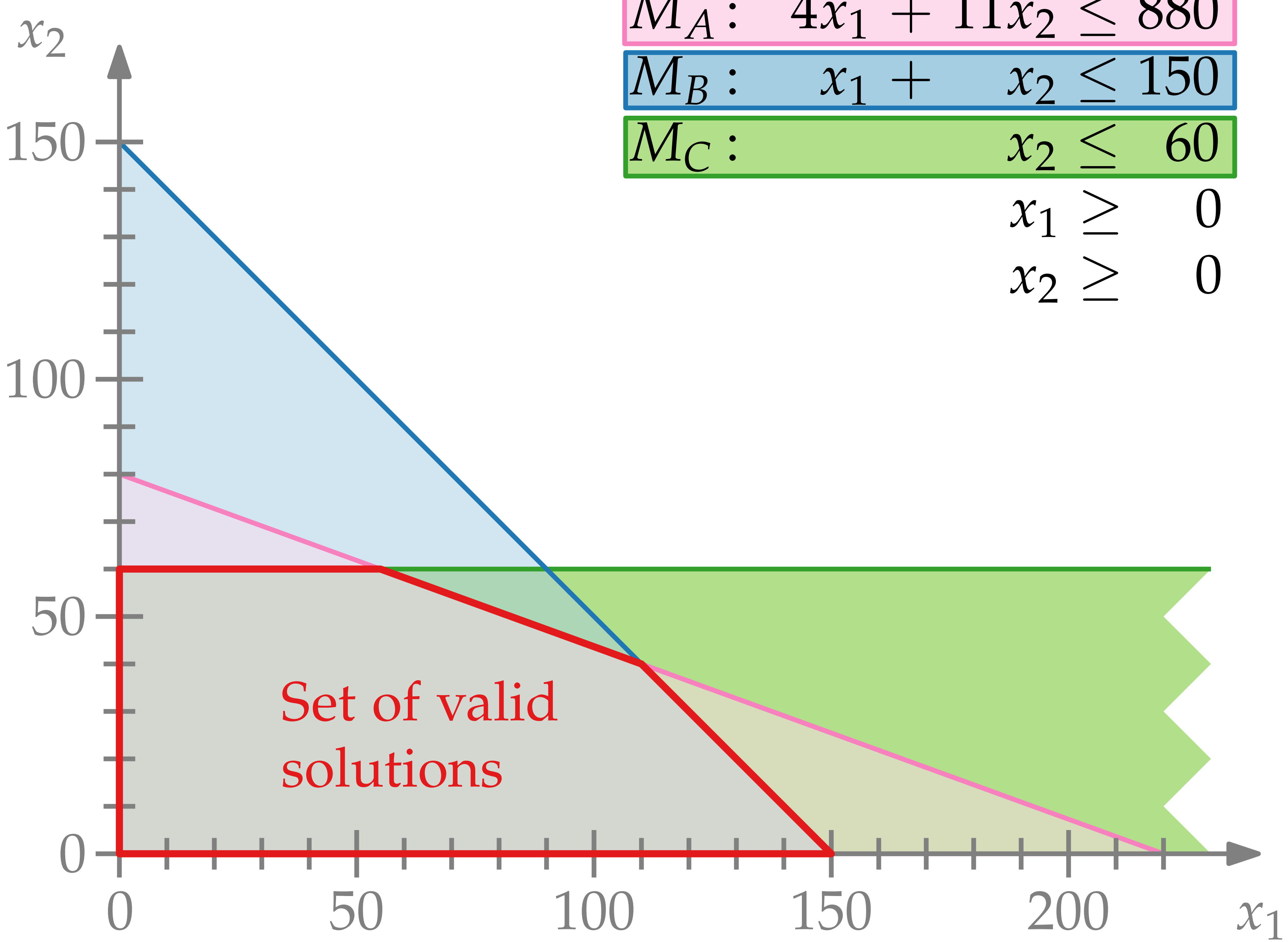
$$
\begin{aligned}
M_A: &\quad 4x_1 + 11x_2 \le 880 \\
M_B: &\quad x_1 + x_2 \le 150 \\
M_C: &\quad x_2 \le 60 \\
&\quad x_1 \ge 0 \\
&\quad x_2 \ge 0
\end{aligned}
$$



Solution

Linear constraints:

$$\begin{aligned} M_A: \quad 4x_1 + 11x_2 &\le 880 \\ M_B: \quad x_1 + x_2 &\le 150 \\ M_C: \quad x_2 &\le 60 \\ x_1 &\ge 0 \\ x_2 &\ge 0 \end{aligned}$$



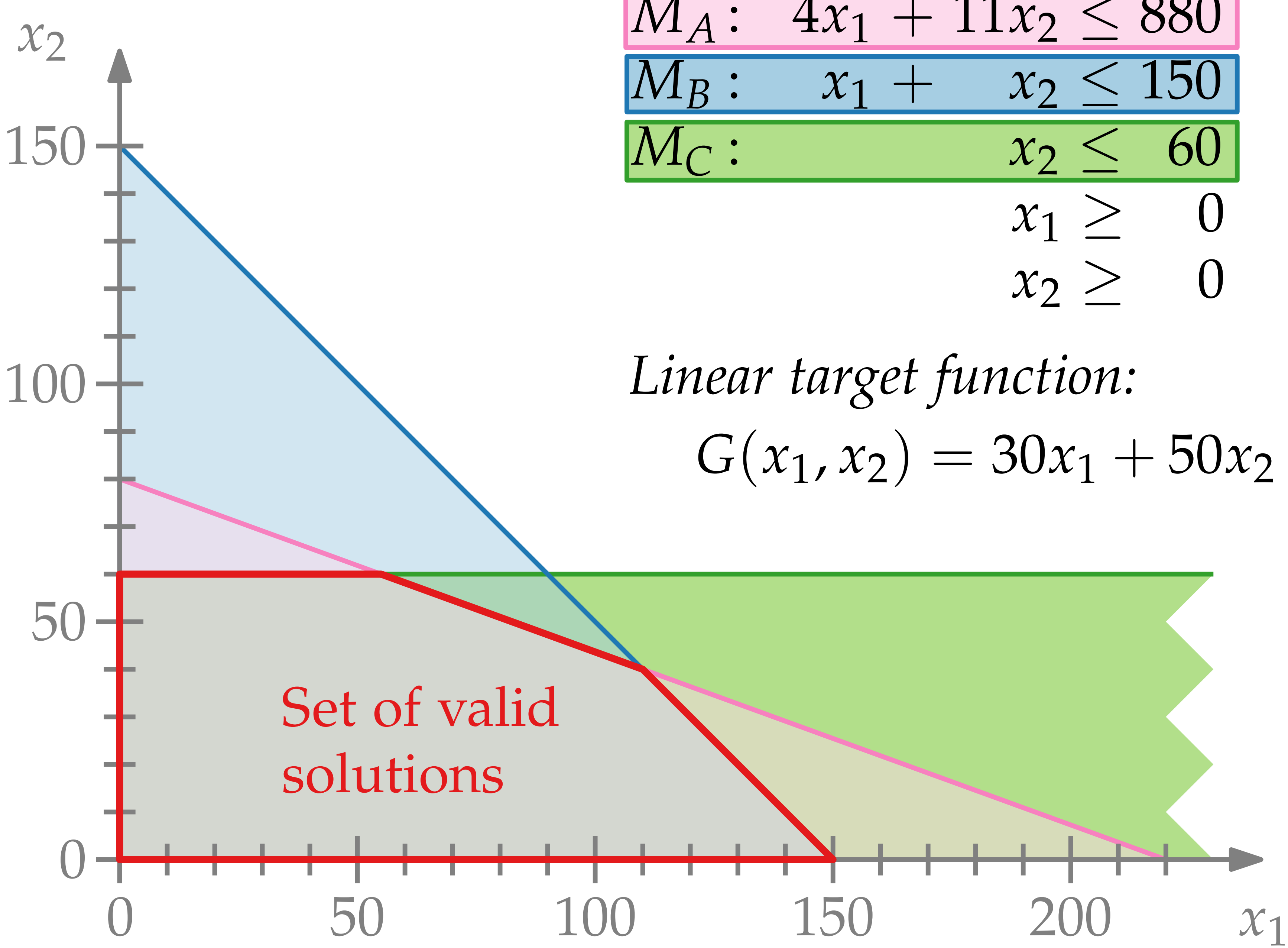
Solution

Linear constraints:

$$
\begin{aligned}
M_A: \quad 4x_1 + 11x_2 &\leq 880 \\
M_B: \quad x_1 + x_2 &\leq 150 \\
M_C: \quad x_2 &\leq 60 \\
x_1 &\geq 0 \\
x_2 &\geq 0
\end{aligned}
$$

Linear target function:

$$G(x_1, x_2) = 30x_1 + 50x_2$$



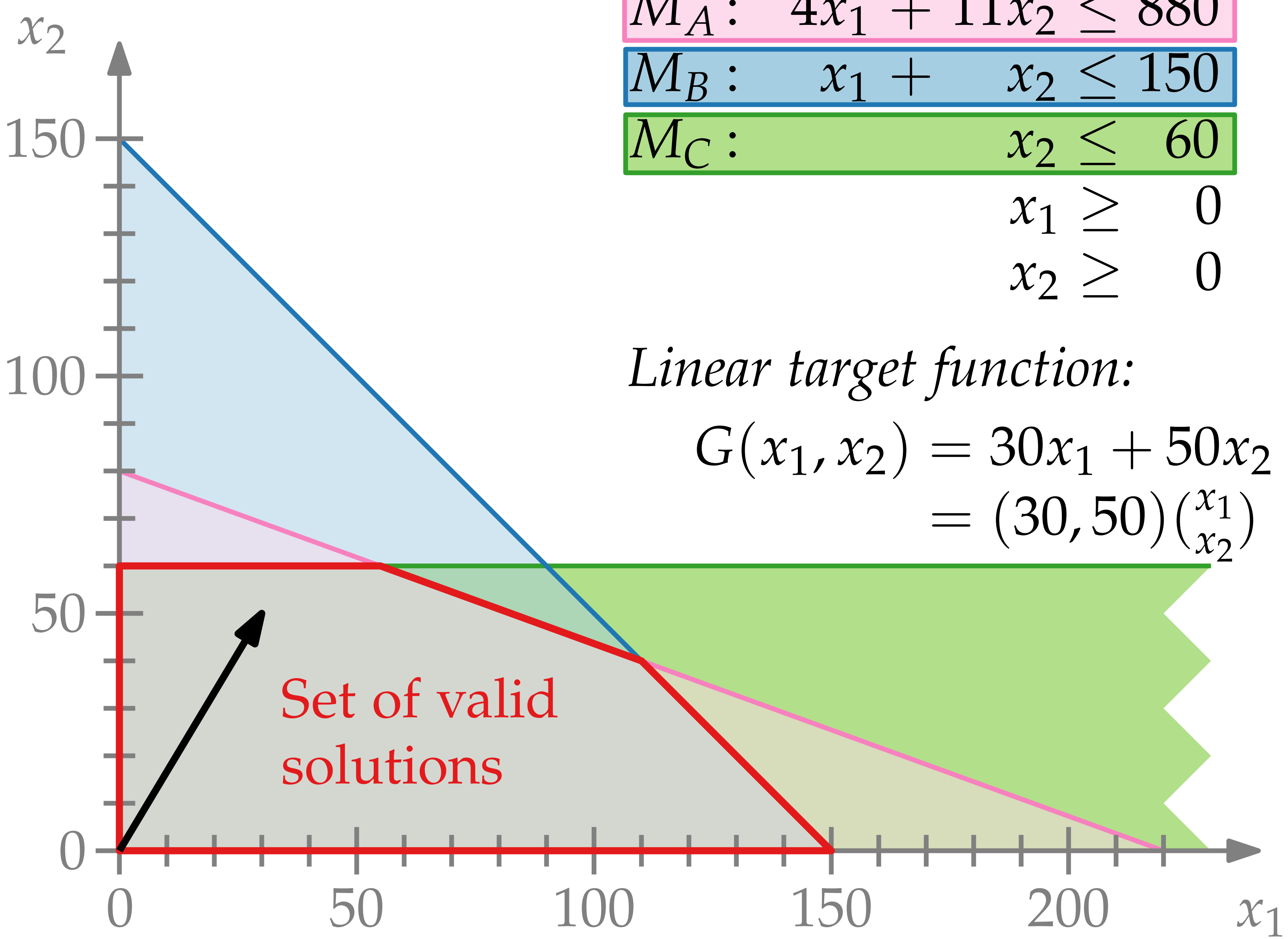
Solution

Linear constraints:

$$\begin{aligned} M_A: \quad 4x_1 + 11x_2 &\le 880 \\ M_B: \quad x_1 + x_2 &\le 150 \\ M_C: \quad x_2 &\le 60 \\ x_1 &\ge 0 \\ x_2 &\ge 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$



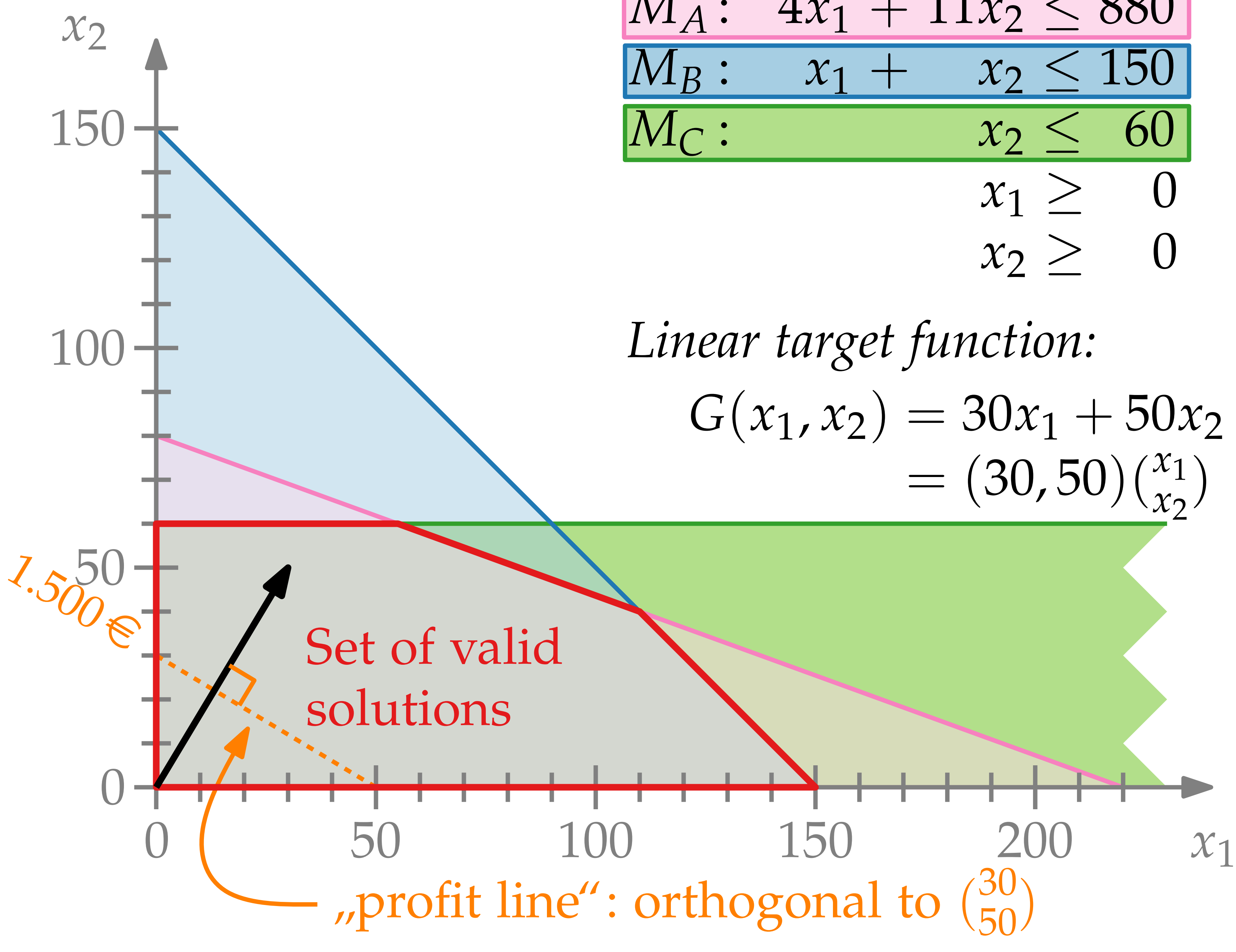
Solution

Linear constraints:

$$\begin{aligned} M_A: \quad 4x_1 + 11x_2 &\leq 880 \\ M_B: \quad x_1 + x_2 &\leq 150 \\ M_C: \quad x_2 &\leq 60 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$



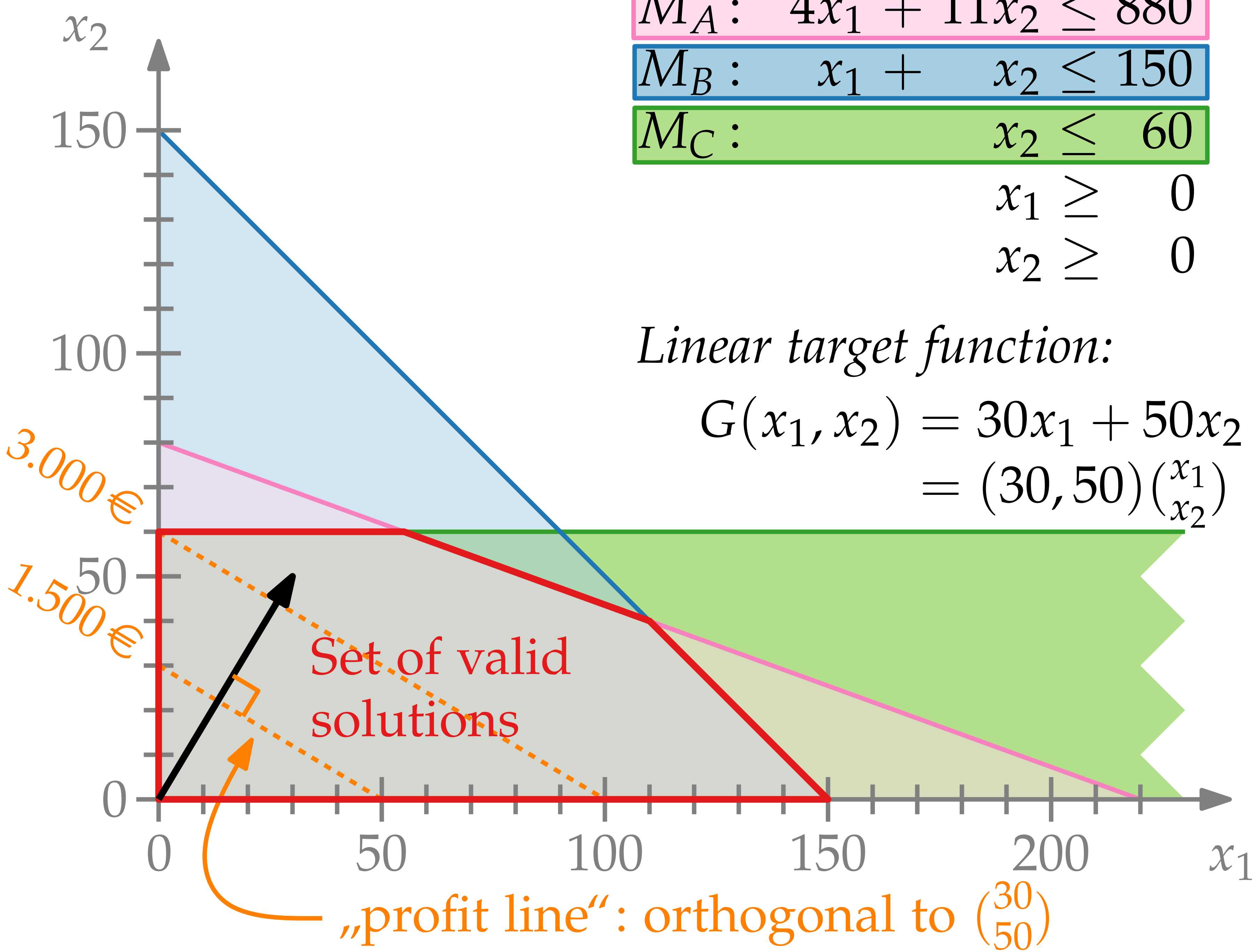
Solution

Linear constraints:

$$
\begin{aligned}
M_A: &\quad 4x_1 + 11x_2 \leq 880 \\
M_B: &\quad x_1 + x_2 \leq 150 \\
M_C: &\quad x_2 \leq 60 \\
&\quad x_1 \geq 0 \\
&\quad x_2 \geq 0
\end{aligned}
$$

Linear target function:

$$
\begin{aligned}
G(x_1, x_2) &= 30x_1 + 50x_2 \\
&= (30, 50)\binom{x_1}{x_2}
\end{aligned}
$$



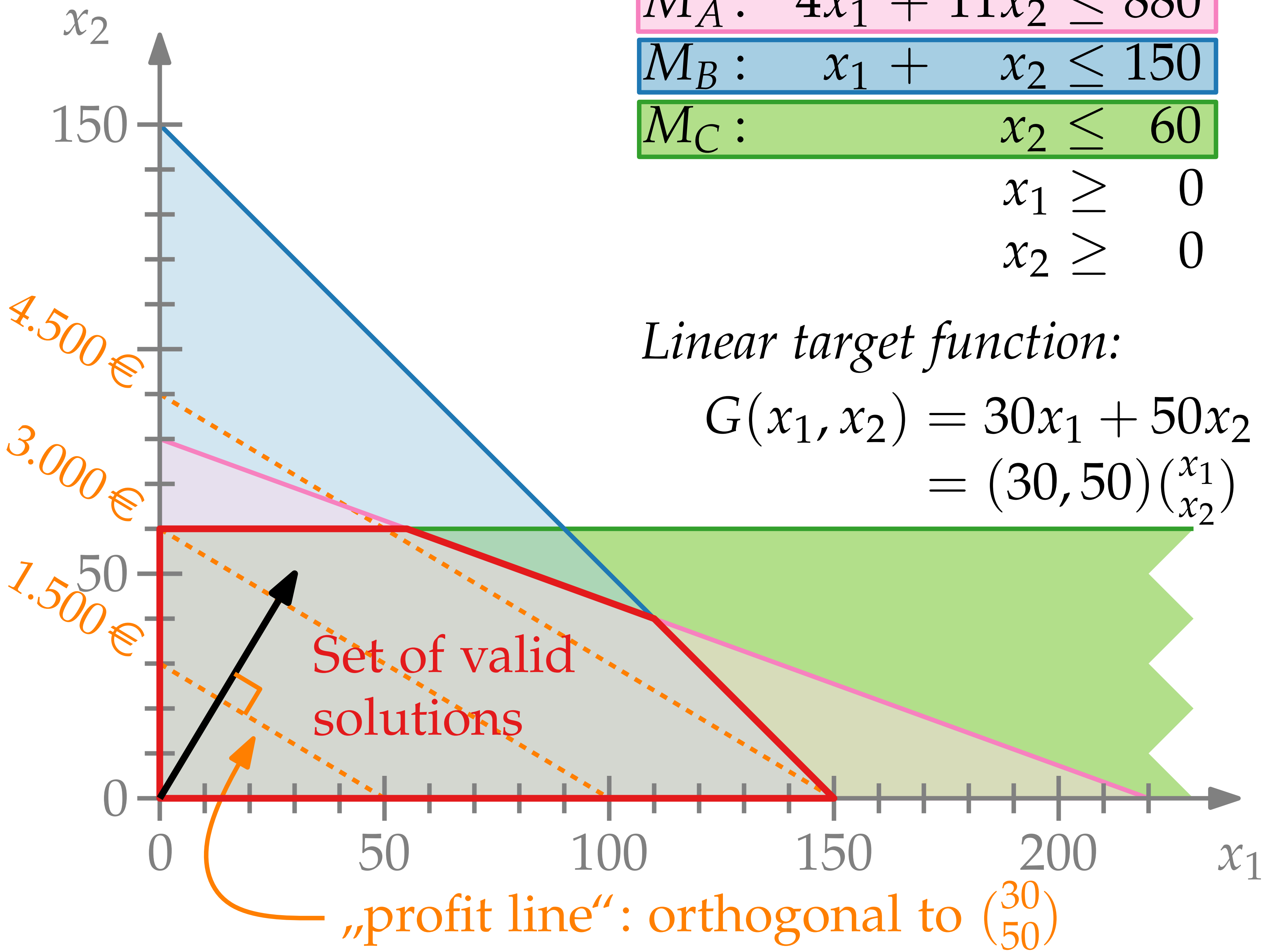
Solution

Linear constraints:

$$\begin{aligned} M_A: \quad 4x_1 + 11x_2 &\le 880 \\ M_B: \quad x_1 + x_2 &\le 150 \\ M_C: \quad x_2 &\le 60 \\ x_1 &\ge 0 \\ x_2 &\ge 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$



