Feasibility Jump: an LP-free Lagrangian MIP heuristic

Fan Tailin

Jan 21st, 2025

Georgia Institute of Technology

MIP

Mixed-Integer Programming Problem

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j \in N} c_j x_j \\
\text{subject to} \quad & \sum_{j \in N} a_{ij} x_j \leq b_i \quad \forall i \in M, \\
& l_j \leq x_j \leq u_j \quad \forall j \in N, \\
& x_j \in \mathbb{Z} \quad \forall j \in I.
\end{aligned}
$$

where $N = \{1, \ldots, n\}$, $M = \{1, \ldots, m\}$, $x \in \mathbb{R}^n$, $a_{ij}, c_j, b_i \in \mathbb{R}$, $l_j \in \mathbb{R}$ and $u_j \in \mathbb{R}$ are the variable bounds, and $I \subseteq N$ are the indices of variables that are constrained to take only integer values.

Georgia Institute of Technology

MIP

Relaxed MIP

$$\begin{aligned} \text{minimize} \quad & F_w(x) \\ \text{subject to} \quad & l_j \leq x_j \leq u_j \quad \forall j \in N, \\ & x_j \in \mathbb{Z} \quad \forall j \in I, \end{aligned}$$

where $F_w(x)$ is the total infeasibility penalty incurred by x. It is defined as the sum of the infeasibility penalties computed for each linear constraint:

$$F_w(x) = \sum_{i \in M} w_i f_i(x),$$

Georgia Institute of Technology

MIP

F_w

$$F_w(x) = \sum_{i \in M} w_i f_i(x),$$

where $w_i \geq 0$ is a weight associated with each constraint, and the infeasibility penalty $f_i(\cdot)$, $i \in M$, for a single linear constraint $\sum_{j \in N} a_{ij} x_j \leq b_i$ is:

$$f_i(x) = \max \left\{ 0, \sum_{j \in N} a_{ij} x_j - b_i \right\}.$$

Georgia Institute of Technology

Jump Value

Limit point

$$t_{ij}(\bar{x}_{k\neq j}) = \begin{cases} \lfloor r \rfloor & \text{if } a_{ij} > 0, \\ \lceil r \rceil & \text{if } a_{ij} < 0, \end{cases} \quad \text{where } r = \frac{1}{a_{ij}} \left(b_i - \sum_{k\neq j} a_{ik} \bar{x}_k \right),$$

where $\bar{x}_{k\neq j}$ is short for $x_k = \bar{x}_k$, $k \neq j$. Essentially t_{ij} is the edge point that x_j can jump to without violating the constraint $\sum_k a_{ik} x_k \leq b_i$. When $\bar{x}$ are fixed, t_{ij} is a constant.

$$\sum_k a_{ik} x_k \leq b_i \Leftrightarrow x_j \leq \frac{1}{a_{ij}} \left(b_i - \sum_{k\neq j} a_{ik} x_k \right) (a_{ij} > 0)$$

Jump Value

Penalty of x_j and constraint i

$$g_{ij}(t|\bar{x}_{k\neq j}) = \begin{cases} \max\left\{0, w_i\left(t - t_{ij}(\bar{x}_{k\neq j})\right)\right\} & \text{if } a_{ij} > 0, \\ \max\left\{0, -w_i\left(t - t_{ij}(\bar{x}_{k\neq j})\right)\right\} & \text{if } a_{ij} < 0, \end{cases}$$

where $t \in \mathbb{R}$ and $j \in N$.

Total penalty

$$G_j(t|\bar{x}_{k\neq j}) = \sum_{i\in M:\, a_{ij}\neq 0} g_{ij}(t|\bar{x}_{k\neq j}).$$

Georgia Institute of Technology

Jump Value

We will show that g_{ij} is in fact f_{ij}. Suppose $a_{ij} > 0$:

$$\begin{aligned} f_i(x) &\leftarrow \sum_{j \in N} a_{ij} x_j - b_i, \\ &= \sum_{k \neq j} a_{ik} x_k + a_{ij} x_j - b_i \\ &= a_{ij} \left[x_j - \frac{1}{a_{ij}} \left(b_i - \sum_{k \neq j} a_{ik} x_k \right) \right] \\ &= a_{ij} \left(x_j - t_{ij} \right). \end{aligned}$$

Recalling that $F^w(x) = \sum_{i \in M} w_i f_i(x)$, we define $f_{ij} = a_{ij} w_i (x_j - t_{ij})$.
Compare f_{ij} with g_{ij}, one can find that f_{ij} normalized by a_{ij} is g_{ij}.
However, according to the authors, using f_{ij} did not perform as well as g_{ij}.

Jump Value

$$\mathrm{Jump}_j(\bar{x}_{k\neq j}) = \min\left(\underset{\{t\in T_j(\bar{x}),t\neq\bar{x}_j\}}{\arg\min} G_j(t|\bar{x}_{k\neq j})\right).$$

$$T_j(\bar{x}) = l_j \cup u_j \cup \bigcup_{i\in M} t_{ij}(\bar{x}_{k\neq j}).$$

Note

- G_j and g_{ij} are convex (piece-wise linear).
- g_{ij} is a single variable function.

Georgia Institute of Technology

ALGO I: Jump Value

Algorithm 1 Jump value

Input Problem (2), incumbent solution $\bar{x}$, and variable index j
Output: The *jump* value.