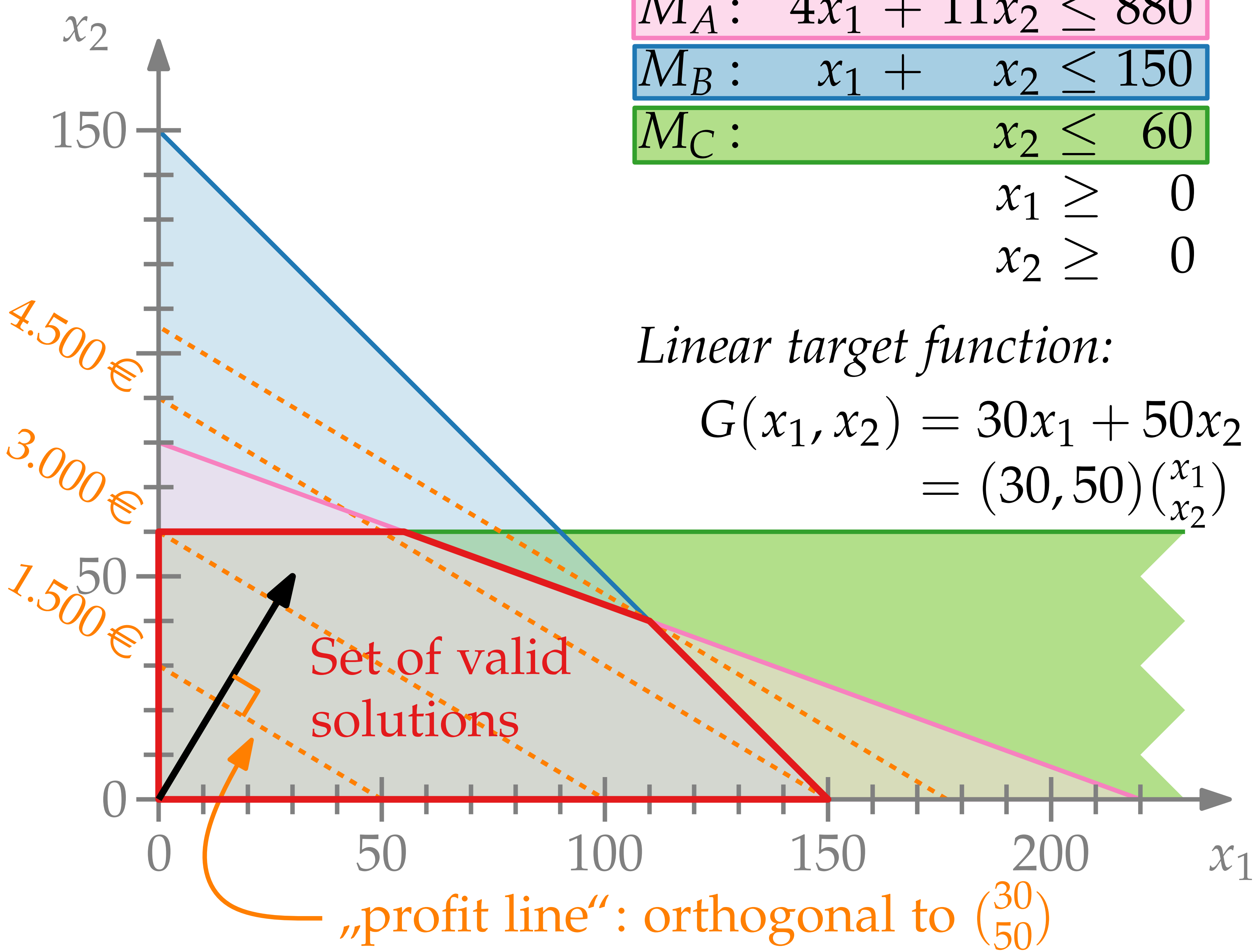
Solution

Linear constraints:

$$
\begin{aligned}
M_A: \quad 4x_1 + 11x_2 &\leq 880 \\
M_B: \quad x_1 + x_2 &\leq 150 \\
M_C: \quad x_2 &\leq 60 \\
x_1 &\geq 0 \\
x_2 &\geq 0
\end{aligned}
$$

Linear target function:

$$
\begin{aligned}
G(x_1, x_2) &= 30x_1 + 50x_2 \\
&= (30, 50)\binom{x_1}{x_2}
\end{aligned}
$$



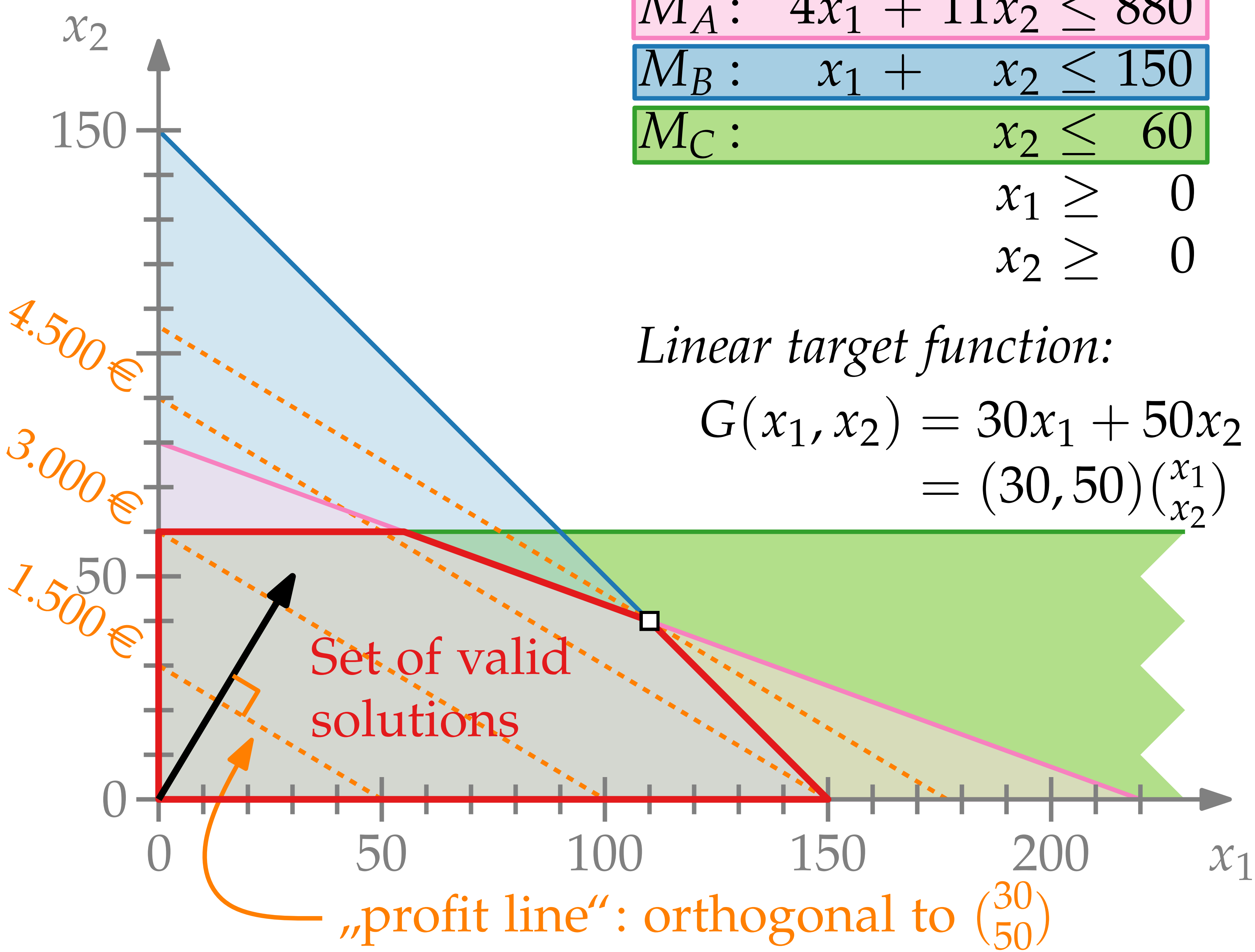
Solution

Linear constraints:

$$M_A: \quad 4x_1 + 11x_2 \leq 880$$
$$M_B: \quad x_1 + x_2 \leq 150$$
$$M_C: \quad x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

Linear target function:

$$G(x_1, x_2) = 30x_1 + 50x_2 = (30, 50)\binom{x_1}{x_2}$$



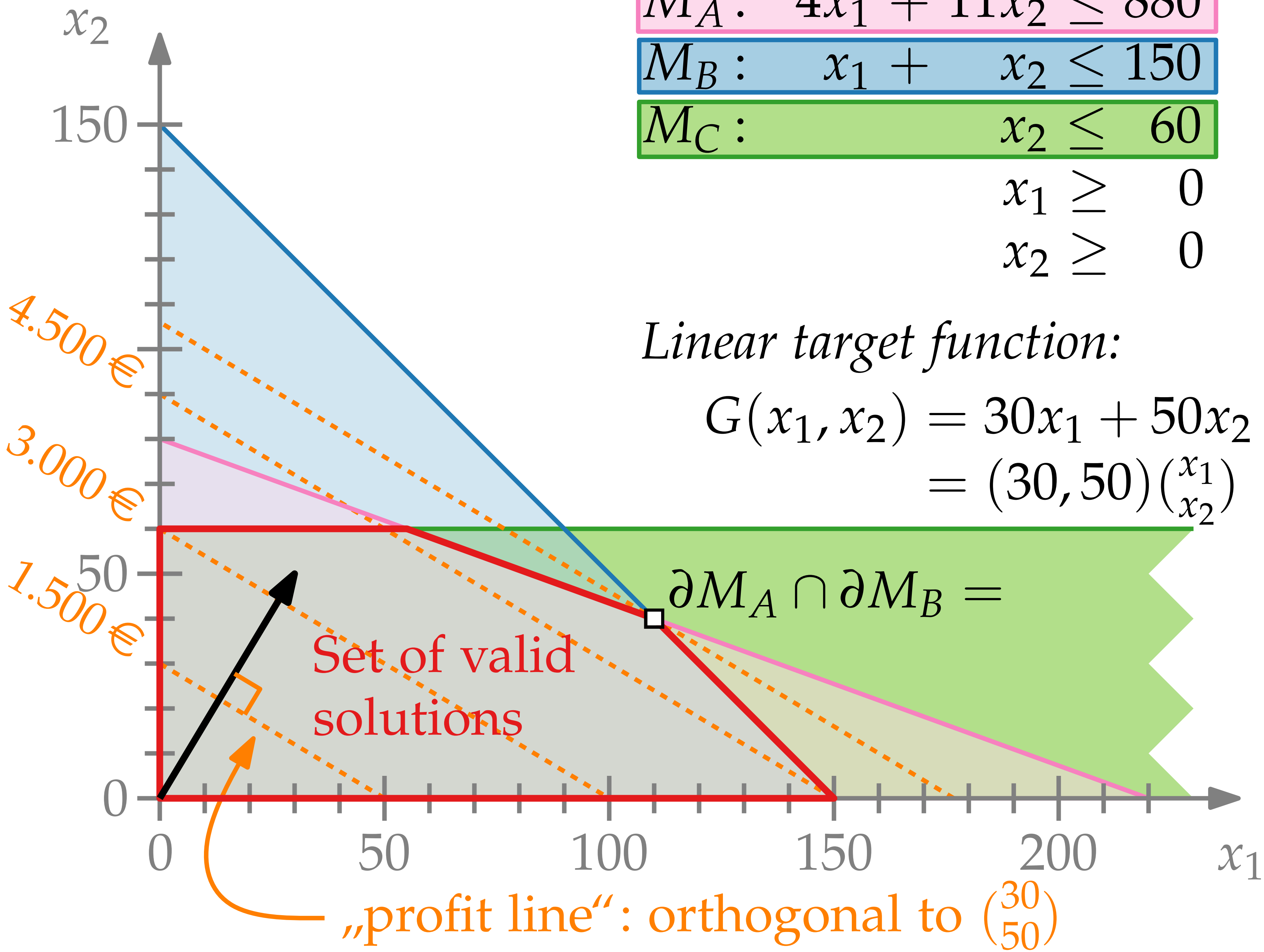
Solution

Linear constraints:

$$\begin{aligned} M_A: \quad 4x_1 + 11x_2 &\leq 880 \\ M_B: \quad x_1 + x_2 &\leq 150 \\ M_C: \quad x_2 &\leq 60 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$



Solution

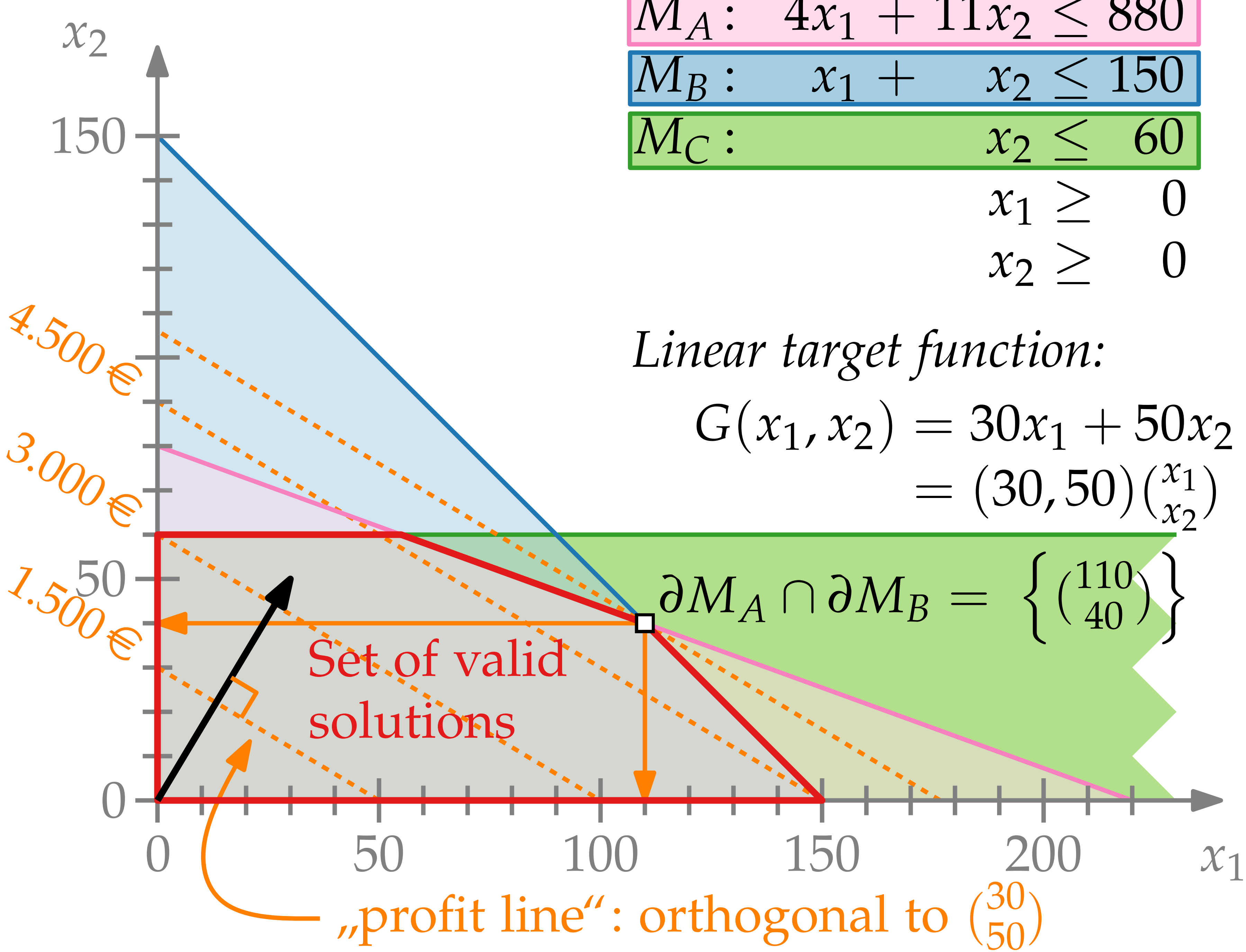
Linear constraints:

$$\begin{aligned} M_A:&\quad 4x_1 + 11x_2 \leq 880 \\ M_B:&\quad x_1 + x_2 \leq 150 \\ M_C:&\quad x_2 \leq 60 \\ &\quad x_1 \geq 0 \\ &\quad x_2 \geq 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$

$$\partial M_A \cap \partial M_B = \left\{ \binom{110}{40} \right\}$$



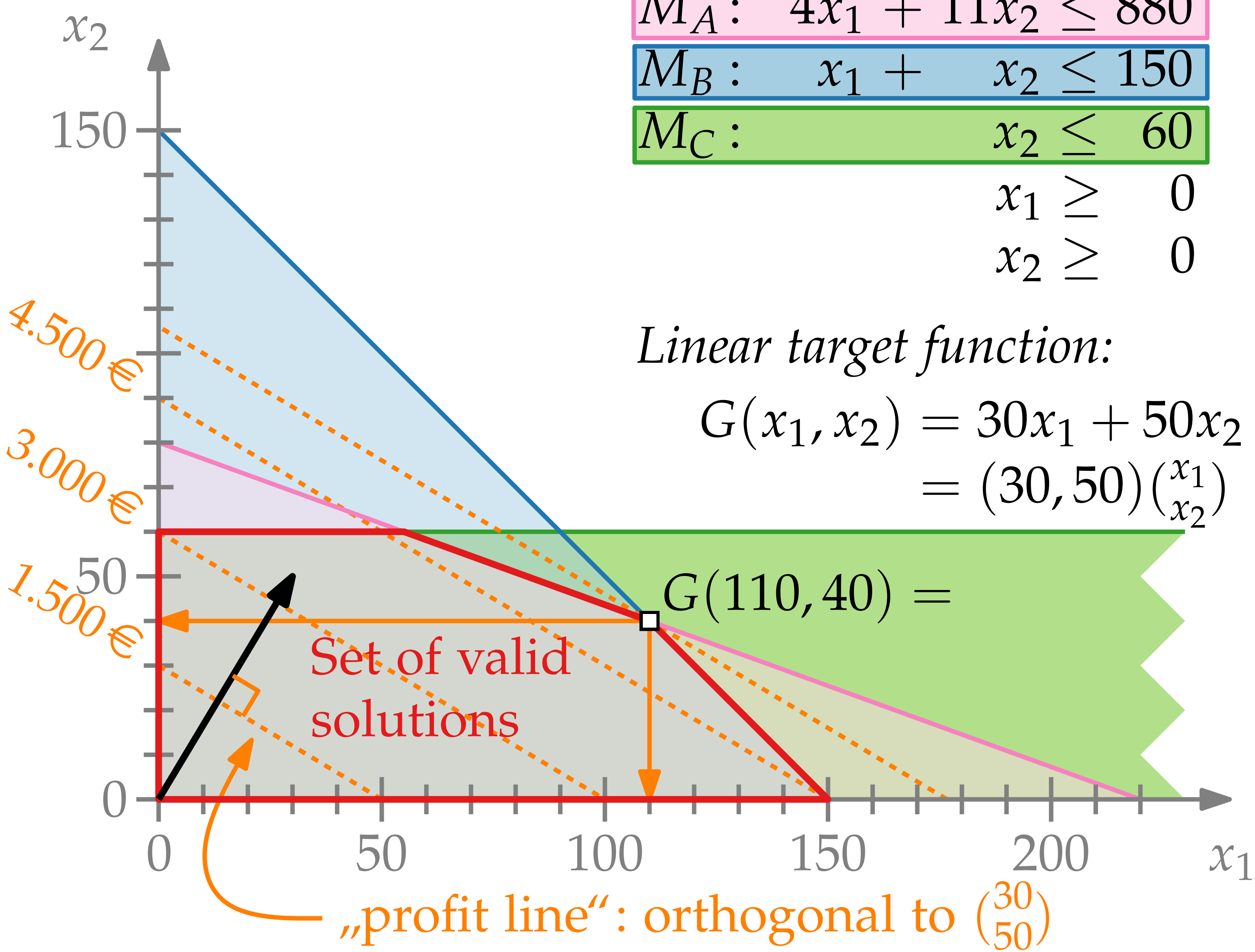
Solution

Linear constraints:

$$\begin{aligned} M_A: &\quad 4x_1 + 11x_2 \leq 880 \\ M_B: &\quad x_1 + x_2 \leq 150 \\ M_C: &\quad x_2 \leq 60 \\ &\quad x_1 \geq 0 \\ &\quad x_2 \geq 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$



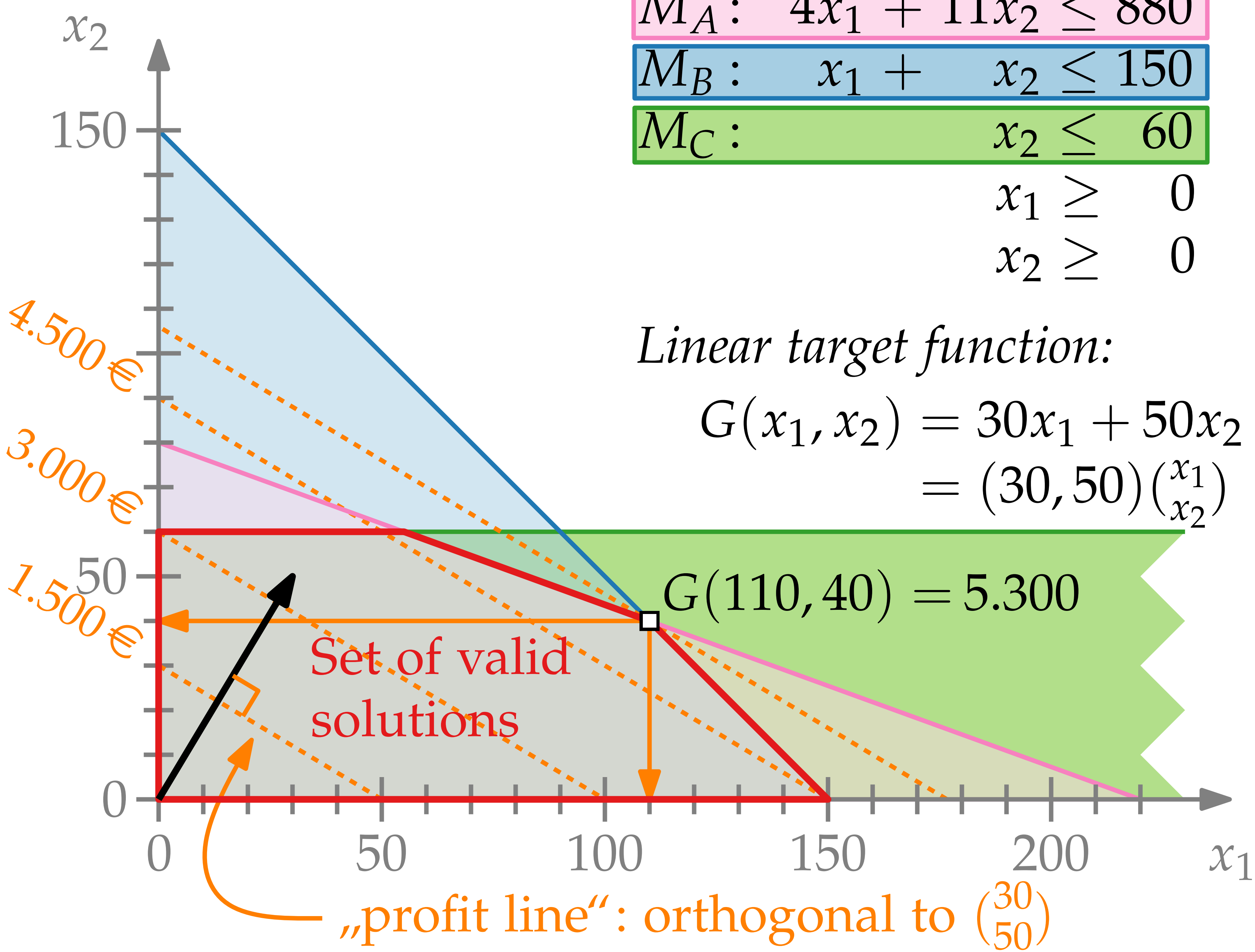
Solution

Linear constraints:

$$
\begin{aligned}
M_A: &\quad 4x_1 + 11x_2 \leq 880 \\
M_B: &\quad x_1 + x_2 \leq 150 \\
M_C: &\quad x_2 \leq 60 \\
& \quad x_1 \geq 0 \\
& \quad x_2 \geq 0
\end{aligned}
$$

Linear target function:

$$
\begin{aligned}
G(x_1, x_2) &= 30x_1 + 50x_2 \\
&= (30, 50)\binom{x_1}{x_2}
\end{aligned}
$$



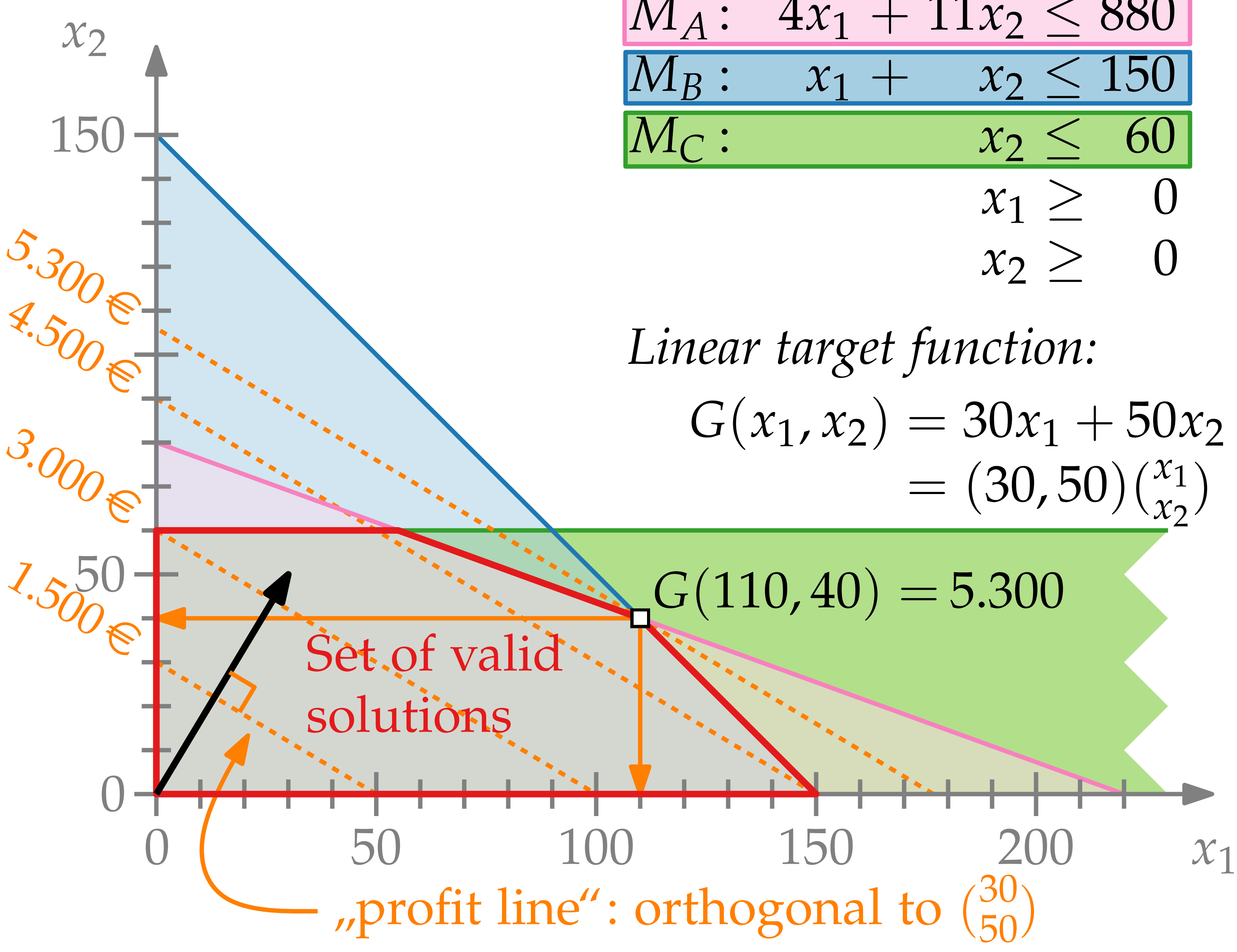
Solution

Linear constraints:

$$
\begin{aligned}
M_A: \quad 4x_1 + 11x_2 &\leq 880 \\
M_B: \quad x_1 + x_2 &\leq 150 \\
M_C: \quad x_2 &\leq 60 \\
x_1 &\geq 0 \\
x_2 &\geq 0
\end{aligned}
$$

Linear target function:

$$
G(x_1, x_2) = 30x_1 + 50x_2 = (30, 50)\binom{x_1}{x_2}
$$



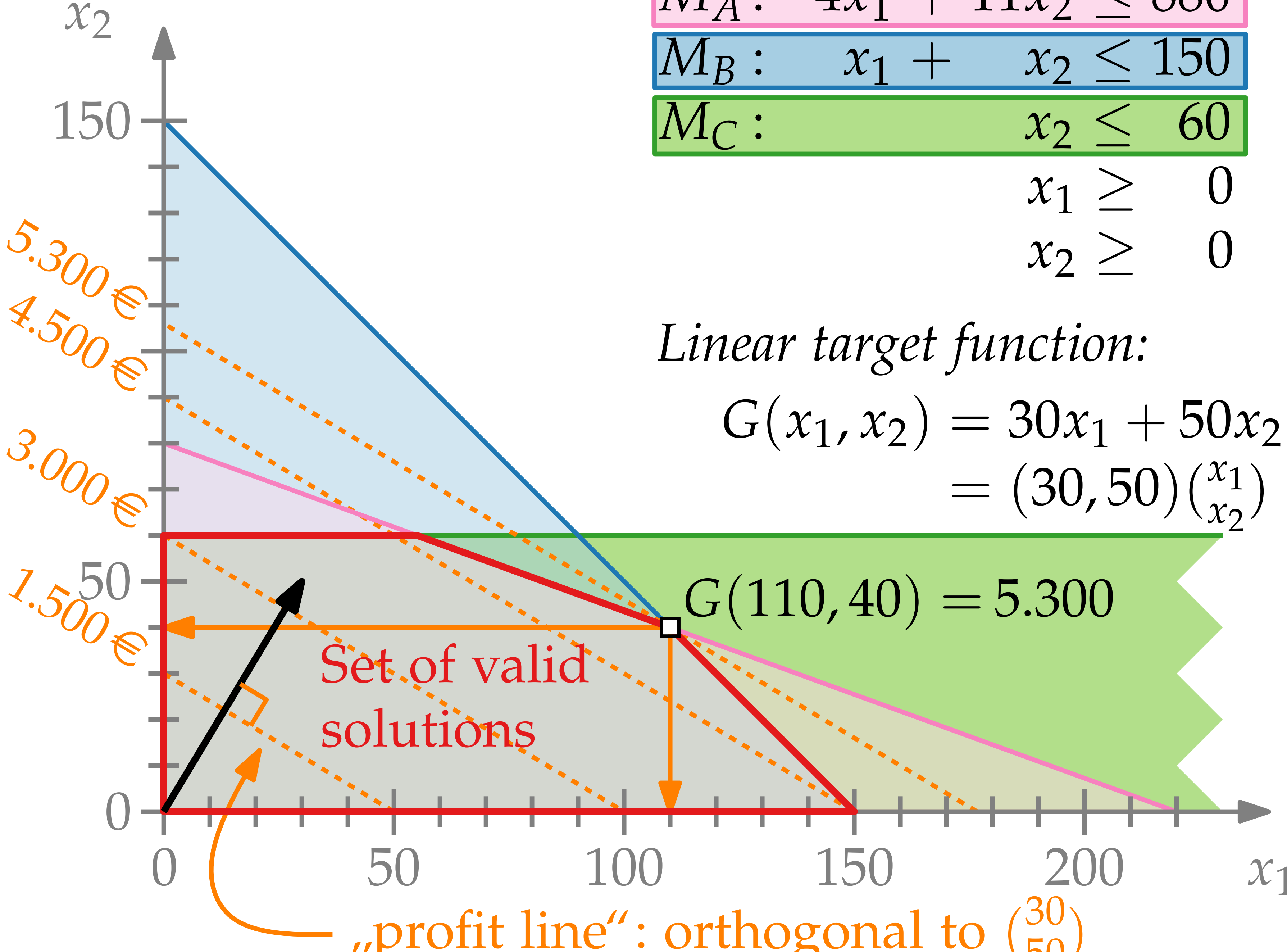
Solution

Linear constraints:

$$\begin{aligned} M_A:\quad 4x_1 + 11x_2 &\leq 880 \\ M_B:\quad x_1 + x_2 &\leq 150 \\ M_C:\quad x_2 &\leq 60 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Linear target function:

$$\begin{aligned} G(x_1, x_2) &= 30x_1 + 50x_2 \\ &= (30, 50)\binom{x_1}{x_2} \end{aligned}$$



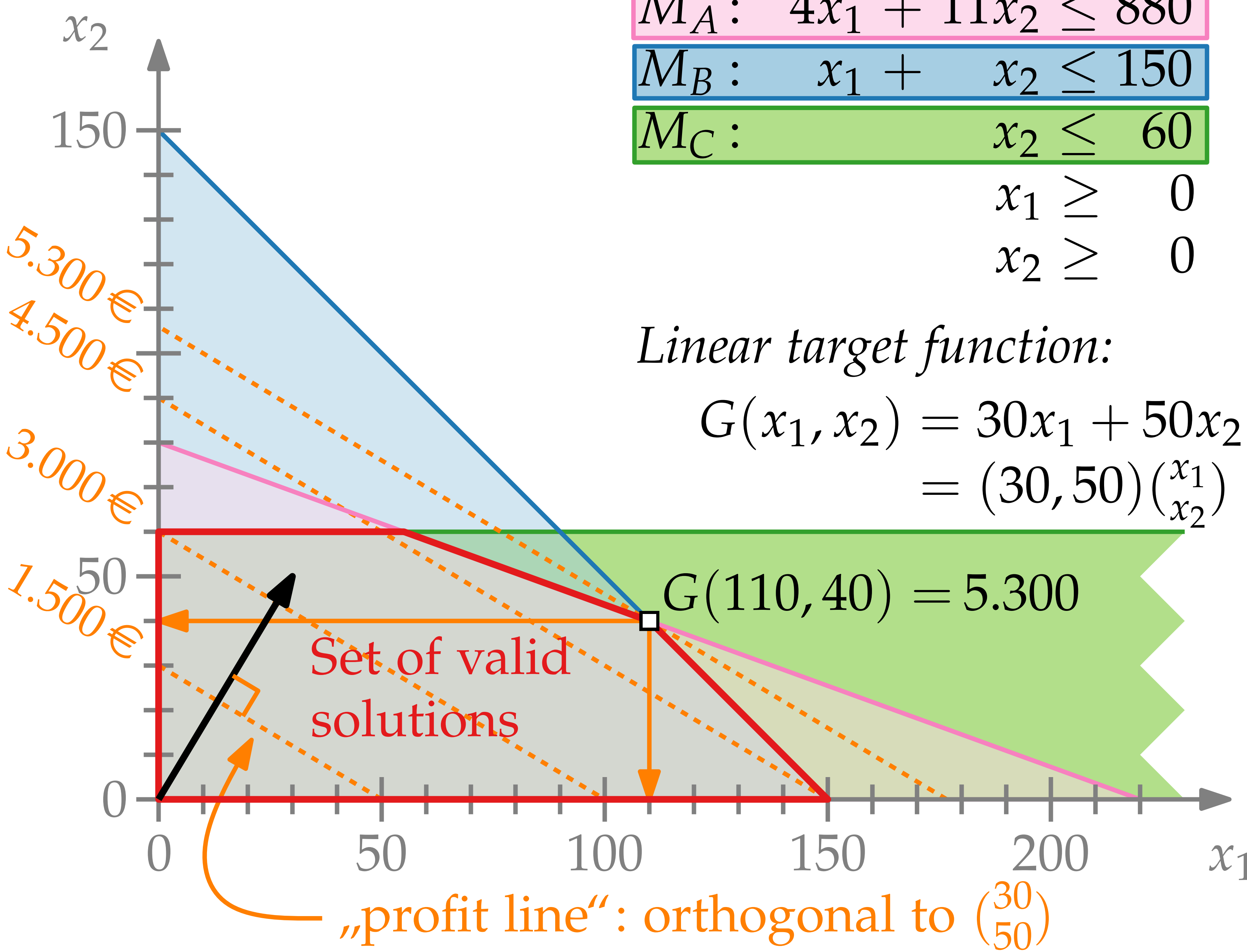
Solution

Linear constraints:

$$
\begin{aligned}
M_A:&\quad 4x_1 + 11x_2 \leq 880\\
M_B:&\quad x_1 + x_2 \leq 150\\
M_C:&\quad x_2 \leq 60\\
&\quad x_1 \geq 0\\
&\quad x_2 \geq 0
\end{aligned}
\qquad Ax \leq b
$$

Linear target function:

$$
\begin{aligned}
G(x_1, x_2) &= 30x_1 + 50x_2\\
&= (30, 50)\binom{x_1}{x_2}
\end{aligned}
$$



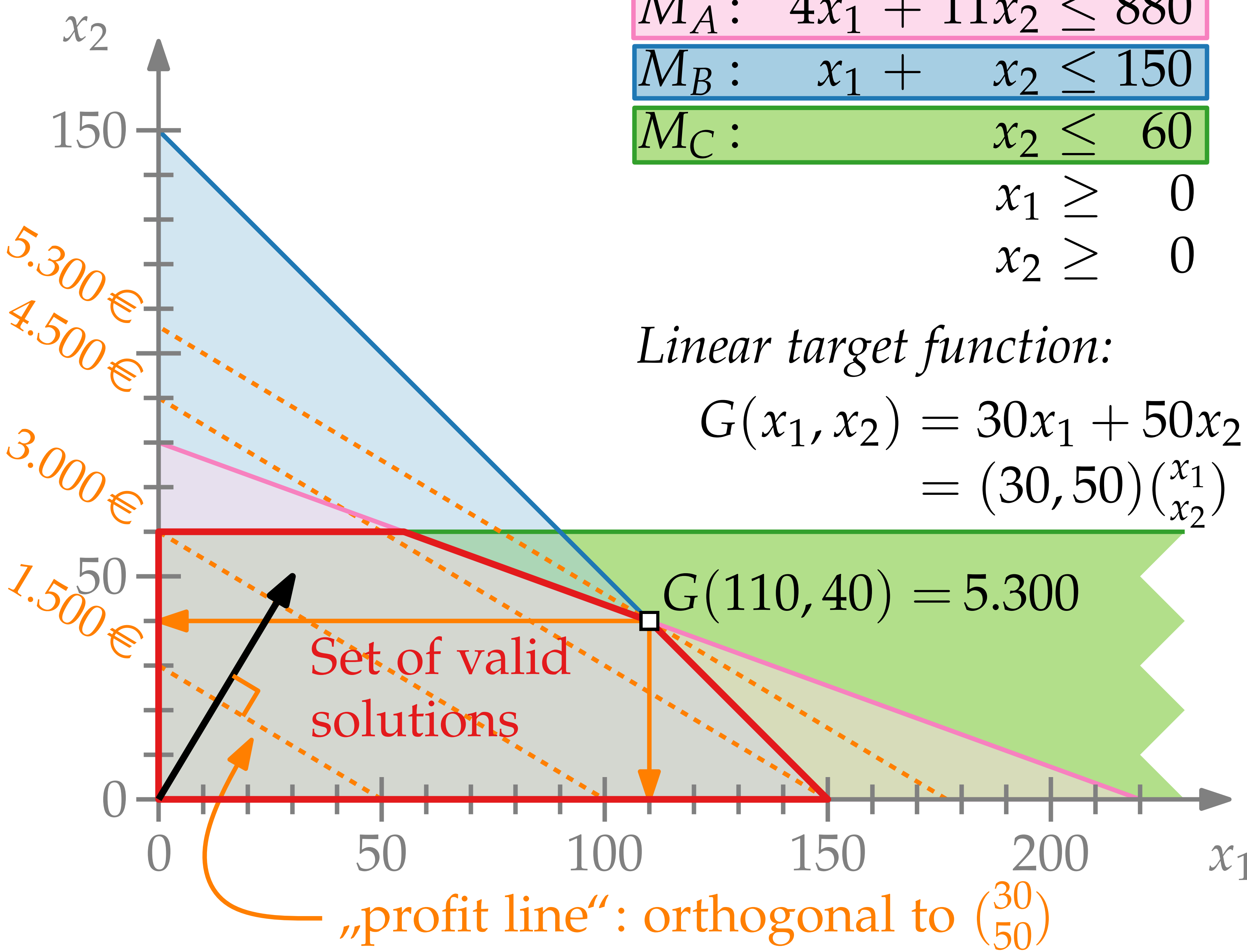
Solution

Linear constraints:

$$
\begin{aligned}
M_A:&\quad 4x_1 + 11x_2 \le 880 \\
M_B:&\quad x_1 + x_2 \le 150 \\
M_C:&\quad x_2 \le 60 \\
&\quad x_1 \ge 0 \\
&\quad x_2 \ge 0
\end{aligned}
\qquad
\begin{aligned}
Ax &\le b \\
x &\ge 0
\end{aligned}
$$

Linear target function:

$$
G(x_1, x_2) = 30x_1 + 50x_2 = (30, 50)\binom{x_1}{x_2}
$$



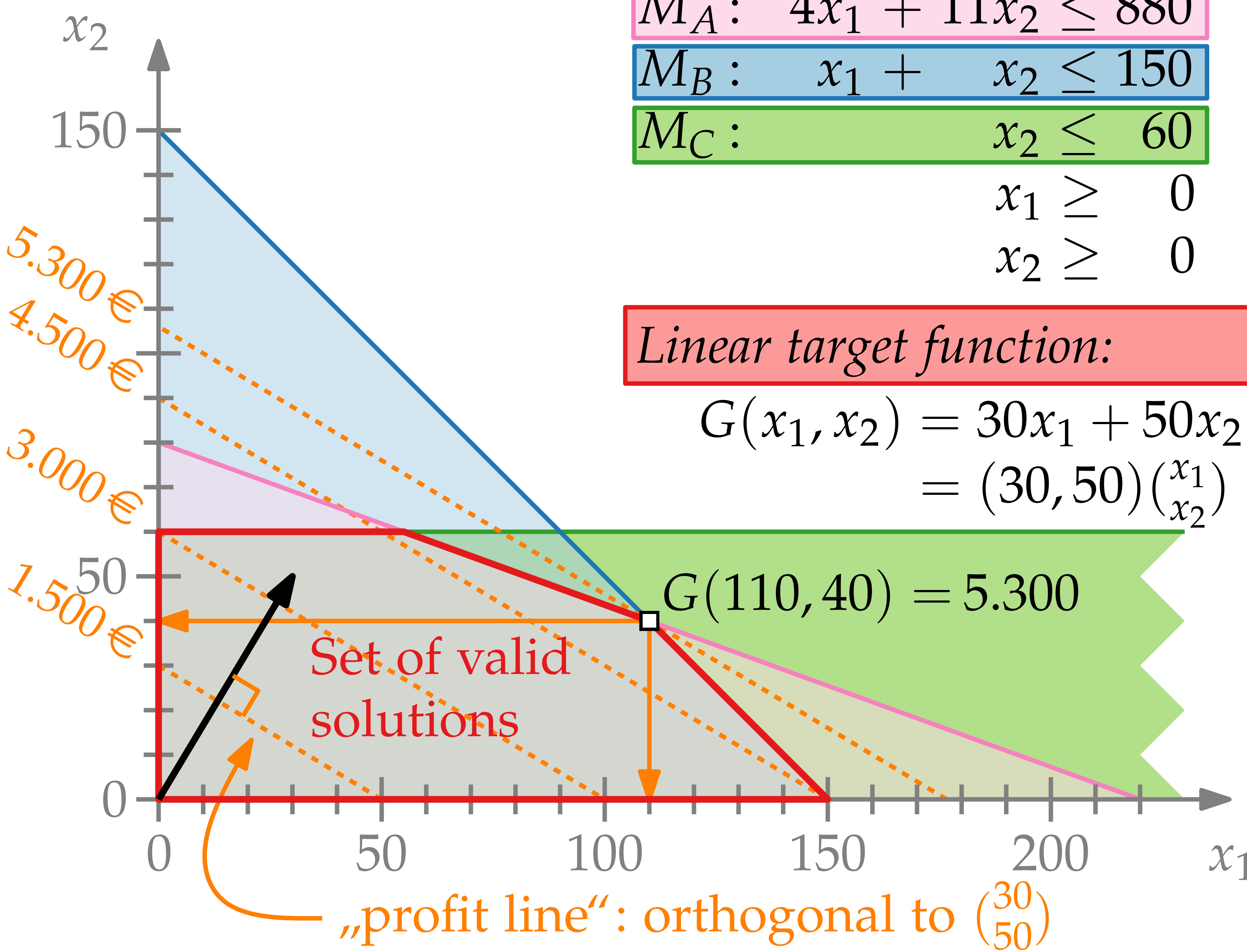
Solution

Linear constraints:

$$
\begin{aligned}
M_A: &\quad 4x_1 + 11x_2 \le 880 \\
M_B: &\quad x_1 + x_2 \le 150 \\
M_C: &\quad x_2 \le 60 \\
&\quad x_1 \ge 0 \\
&\quad x_2 \ge 0
\end{aligned}
\qquad
\begin{aligned}
Ax \le b \\
x \ge 0
\end{aligned}
$$

Linear target function:

$$
\begin{aligned}
G(x_1, x_2) &= 30x_1 + 50x_2 \\
&= (30, 50)\binom{x_1}{x_2}
\end{aligned}
$$



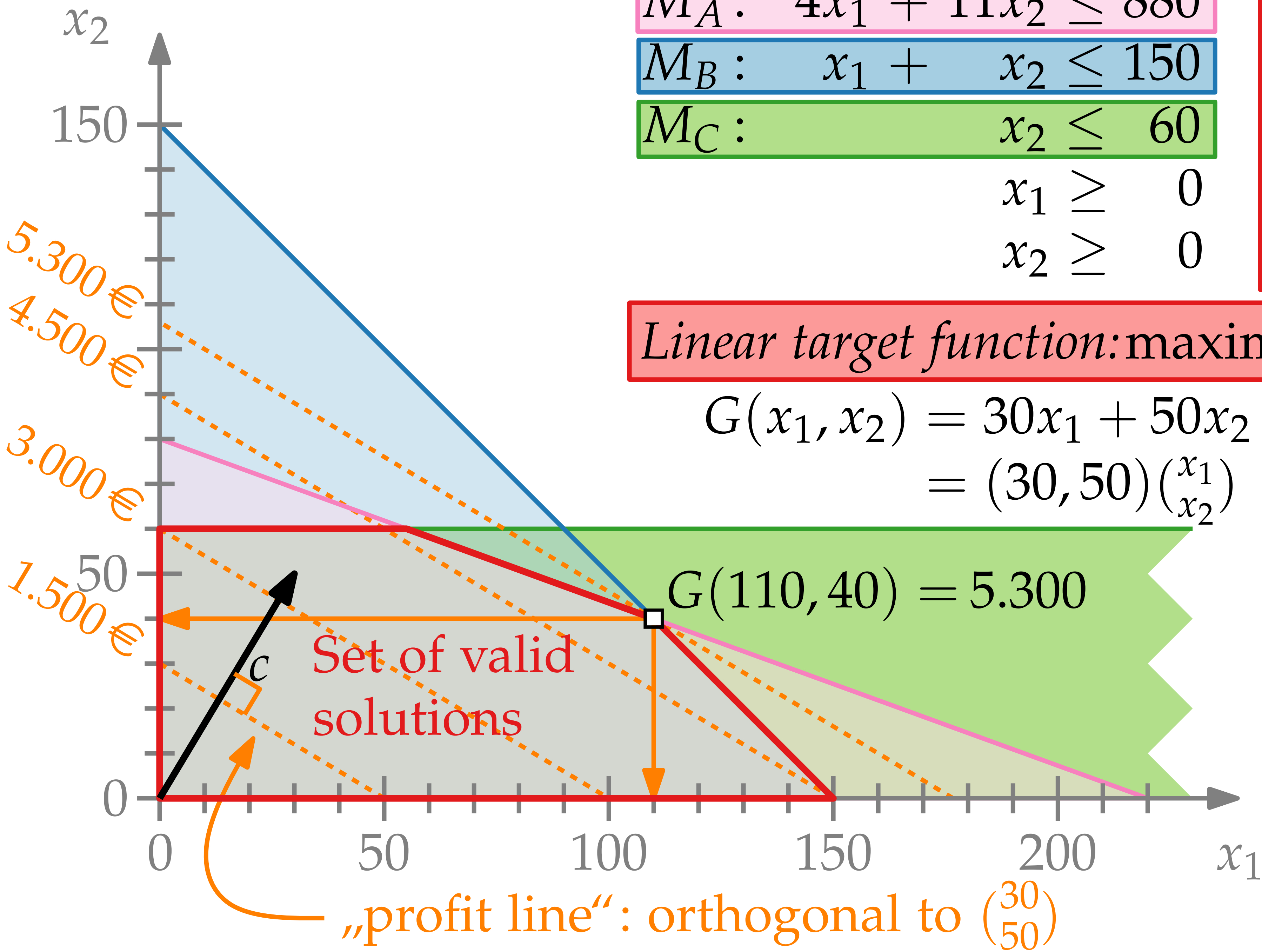
Solution

Linear constraints:

$M_A:$	$4x_1 + 11x_2 \leq 880$	
$M_B:$	$x_1 + x_2 \leq 150$	$Ax \leq b$
$M_C:$	$x_2 \leq 60$	
	$x_1 \geq 0$	$x \geq 0$
	$x_2 \geq 0$	

Linear target function: maximize $c^T x$

$$G(x_1, x_2) = 30x_1 + 50x_2 = (30, 50)\binom{x_1}{x_2}$$



Solution

Linear constraints:

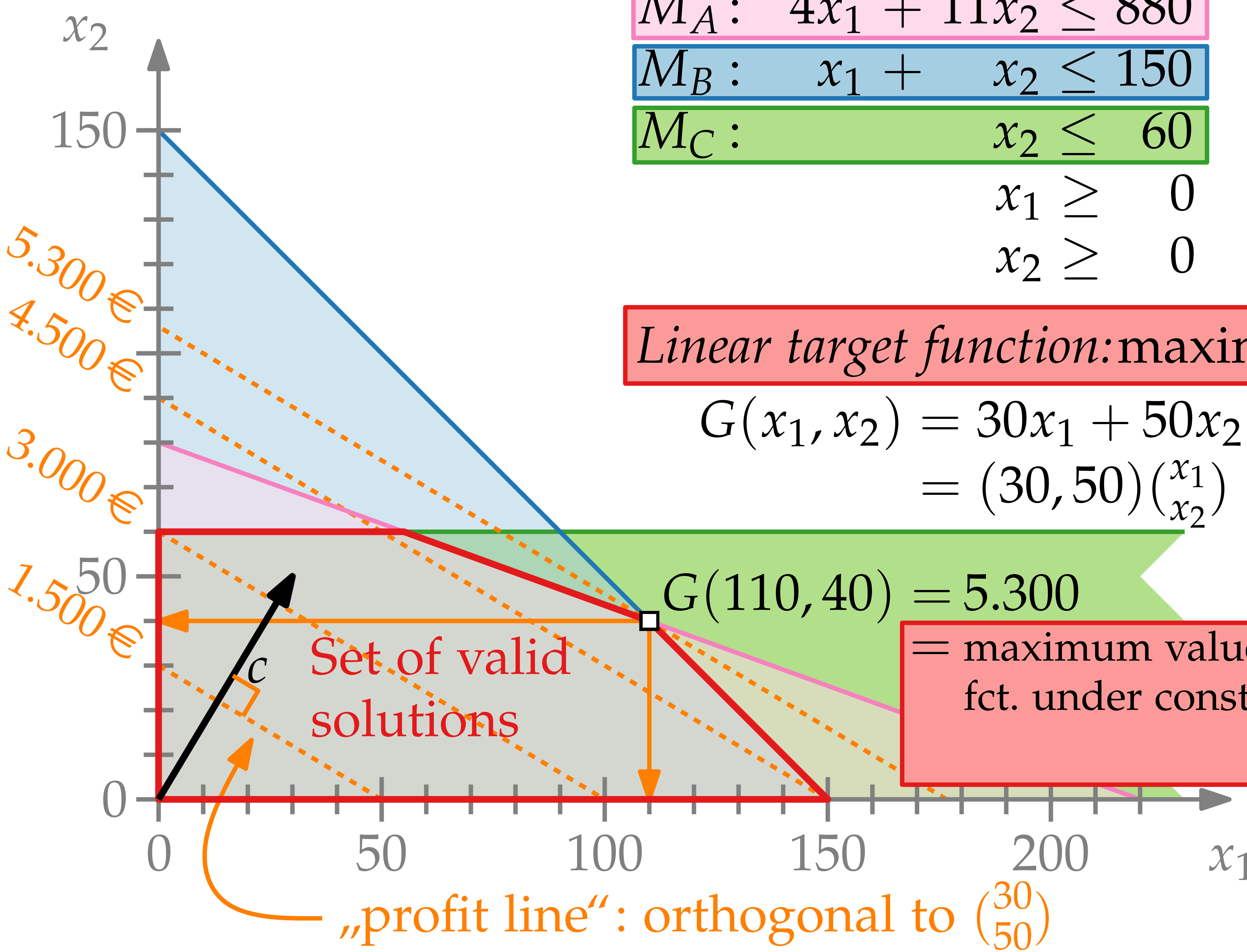
$$
\begin{aligned}
M_A: \quad 4x_1 + 11x_2 &\leq 880\\
M_B: \quad x_1 + x_2 &\leq 150\\
M_C: \quad x_2 &\leq 60\\
x_1 &\geq 0\\
x_2 &\geq 0
\end{aligned}
$$

$Ax \leq b$

$x \geq 0$

Linear target function: maximize $c^{\mathrm{T}}x$

$$
\begin{aligned}
G(x_1, x_2) &= 30x_1 + 50x_2\\
&= (30, 50)\binom{x_1}{x_2}
\end{aligned}
$$



$G(110, 40) = 5.300$

= maximum value of target fct. under constraints.

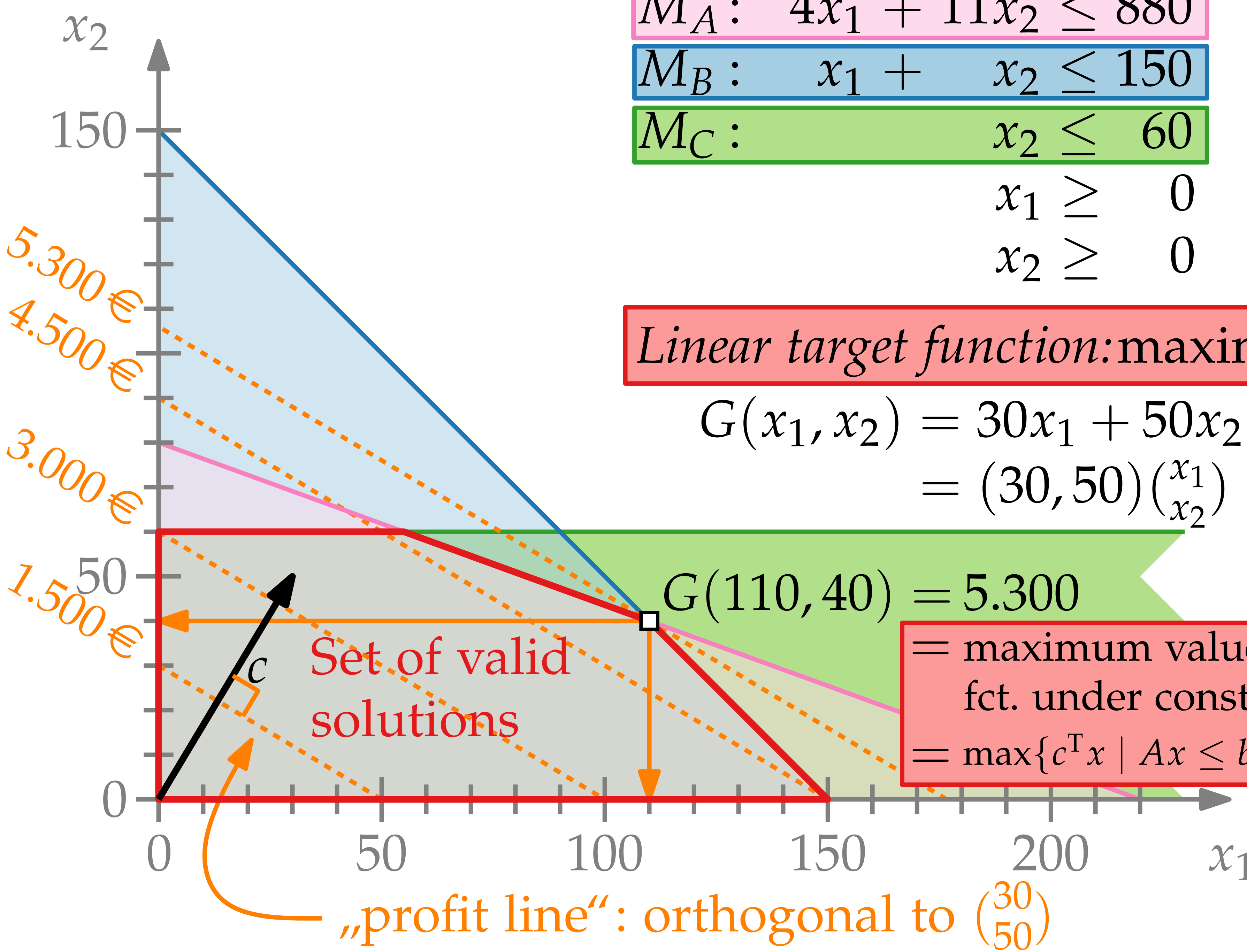
Solution

Linear constraints:

$M_A:$	$4x_1 + 11x_2 \leq 880$	
$M_B:$	$x_1 + x_2 \leq 150$	$Ax \leq b$
$M_C:$	$x_2 \leq 60$	
	$x_1 \geq 0$	
	$x_2 \geq 0$	$x \geq 0$

Linear target function: maximize $c^{\mathrm{T}} x$

$$G(x_1, x_2) = 30x_1 + 50x_2 = (30, 50)\binom{x_1}{x_2}$$



$= $ maximum value of target fct. under constraints.

$= \max\{c^{\mathrm{T}} x \mid Ax \leq b, x \geq 0\}$

Computational Geometry

Lecture 4:
Linear Programming
or
Profit Maximization

Part II:
A First Approach

Philipp Kindermann

Winter Semester 2020

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

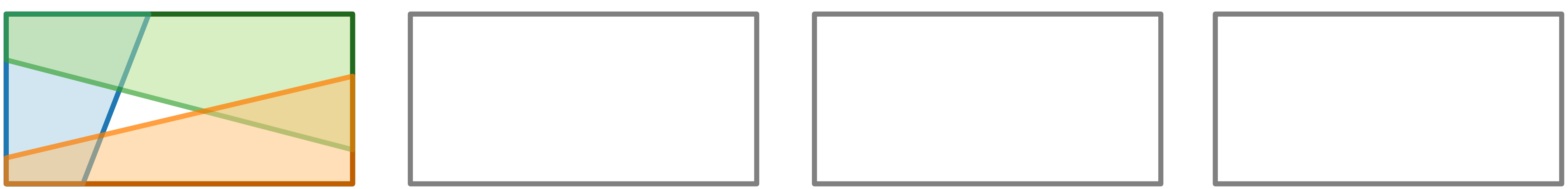
Many algorithms known, e.g.:

- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

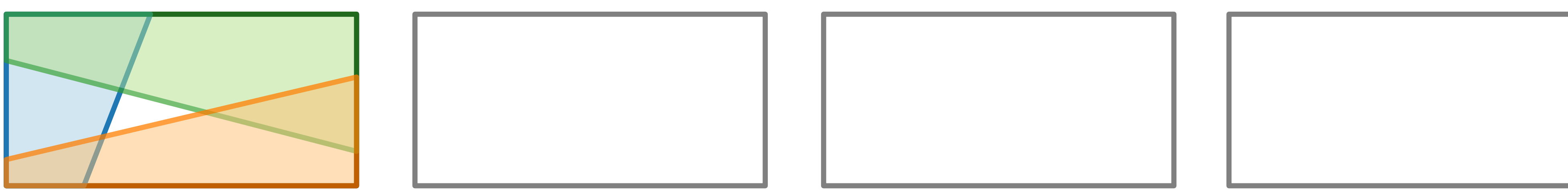
– Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]



$\bigcap H = \emptyset$

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

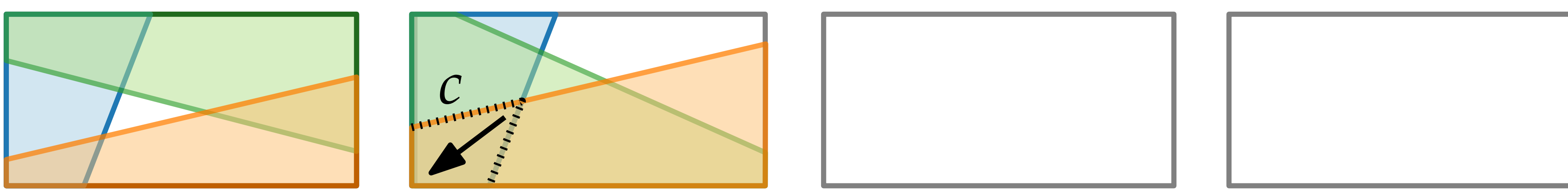
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where *n and d* are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]



$\bigcap H = \emptyset$

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

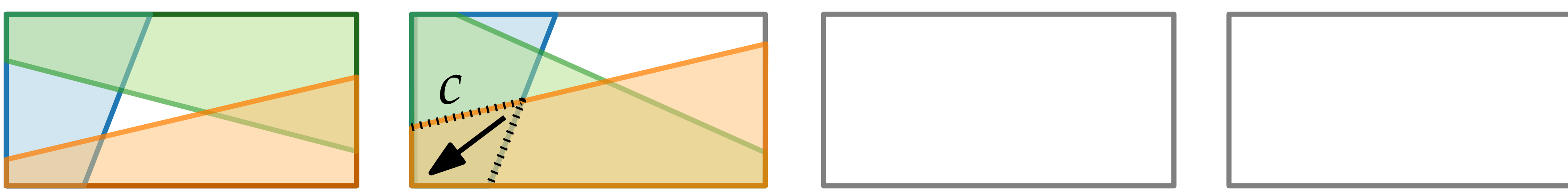
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where *n and d* are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]



$\bigcap H = \emptyset$ $\bigcap H$ unbnd. in dir. c

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

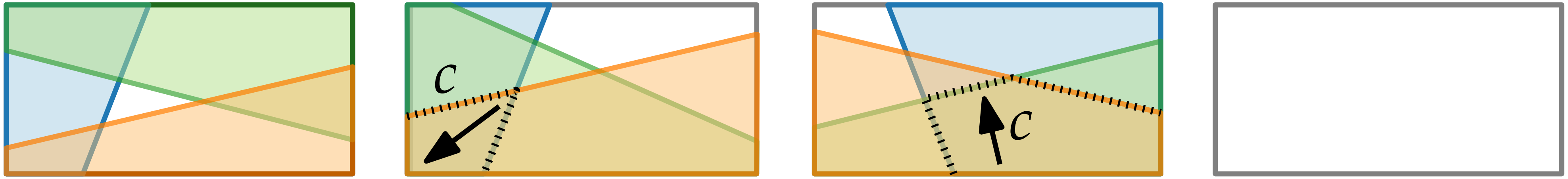
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]



$\bigcap H = \emptyset$ $\bigcap H$ unbnd. in dir. c

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

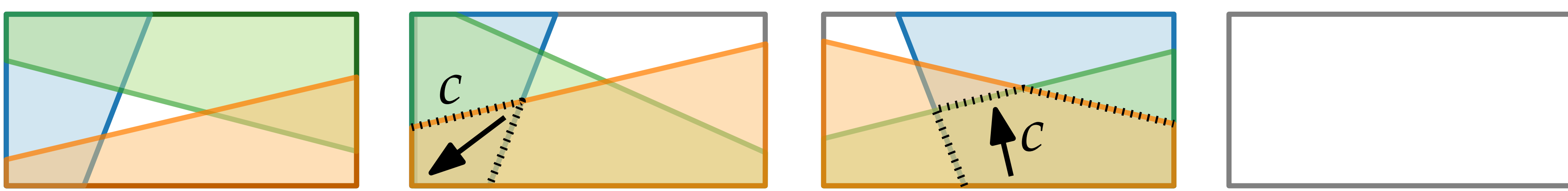
Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]

$\bigcap H$ bounded.



$\bigcap H = \emptyset$ $\bigcap H$ unbnd. in dir. c

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

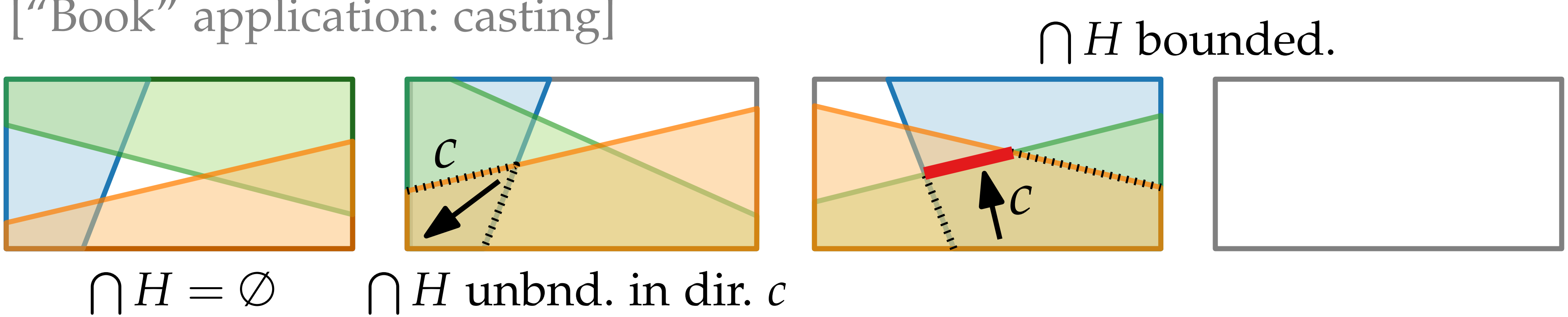
– Inner-point method [Karmakar' 84]

Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

[“Book” application: casting]



Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

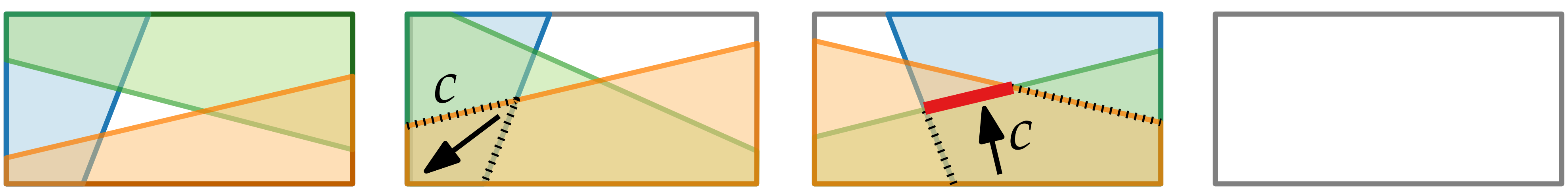
Good for instances where *n and d* are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]

$\bigcap H$ bounded.



$\bigcap H = \emptyset$ $\bigcap H$ unbnd. in dir. c set of optima: segment

Definition and Known Algorithms

Given a set H of n halfspaces in $\mathbb{R}^d$ and a direction c, find a point $x \in \bigcap H$ such that cx is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig '47]

– Ellipsoid method [Khatchiyan '79]

– Inner-point method [Karmakar' 84]

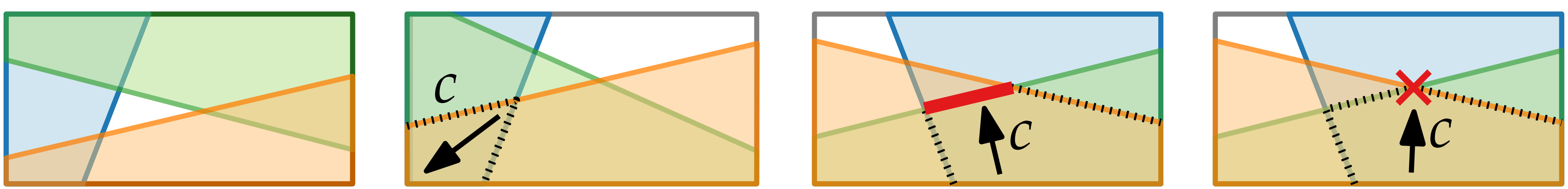
Good for instances where n *and* d are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

["Book" application: casting]

$\bigcap H$ bounded.



$\bigcap H = \emptyset$ $\bigcap H$ unbnd. in dir. c set of optima: segment vs. point

First Approach

First Approach

- compute $\bigcap H$ explicitly

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

IntersectHalfplanes(H)

return C

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

```
IntersectHalfplanes(H)
  if |H| = 1 then
  |   C ← h, where {h} = H
  else
  |
  |
  |
  |
  └
  return C
```

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$

return C

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

```
IntersectHalfplanes(H)
  if |H| = 1 then
  |   C ← h, where {h} = H
  else
  |   split H into sets H_1 and H_2 with |H_1|, |H_2| ≈ |H|/2
  |   C_1 ← IntersectHalfplanes(H_1)
  |
  |
  └
  return C
```

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

```
IntersectHalfplanes(H)
  if |H| = 1 then
  |   C ← h, where {h} = H
  else
  |   split H into sets H_1 and H_2 with |H_1|, |H_2| ≈ |H|/2
  |   C_1 ← IntersectHalfplanes(H_1)
  |   C_2 ← IntersectHalfplanes(H_2)
  |
  return C
```

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Running time:

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

IntersectHalfplanes(H)

if $|H| = 1$ **then**
  $C \leftarrow h$, where $\{h\} = H$
else
  split H into sets H_1 and H_2 with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Running time: $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

First Approach

- compute $\bigcap H$ explicitly
- walk along $\partial\left(\bigcap H\right)$ to find a vertex x with cx maximum

```
IntersectHalfplanes(H)
  if |H| = 1 then
  |   C ← h, where {h} = H
  else
  |   split H into sets H_1 and H_2 with |H_1|, |H_2| ≈ |H|/2
  |   C_1 ← IntersectHalfplanes(H_1)
  |   C_2 ← IntersectHalfplanes(H_2)
  |_  C ← IntersectConvexRegions(C_1, C_2)
  return C
```

How??

Running time: $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

Computational Geometry

Lecture 4:
Linear Programming
or
Profit Maximization

Part III:
Intersecting Convex Regions

Philipp Kindermann

Winter Semester 2020

Intersecting Convex Regions

Any ideas?

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) =$

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

here: $I \leq$

Intersecting Convex Regions

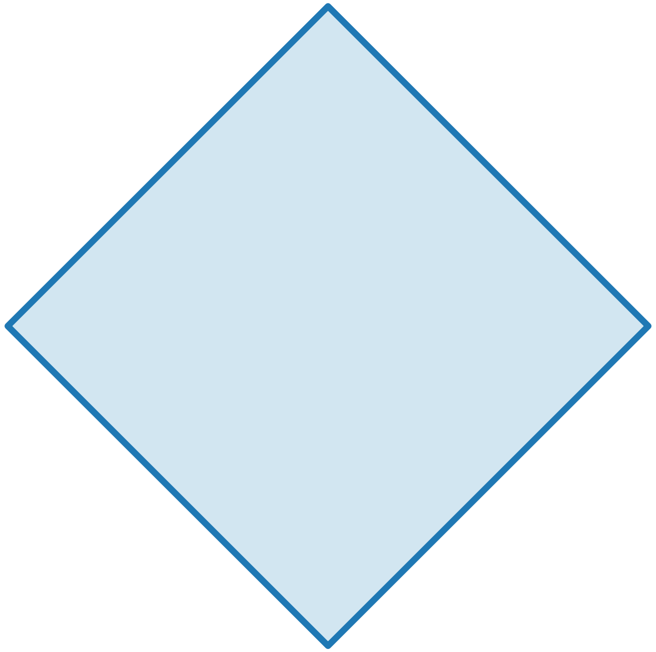
Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

here: $I \leq$



Intersecting Convex Regions

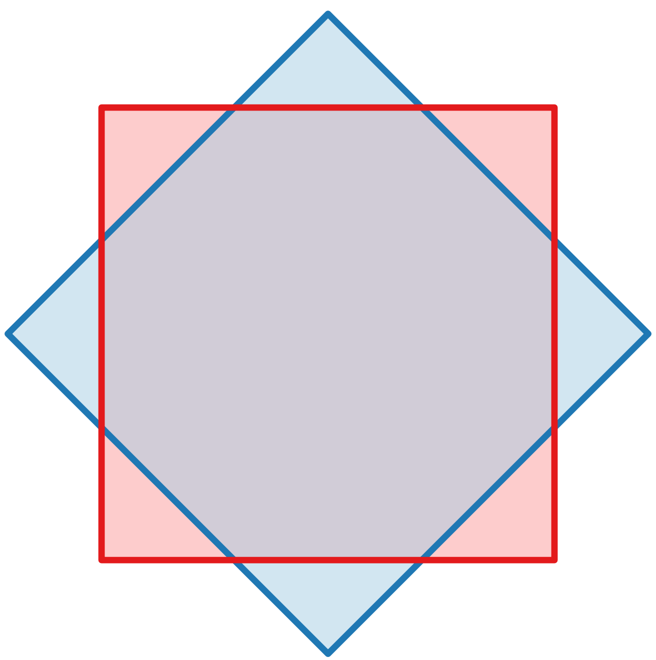
Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

here: $I \leq$



Intersecting Convex Regions

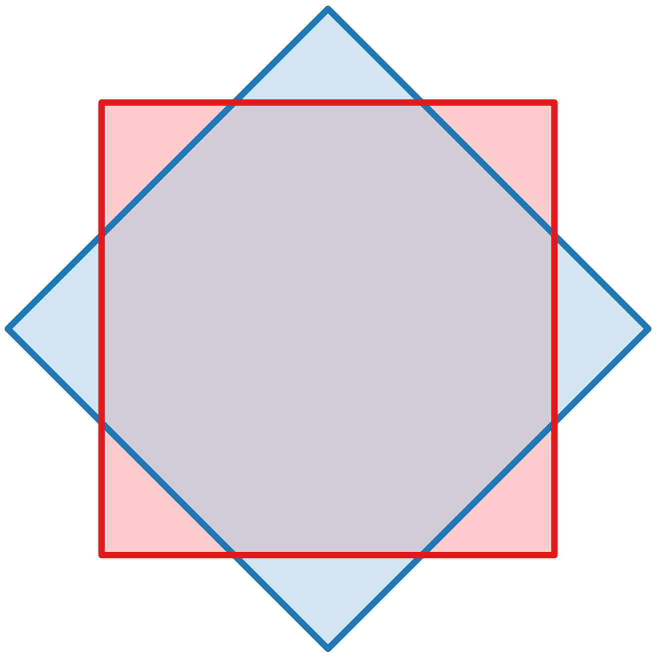
Any ideas?

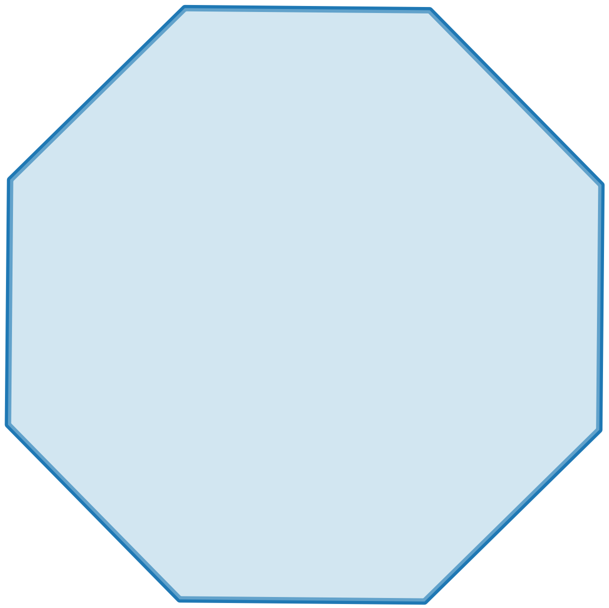
Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I) \log n)$,

where $I = \#$ intersection points.

here: $I \leq$





Intersecting Convex Regions

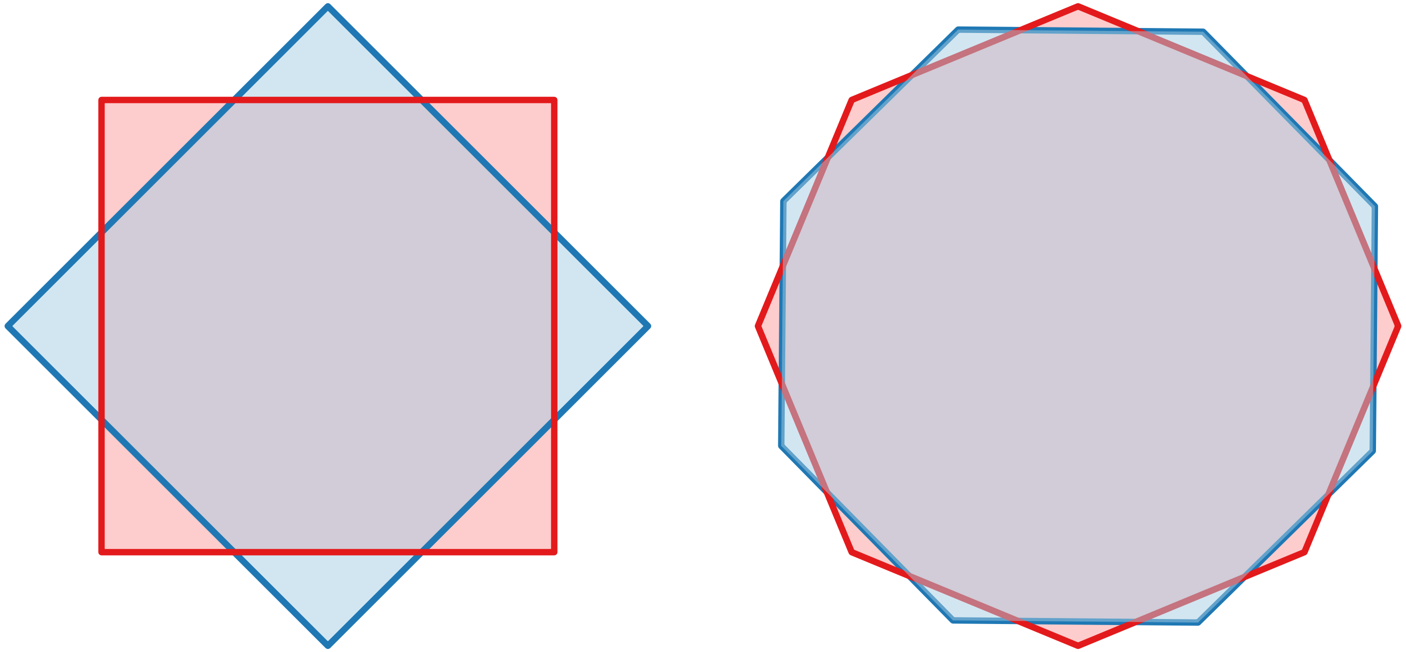
Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

here: $I \leq$



Intersecting Convex Regions

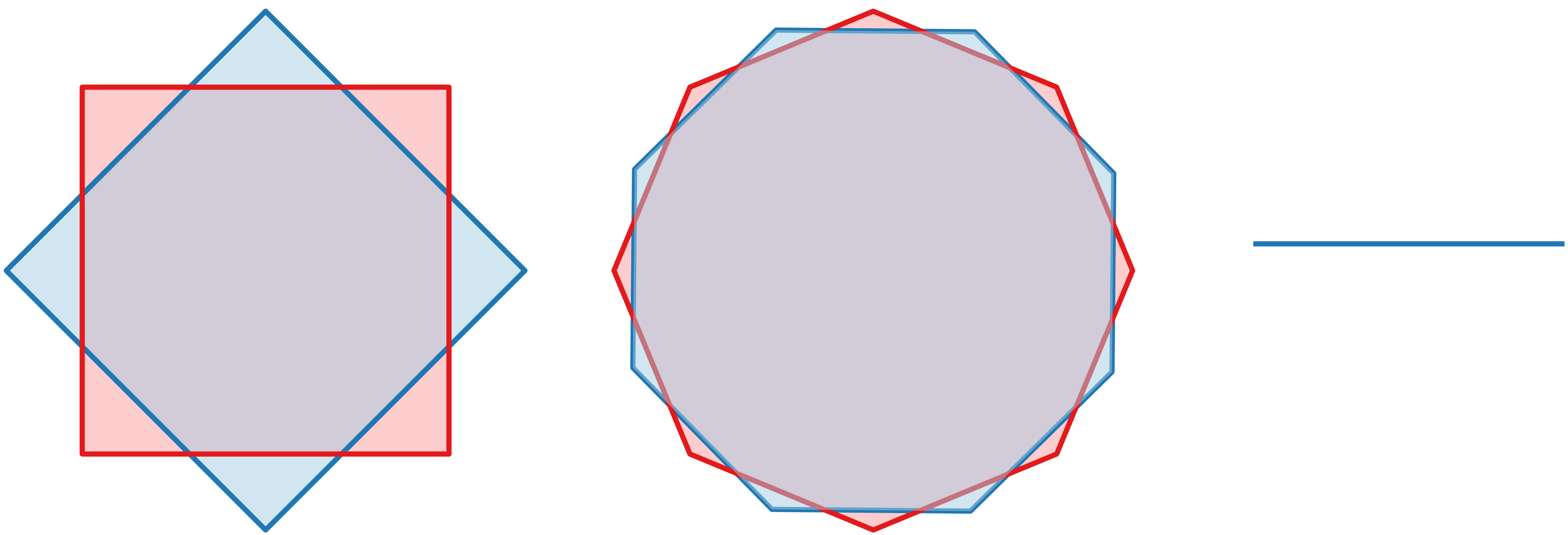
Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

here: $I \leq$



Intersecting Convex Regions

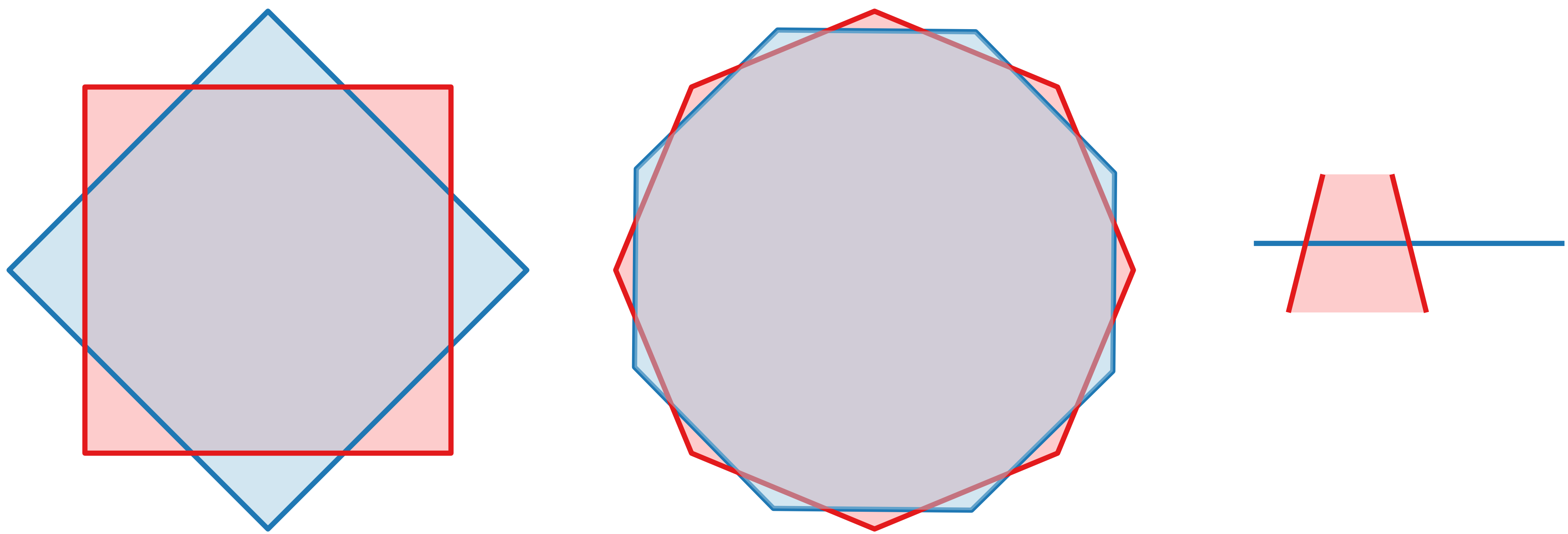
Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I) \log n)$,

where $I = \#$ intersection points.

here: $I \leq$



Intersecting Convex Regions

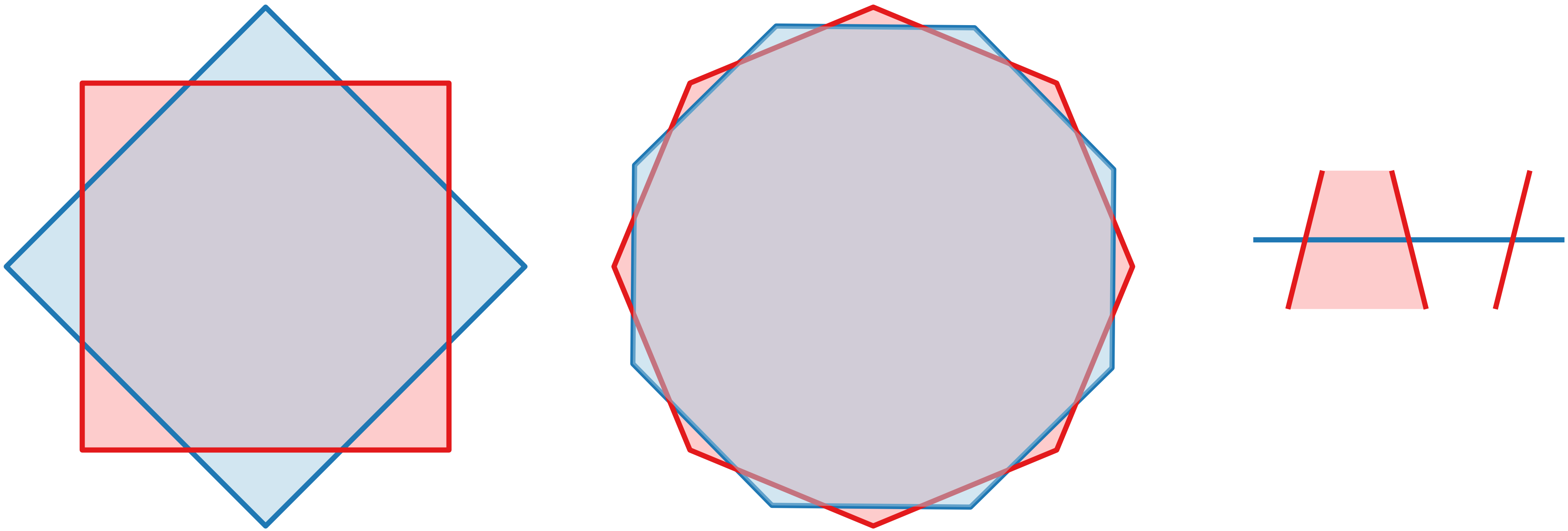
Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I) \log n)$,

where $I = \#$ intersection points.

here: $I \leq$



Intersecting Convex Regions

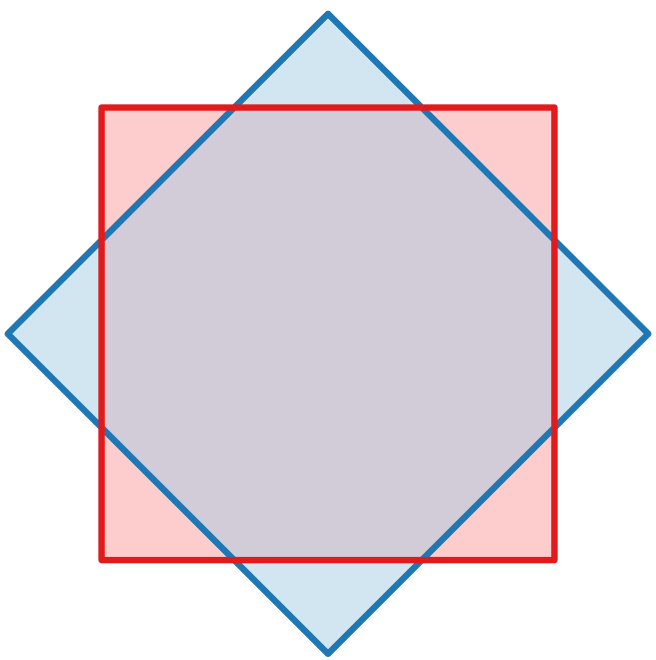
Any ideas?

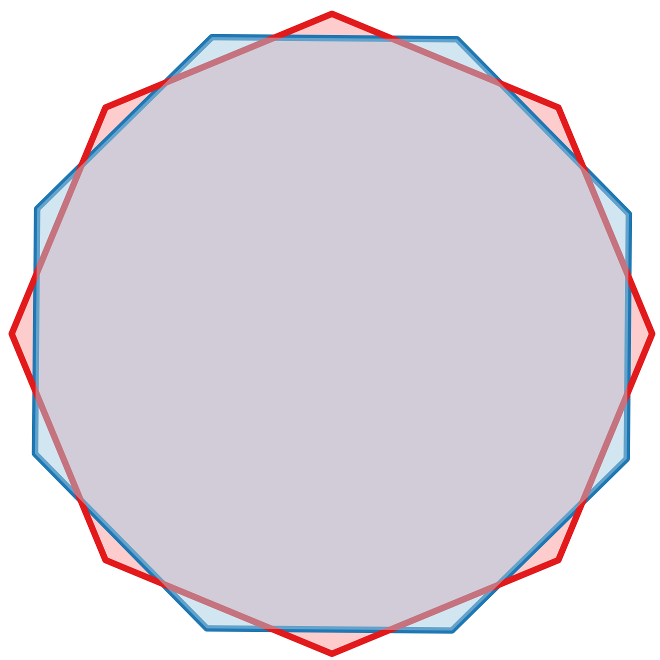
Use sweep-line alg. for map overlay (line-segment intersections)!

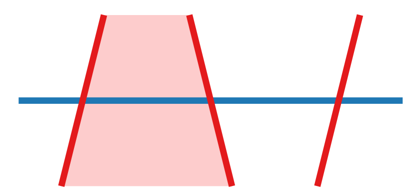
Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,

where $I = \#$ intersection points.

here: $I \le n$





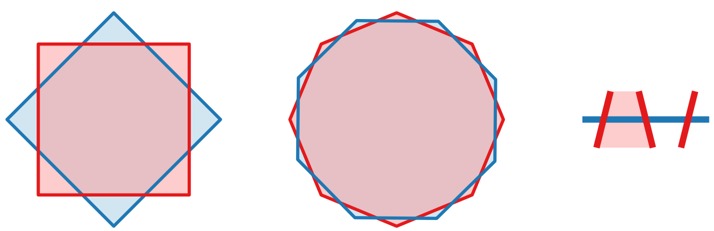


Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n+I)\log n)$,



where $I = \#$ intersection points.

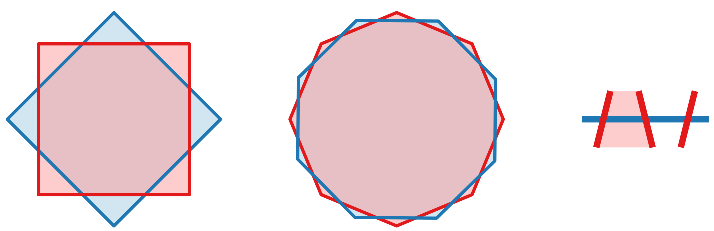
here: $I \leq n$

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,



where $I = \#$ intersection points.

here: $I \leq n$

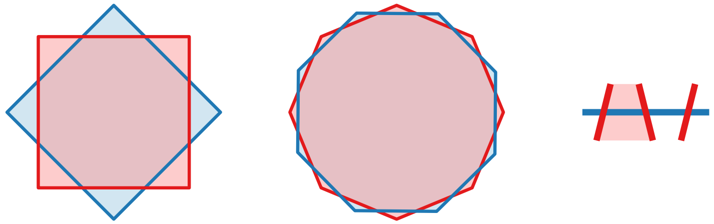
Running time $T_{\text{IH}}(n) =$

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\text{ICR}}(n) = O((n + I)\log n)$,



where $I = \#$ intersection points.

here: $I \leq n$

Running time $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

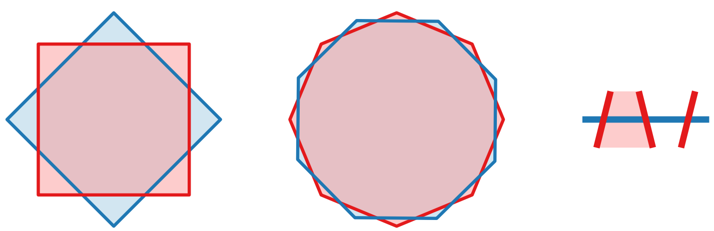
$\leq$

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I) \log n)$,



where $I = \#$ intersection points.

here: $I \leq n$

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

$\leq 2T_{\mathrm{IH}}(n/2) + O(n \log n)$

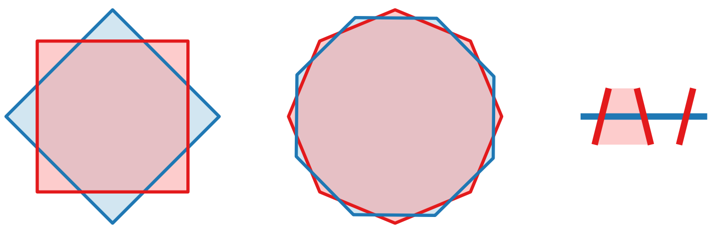
$\in$

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I)\log n)$,



where $I = \#$ intersection points.

here: $I \leq n$

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

$$\leq 2T_{\mathrm{IH}}(n/2) + O(n \log n)$$

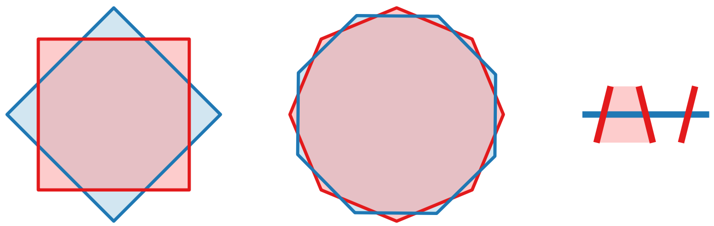
$$\in O(n \log^2 n)$$

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I)\log n)$,



where $I = \#$ intersection points.

here: $I \le n$

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

$$\le 2T_{\mathrm{IH}}(n/2) + O(n \log n)$$

$$\in O(n \log^2 n)$$

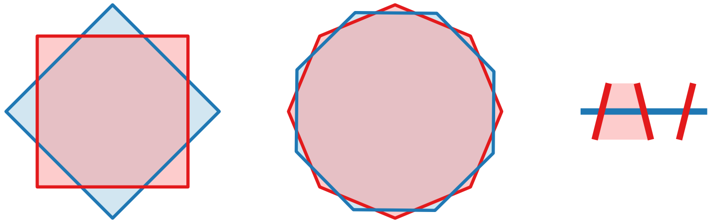
Better ideas?

Intersecting Convex Regions

Any ideas?

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{\mathrm{ICR}}(n) = O((n + I)\log n)$,



where $I = \#$ intersection points.

here: $I \leq n$

Running time $T_{\mathrm{IH}}(n) = 2T_{\mathrm{IH}}(n/2) + T_{\mathrm{ICR}}(n)$

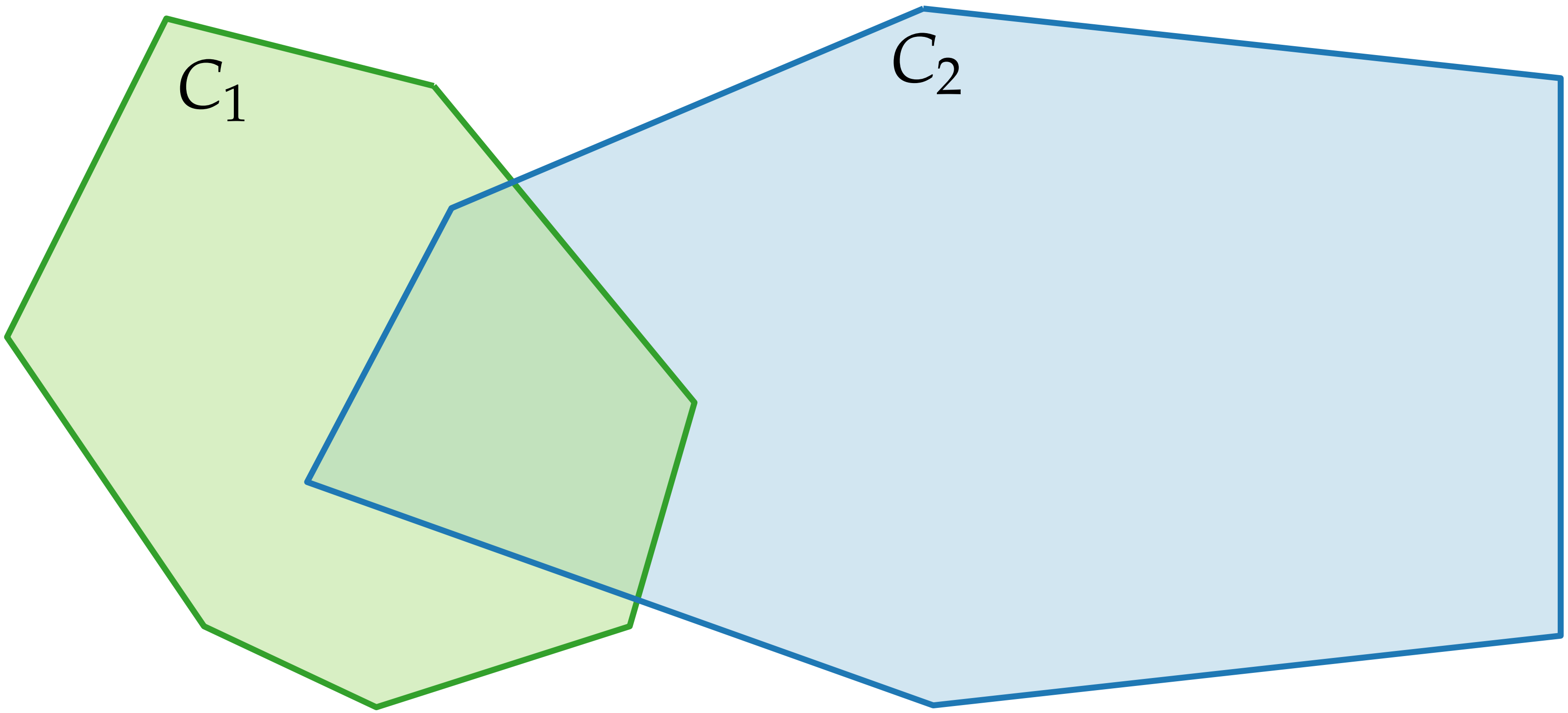
$$\leq 2T_{\mathrm{IH}}(n/2) + O(n \log n)$$

$$\in O(n \log^2 n)$$

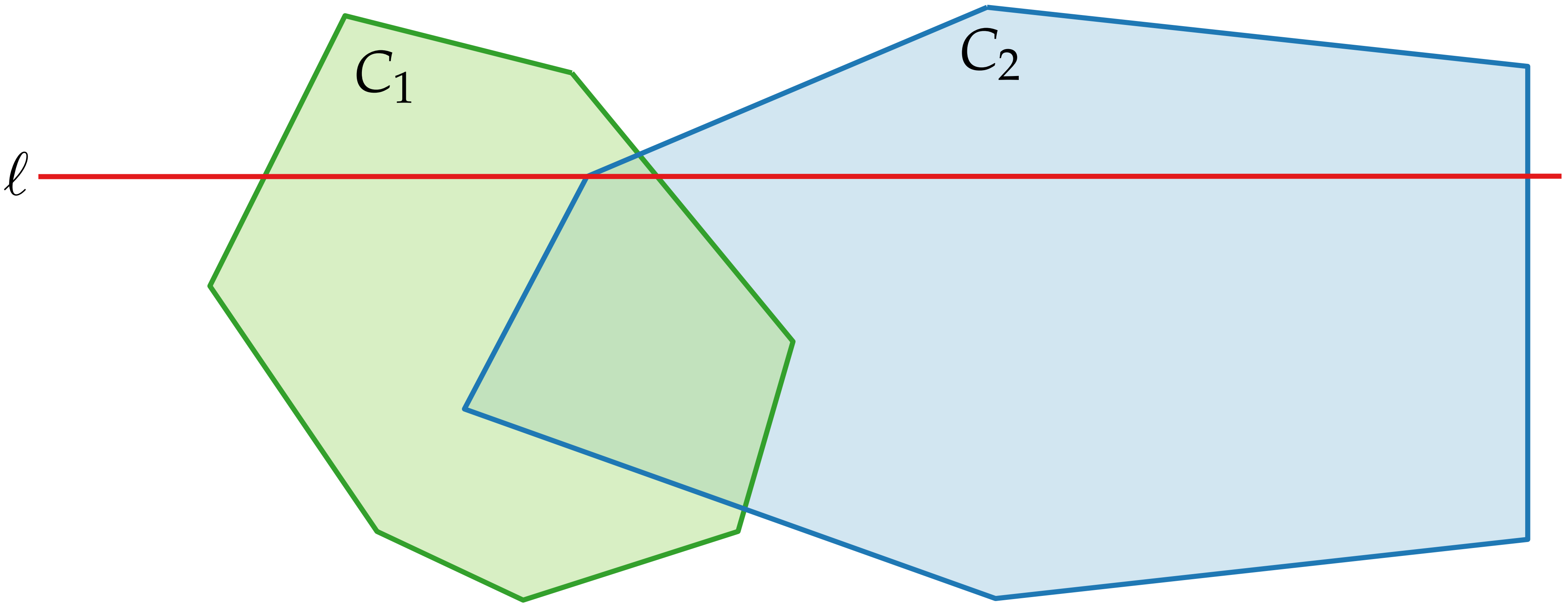
Better ideas?

Better analysis of the sweep-line for *convex* regions/polygons!

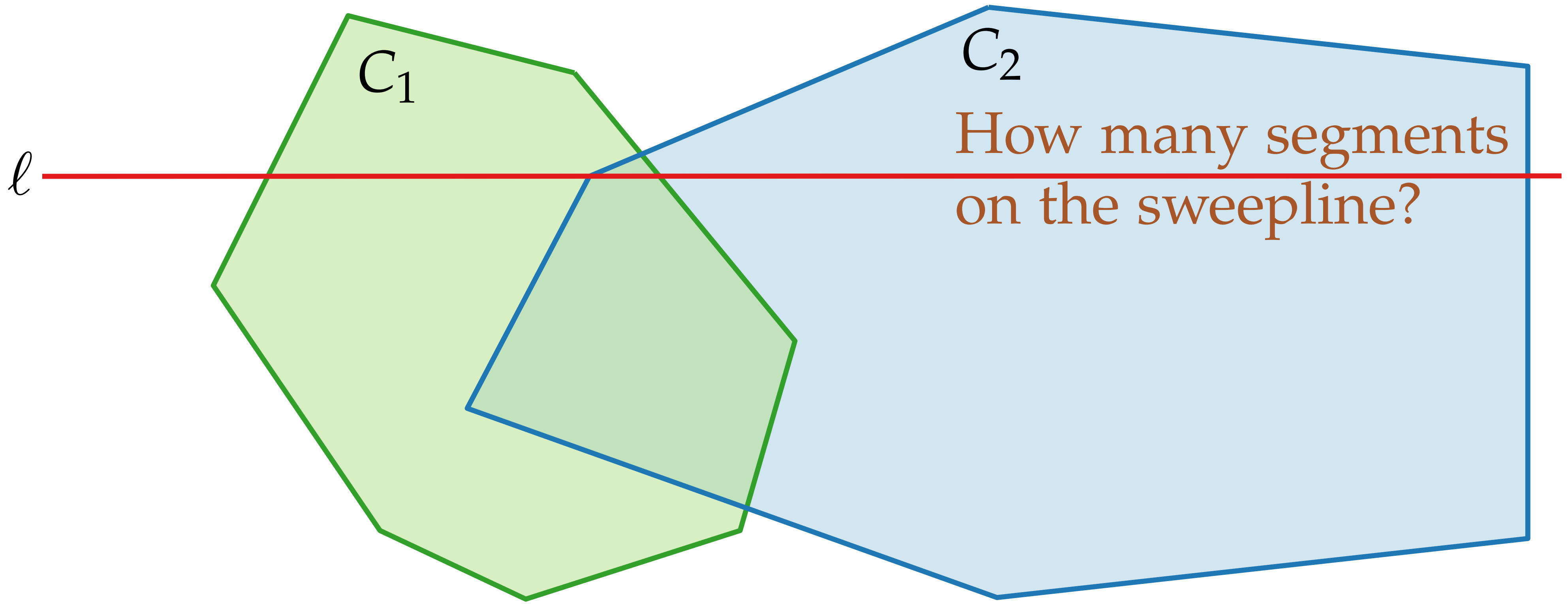
Intersecting Convex Regions Faster



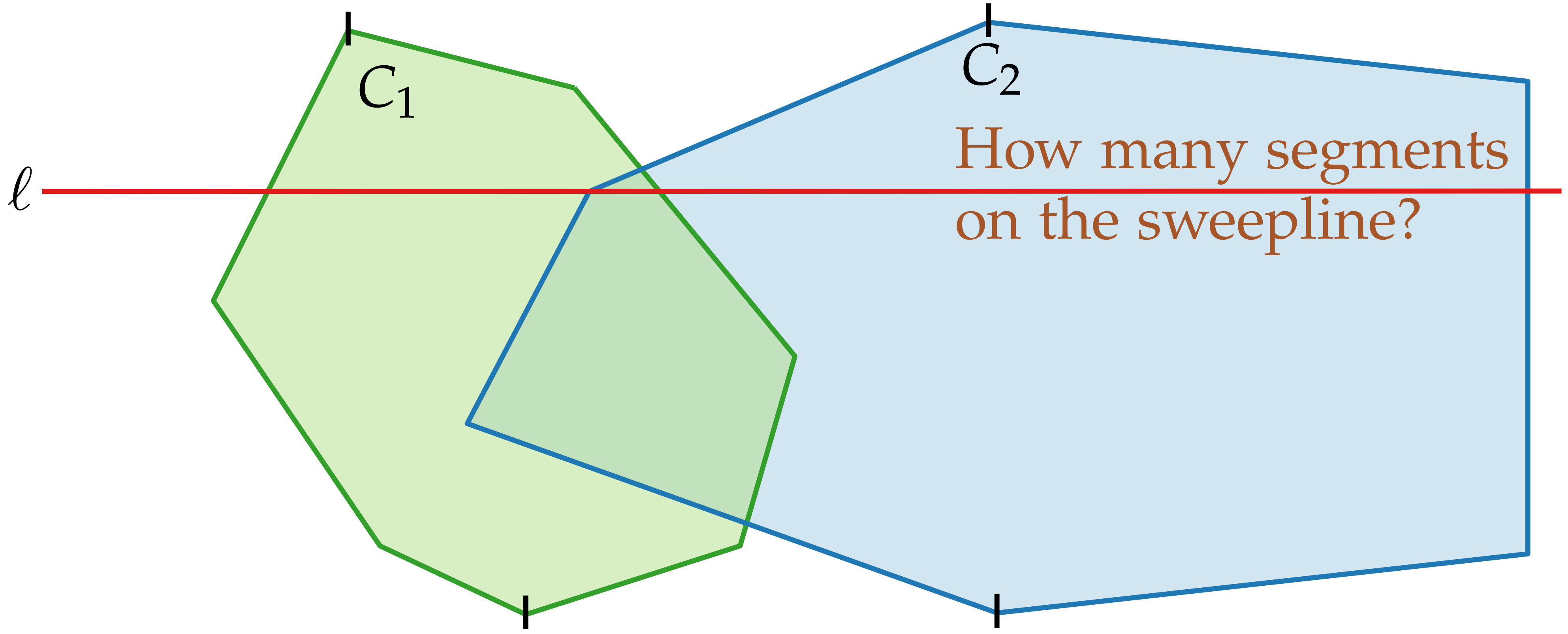
Intersecting Convex Regions Faster



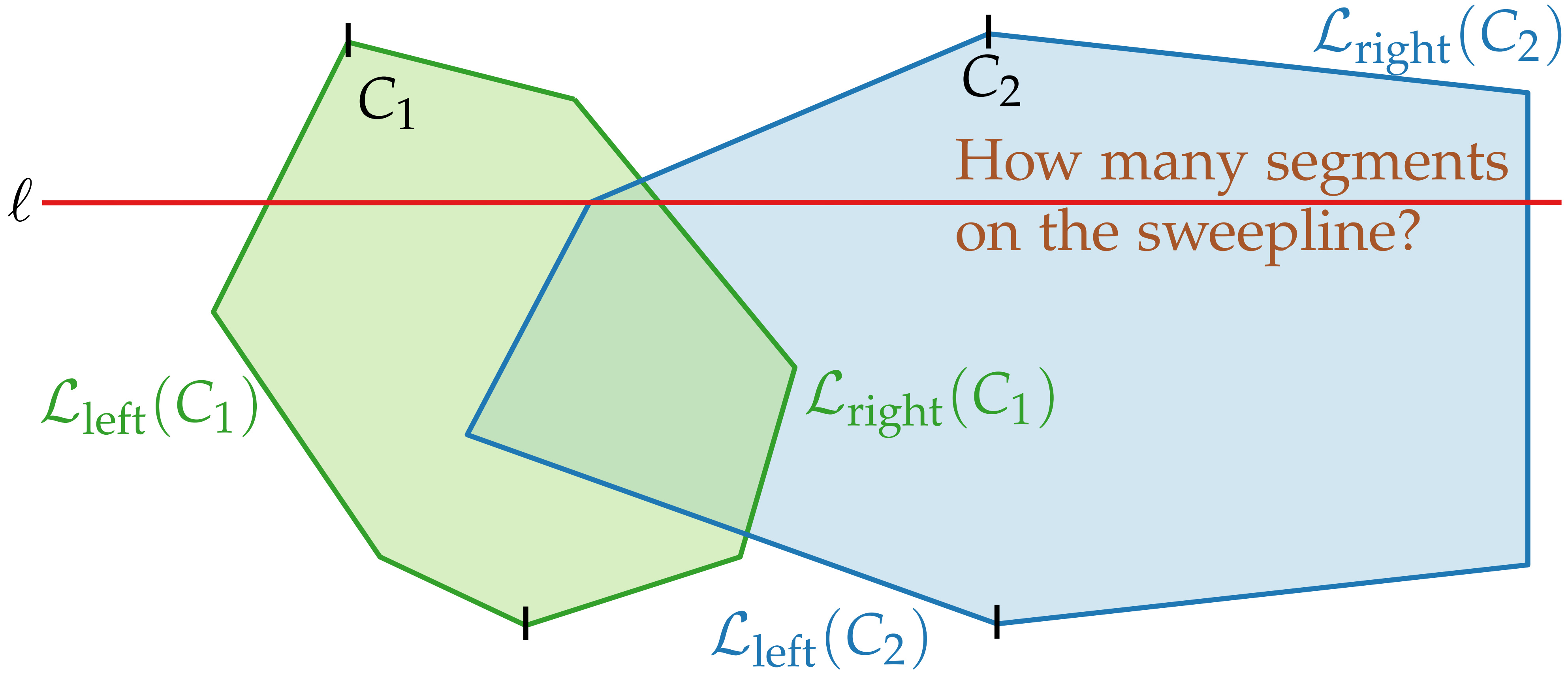
Intersecting Convex Regions Faster



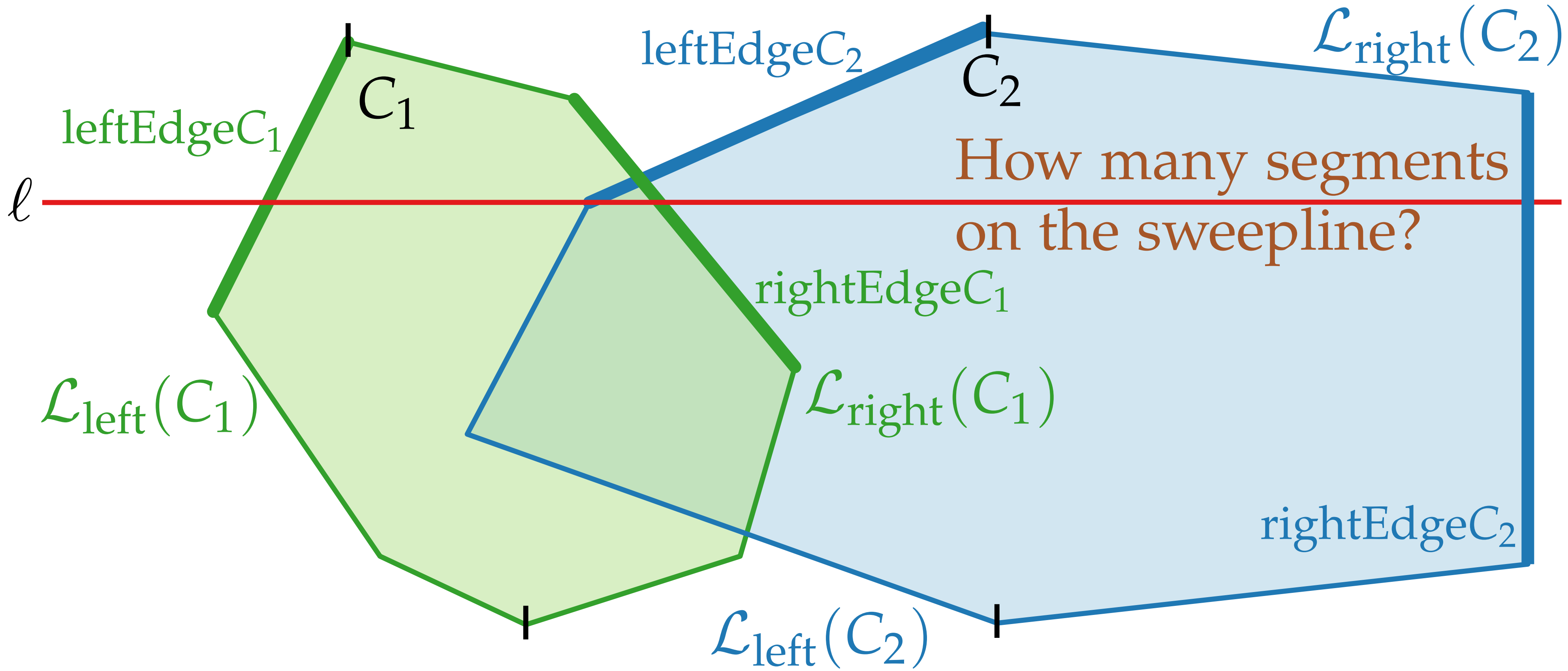
Intersecting Convex Regions Faster



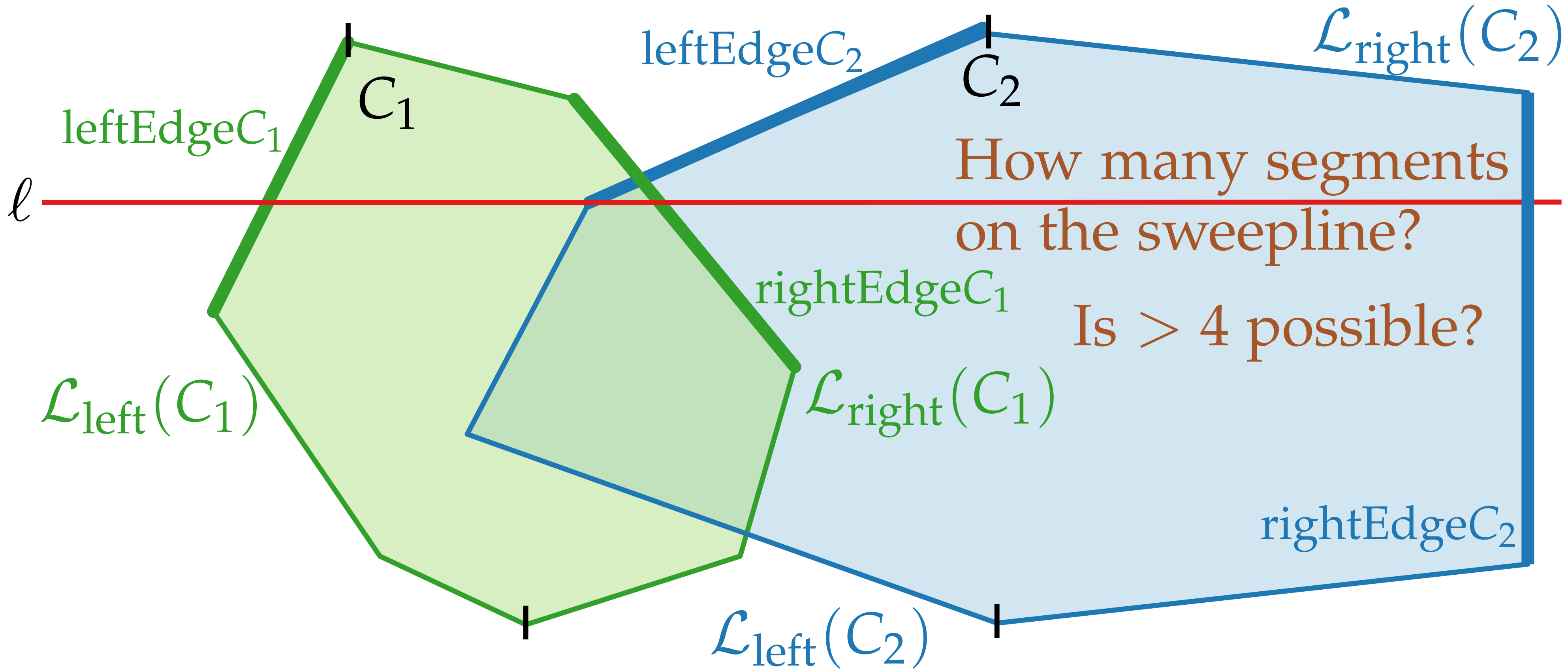
Intersecting Convex Regions Faster



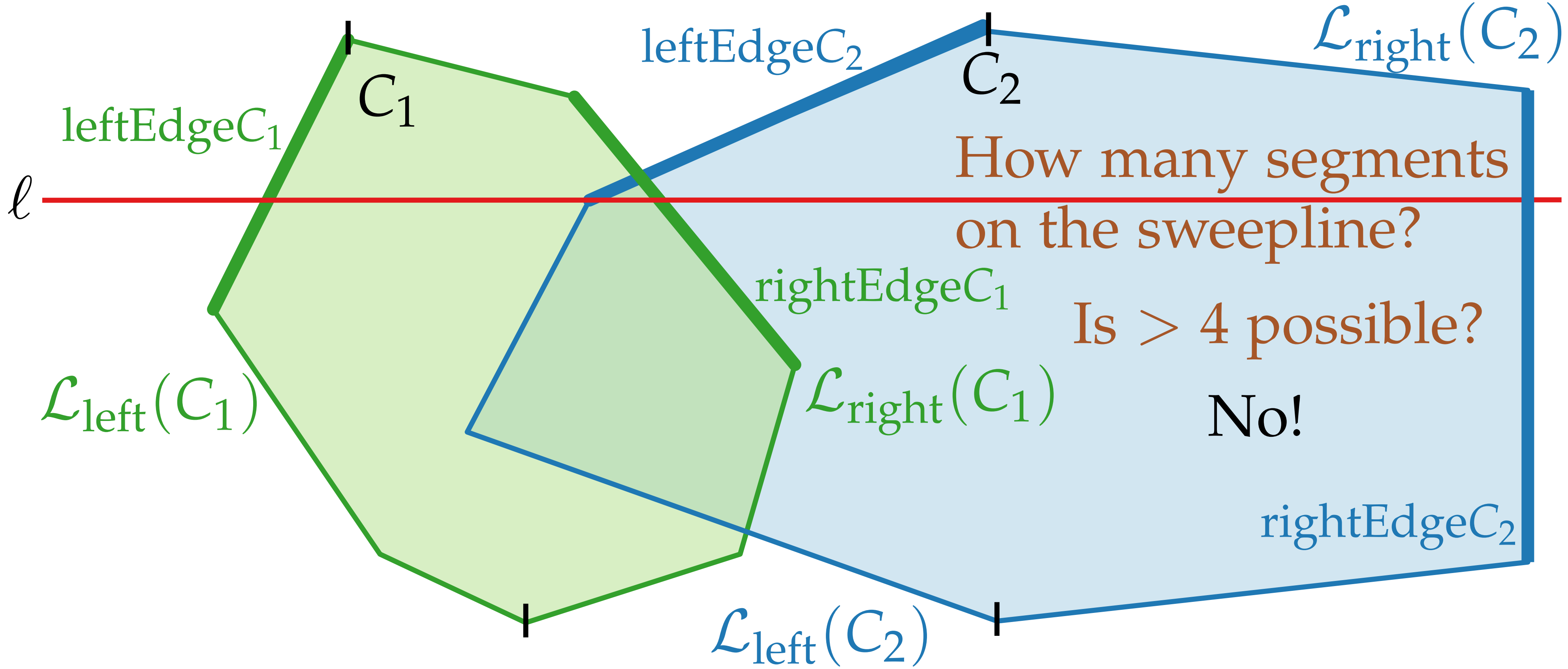
Intersecting Convex Regions Faster



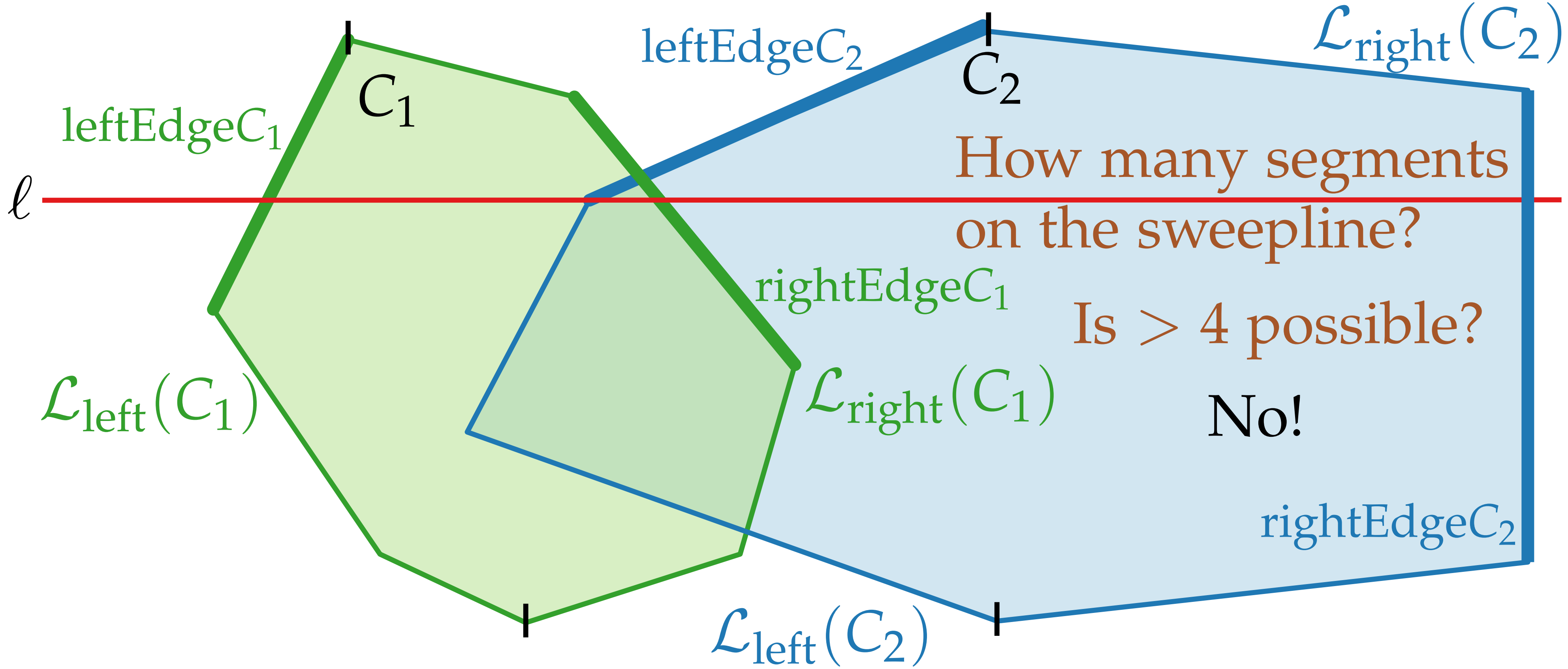
Intersecting Convex Regions Faster



Intersecting Convex Regions Faster

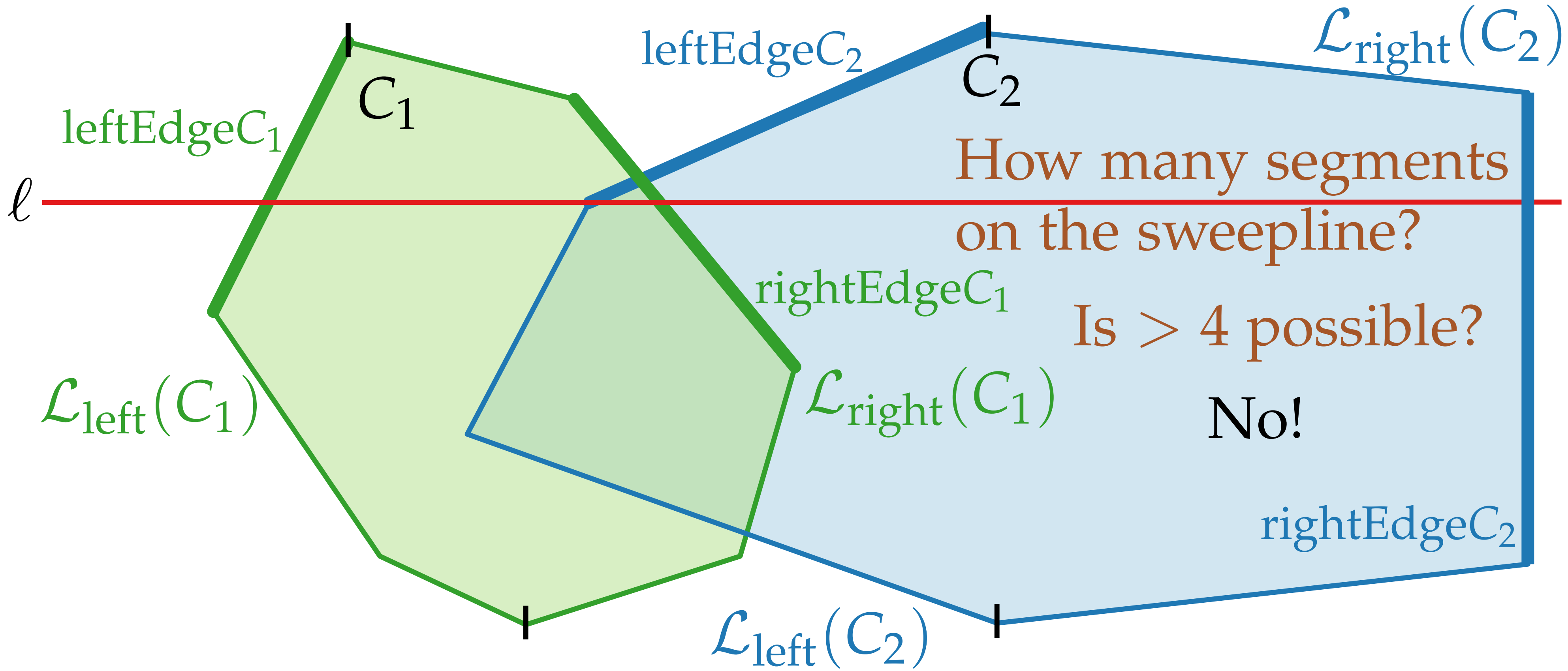


Intersecting Convex Regions Faster



Theorem. The intersection of two convex polygonal regions can be computed in linear time.

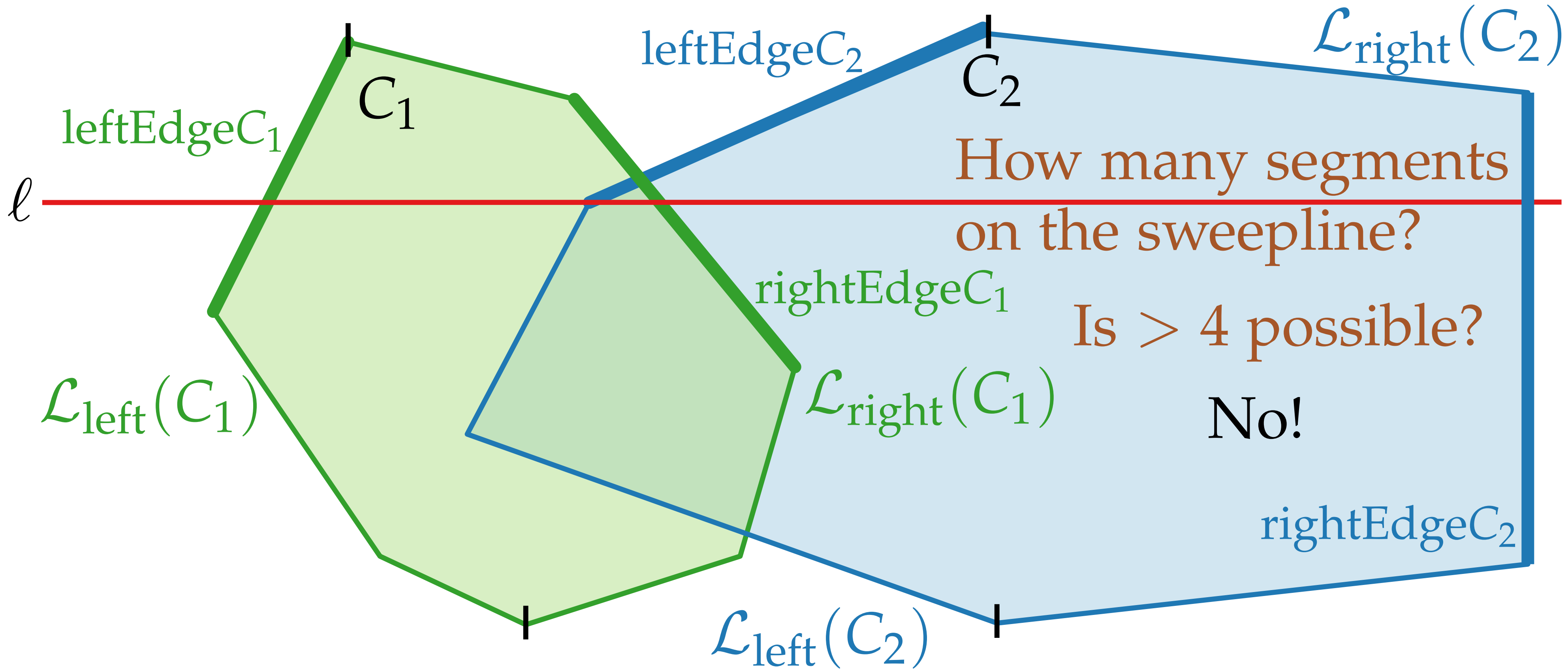
Intersecting Convex Regions Faster



Theorem. The intersection of two convex polygonal regions can be computed in linear time.

Corollary. The intersection of n half planes can be computed in $O(n \log n)$ time.

Intersecting Convex Regions Faster



Theorem. The intersection of two convex polygonal regions can be computed in linear time.

Corollary. The intersection of n half planes can be computed in $O(n \log n)$ time.

Can we do better?

Computational Geometry

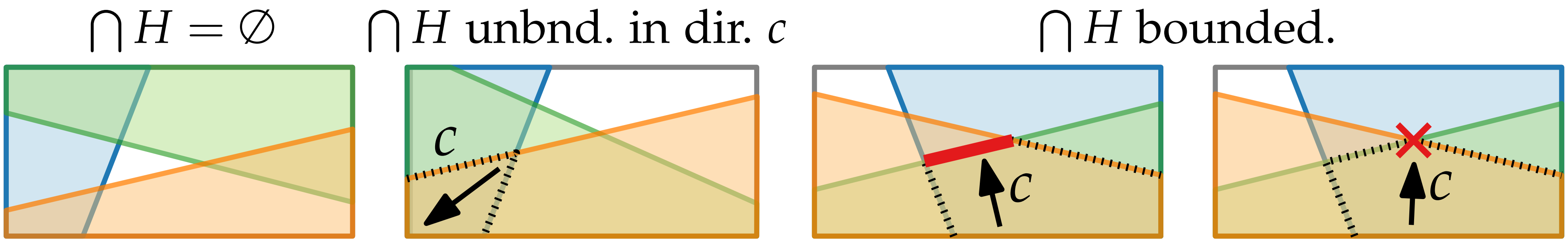
Lecture 4:
Linear Programming
or
Profit Maximization

Part IV:
Incremental Approach

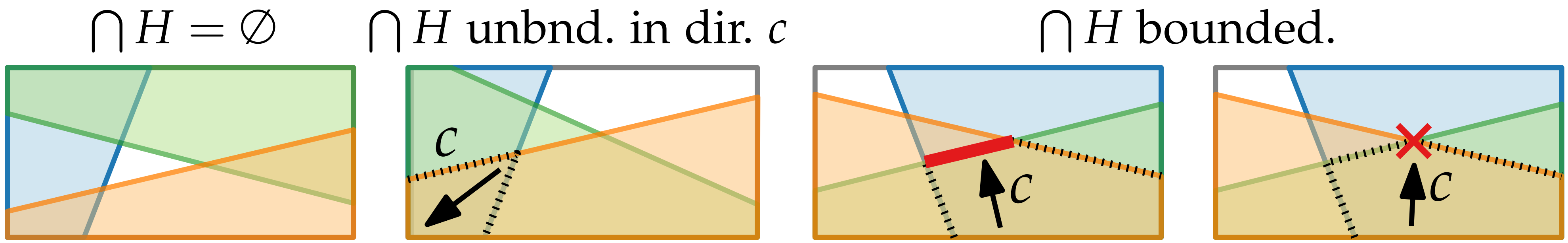
Philipp Kindermann

Winter Semester 2020

A Small Trick: Make Solution Unique

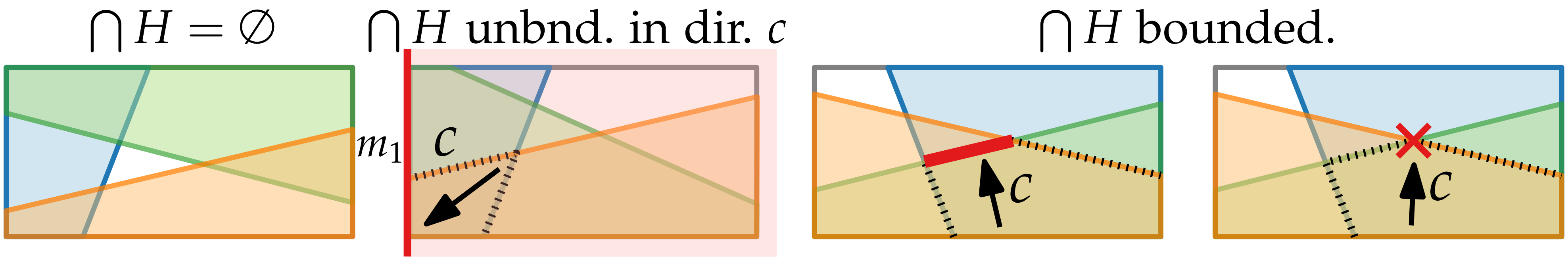


A Small Trick: Make Solution Unique



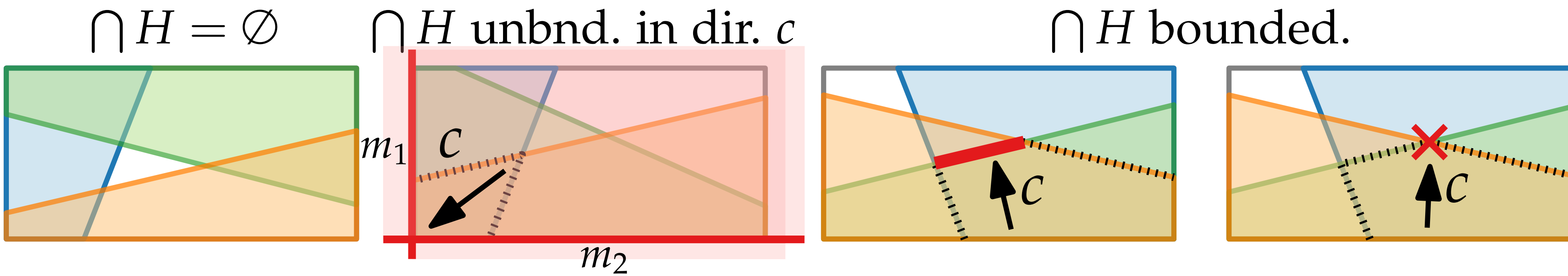
- Add two bounding halfplanes m_1 and m_2

A Small Trick: Make Solution Unique



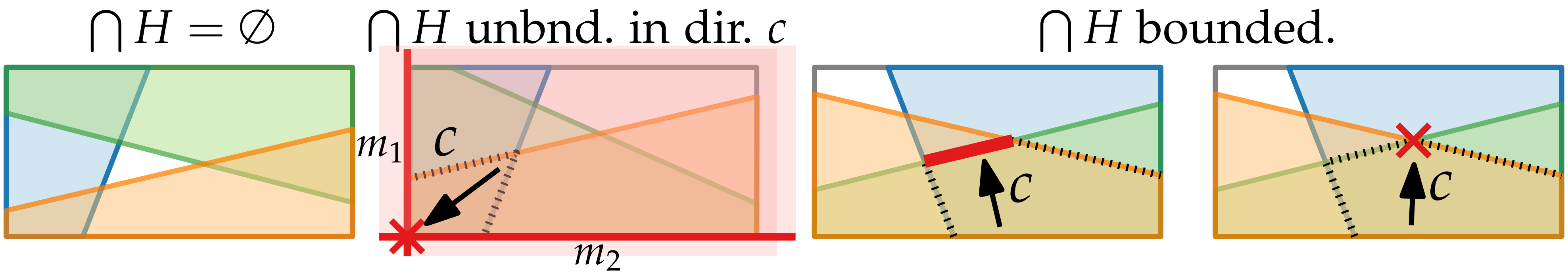
- Add two bounding halfplanes m_1 and m_2

A Small Trick: Make Solution Unique



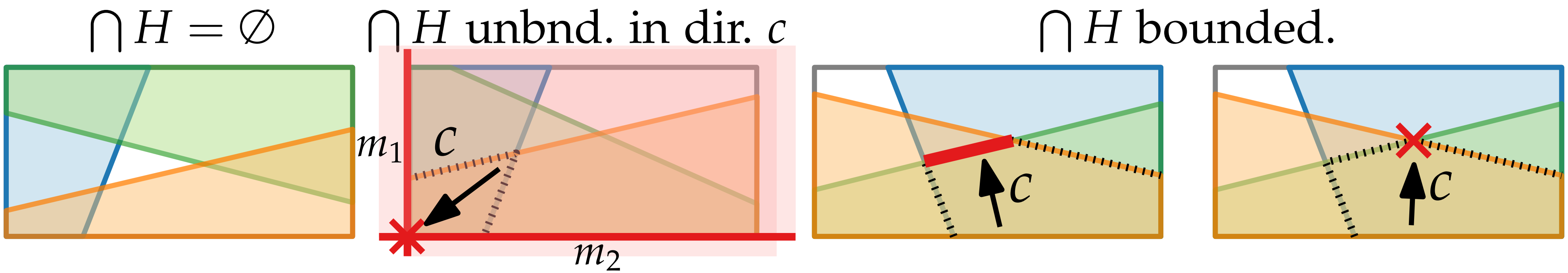
- Add two bounding halfplanes m_1 and m_2

A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

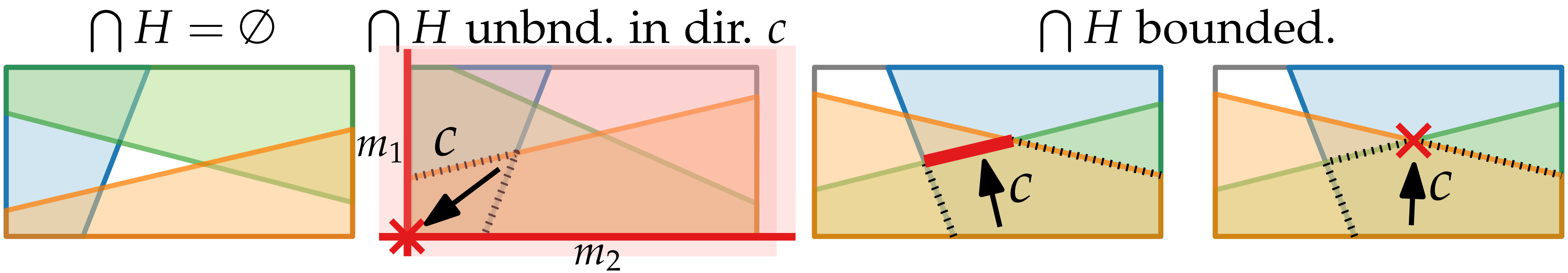
A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases} \text{ for some sufficiently large } M$$

A Small Trick: Make Solution Unique

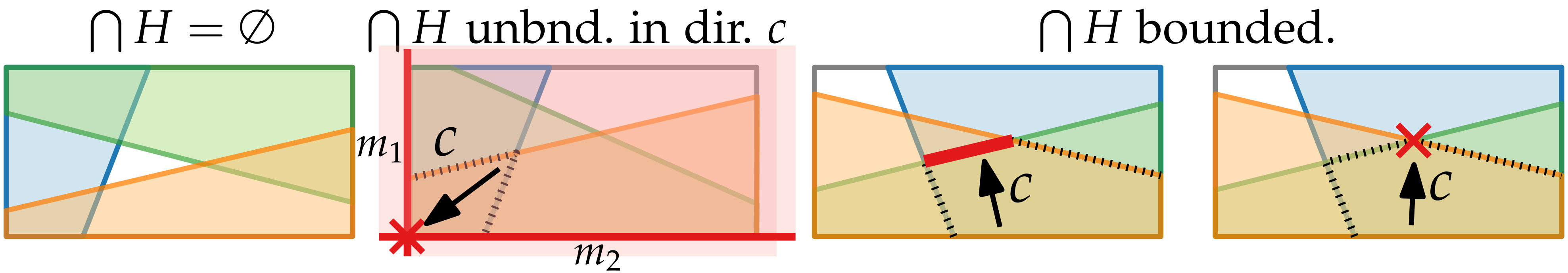


- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases} \quad \text{for some sufficiently large } M$$

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

A Small Trick: Make Solution Unique



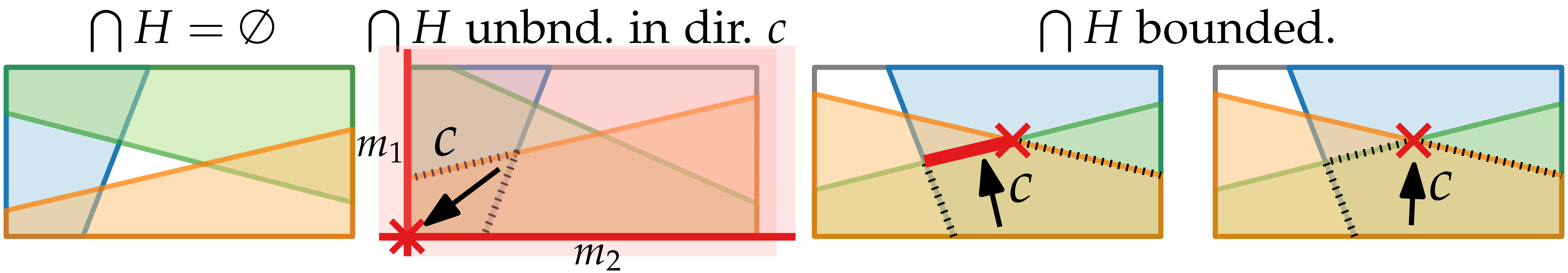
- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases} \quad \text{for some sufficiently large } M$$

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

- Take the lexicographically largest solution.

A Small Trick: Make Solution Unique



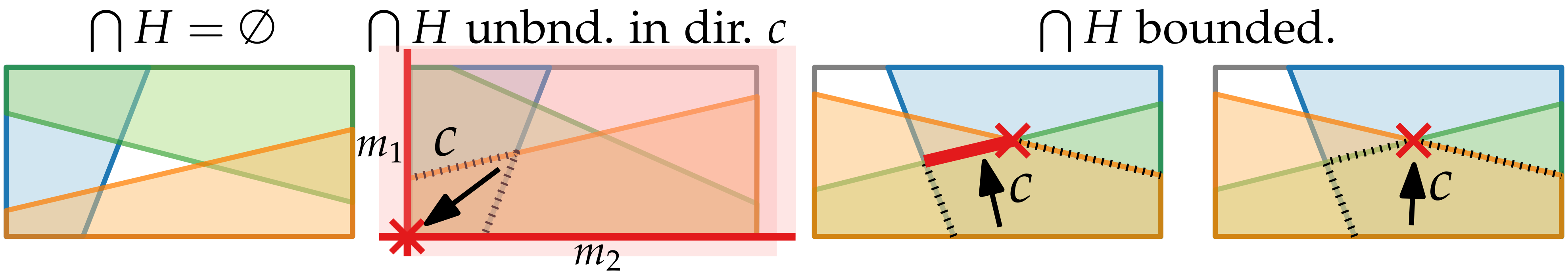
- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases} \quad \text{for some sufficiently large } M$$

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

- Take the lexicographically largest solution.

A Small Trick: Make Solution Unique



- Add two bounding halfplanes m_1 and m_2

$$m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases} \quad \text{for some sufficiently large } M$$

$$m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}$$

- Take the lexicographically largest solution.

$\Rightarrow$ Set of solutions is either empty or a uniquely defined pt.

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$

return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow$
  else
    $v_i \leftarrow$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

∂h_i

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
    if $v_{i-1} \in h_i$ **then**
        $v_i \leftarrow v_{i-1}$
    else
        $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

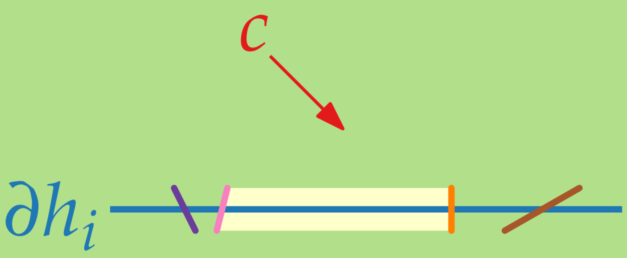
$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

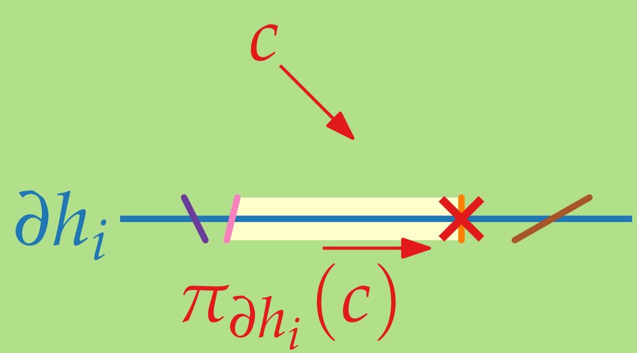


Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

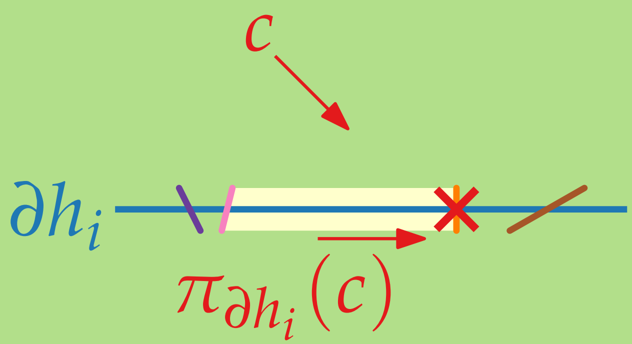


Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n

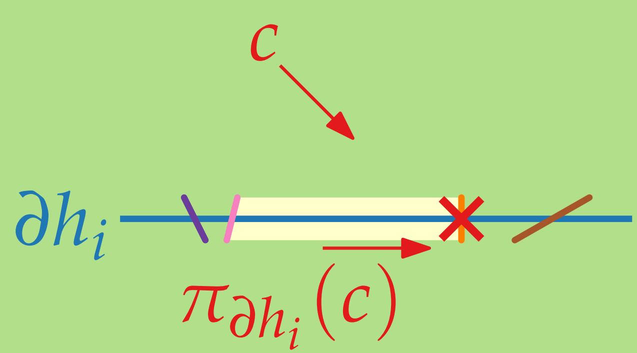


Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n



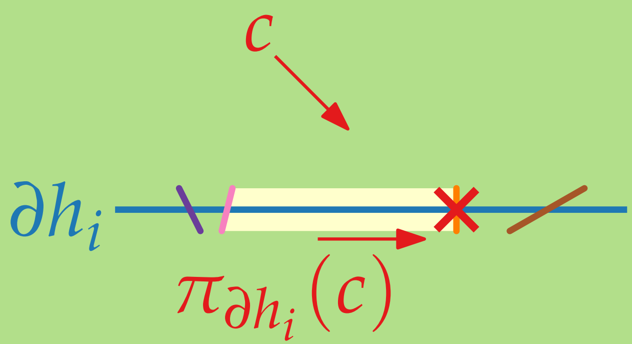
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$
return v_n



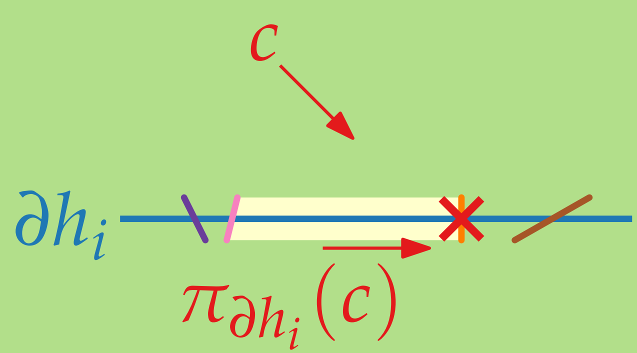
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



$O(1)$

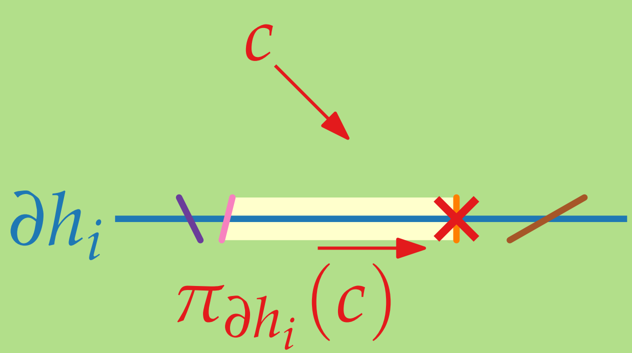
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

$\text{2DBoundedLP}(H, c, m_1, m_2)$

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$ $O(1)$
  else
    $v_i \leftarrow \text{1DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



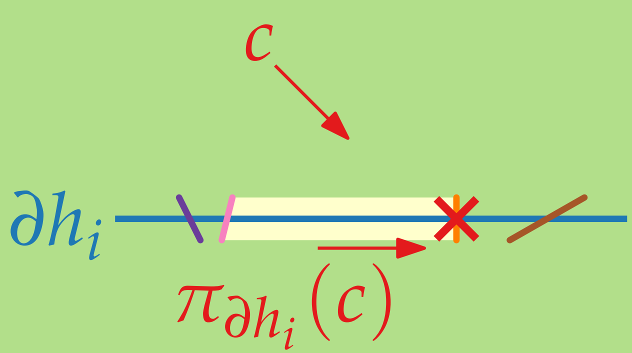
w-c running time:

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$ $O(1)$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
return v_n



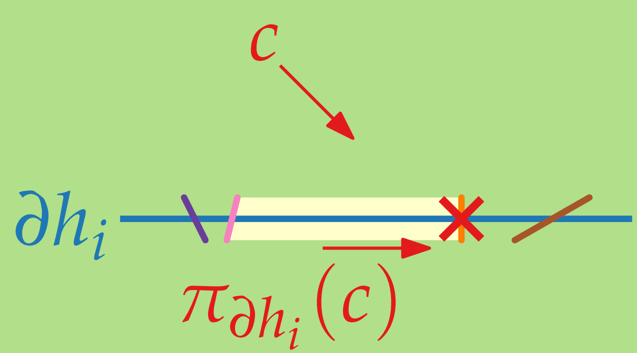
w-c running time:
$T(n) = \sum_{i=1}^{n} O(i) =$

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

2DBoundedLP(H, c, m_1, m_2)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$ — $O(1)$
  else
    $v_i \leftarrow$ 1DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ — $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ — $O(1)$
return v_n



w-c running time:
$T(n) = \sum_{i=1}^{n} O(i) =$
$= O(n^2)$:-(

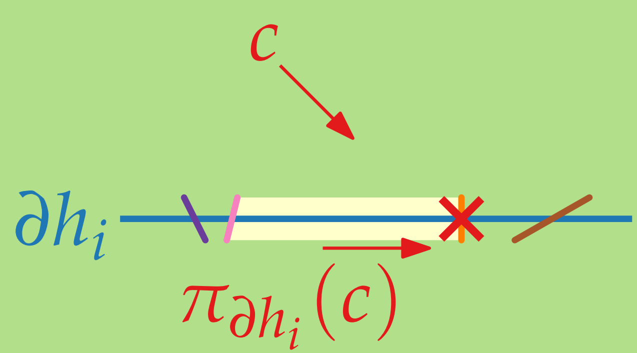
Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

Randomized

$\text{2DBoundedLP}(H, c, m_1, m_2)$

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$
for $i \leftarrow 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$ — $O(1)$
  else
    $v_i \leftarrow \text{1DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ — $O(i)$
    if $v_i =$ nil **then**
      return nil
  $H_i = H_{i-1} \cup \{h_i\}$ — $O(1)$
return v_n



w-c running time:

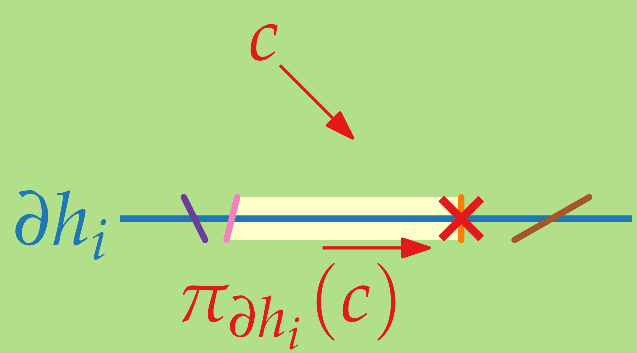
$T(n) = \sum_{i=1}^{n} O(i) =$
$= O(n^2)$:-(

Incremental Approach

Idea: Don't compute $\bigcap H$, but just *one* (optimal) point!

Randomized

2DBoundedLP(H, c, m_1, m_2)
  compute random permutation of H
  $H_0 = \{m_1, m_2\}$
  $v_0 \leftarrow$ corner of $m_1 \cap m_2$
  for $i \leftarrow 1$ **to** n **do**
    if $v_{i-1} \in h_i$ **then**
      $v_i \leftarrow v_{i-1}$ $O(1)$
    else
      $v_i \leftarrow \text{1DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$ $O(i)$
      if $v_i = \text{nil}$ **then**
        return nil
    $H_i = H_{i-1} \cup \{h_i\}$ $O(1)$
  return v_n



w-c running time:

$T(n) = \sum_{i=1}^{n} O(i) =$
$= O(n^2)$:-(

Computational Geometry

Lecture 4:
Linear Programming
or
Profit Maximization

Part V:
The Randomized-Incremental Approach

Philipp Kindermann Winter Semester 2020

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{Bmatrix} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{Bmatrix}$ (indicator random var.).

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$\mathbf{E}[T_{2d}(n)] =$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \left\{ \begin{matrix} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{matrix} \right\}$ (indicator random var.).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$
$$= \sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i)$$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \textstyle\sum \mathbf{E}[X_i] \cdot O(i) \\ &\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1] =$ probability that the optimal solution changes when h_i is added to H_{i-1}.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1] =$ probability that the optimal solution changes when h_i is added to H_{i-1}.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1] =$ probability that the optimal solution changes when h_i is added to H_{i-1}.
$=$ probability that the optimal solution changes when h_i is removed from H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &\le O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1] =$ probability that the optimal solution changes when h_i is added to H_{i-1}.
$=$ probability that the optimal solution changes when h_i is removed from H_i.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\begin{aligned} \mathbf{E}[T_{2d}(n)] &= \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\ &= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i) \\ &\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) \end{aligned}$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.
$\leq 2/i$.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \left\{ \begin{matrix} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{matrix} \right\}$ (indicator random var.).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$
$$= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i)$$
$$\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i)$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1] =$ probability that the optimal solution changes when h_i is added to H_{i-1}.

$=$ probability that the optimal solution changes when h_i is removed from H_i.

$\leq 2/i$. This is independent of the choice of H_i.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \left\{ \begin{matrix} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{matrix} \right\}$ (indicator random var.).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$
$$= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i)$$
$$\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i)$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.

= probability that the optimal solution changes when h_i is removed from H_i.

$\leq 2/i$. This is independent of the choice of H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \left\{ \begin{matrix} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{matrix} \right\}$ (indicator random var.).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\textstyle\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$
$$= \textstyle\sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i)$$
$$\leq O(n) + \textstyle\sum \mathbf{Pr}[X_i = 1] \cdot O(i) = O(n).$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.
= probability that the optimal solution changes when h_i is removed from H_i.
$\leq 2/i$. This is independent of the choice of H_i.

Result

Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random var.).

Then the expected running time is

$$\mathbf{E}[T_{2d}(n)] = \mathbf{E}[\textstyle\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$
$$= \sum \mathbf{E}[1 - X_i] \cdot O(1) + \sum \mathbf{E}[X_i] \cdot O(i)$$
$$\leq O(n) + \sum \mathbf{Pr}[X_i = 1] \cdot O(i) = O(n).$$

We fix the i random halfplanes in H_i.

$\mathbf{Pr}[X_i = 1]$ = probability that the optimal solution changes when h_i is added to H_{i-1}.

Proof technique: *Backward analysis!*

= probability that the optimal solution changes when h_i is removed from H_i.

$\leq 2/i$. This is independent of the choice of H_i.