$t^* \leftarrow l_j$ ▷ *Initialize the best value*
$\Delta \leftarrow \left[(l_j, 0), (u_j, 0)\right]$ ▷ *Initialize the list of (value,slope) pairs*
slope $\leftarrow 0$ ▷ *Initialize the cumulative slope*
for $i \in M$ where $a_{ij} \neq 0$ **do**
  $t \leftarrow t_{ij}(\bar{x}_{k \neq j})$ ▷ *Compute the critical value*
  if $l_j \leq t \leq u_j$ **then**
    Δ.insert$\big((t, w_i)\big)$ ▷ *Add feasible critical value and slope change*
  if $t \geq l_j$ **and** $a_{ij} < 0$ **then**
    slope $\leftarrow$ slope $- w_i$ ▷ *Accumulate the negative slopes before* l_j
  if $t < l_j$ **and** $a_{ij} > 0$ **then**
    slope $\leftarrow$ slope $+ w_i$ ▷ *Accumulate the positive slopes before* l_j
for $(t, w) \in$ sorted(Δ) **do** ▷ *Iterate* Δ *in ascending order by the first component*
  slope $\leftarrow$ slope $+ w$ ▷ *Update the current slope*
  if $t = \bar{x}_j$ **then**
    continue ▷ *Skip values equal to the current incumbent*
  $t^* \leftarrow t$ ▷ *Update the best value*
  if slope ≥ 0 **then**
    break ▷ *Stop when the slope becomes non-negative*
return t^*

titute ogy

Note for line 8 to line 11

Take the case $t_j < l_j$ for example.

$$\frac{1}{a_{ij}}(b_i - \sum_{k \neq j} a_{ik}\bar{x}_k) < l_j.$$

If $a_{ij} < 0$:

$$b_i > l_j a_{ij} + \sum_{k \neq j} a_{ik}\bar{x}_k,$$

which is feasible. Otherwise $a_{ij} > 0$, the constraint will be violated.

Example of piece-wise linear function

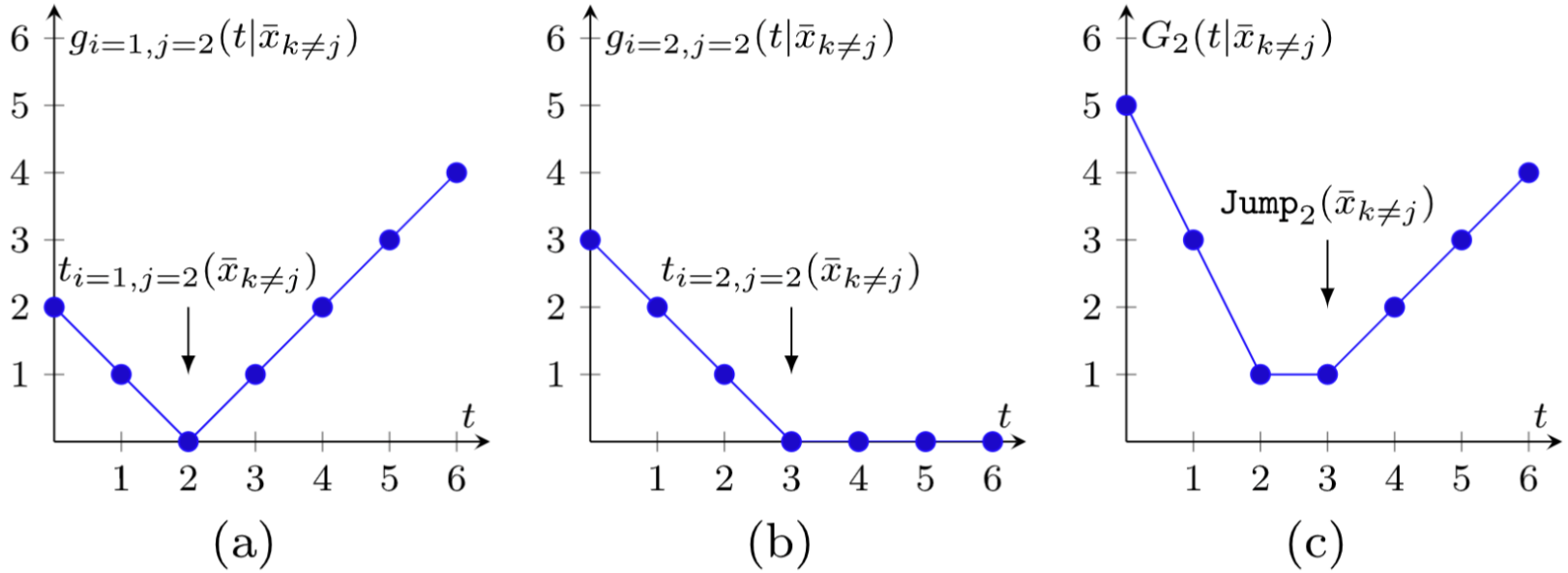

Fig. 1 The constraint violation functions for variable x_2 (**a**, **b**), and their sum (**c**)

Constraints: $x_1 + x_2 = 3,\ x_2 + x_3 \geq 3$. Given $\bar{x}_1 = 1, \bar{x}_2 = 2, \bar{x}_3 = 0$, and $w_1 = w_2 = 1$. In this case, $T_2(\text{barx}_{k\neq j}) = \{0, 2, 3, +\infty\}$

Georgia Institute of Technology

Feasibility Jump

Score

$$s_j = G_j(\bar{x}_j|\bar{x}_{k\neq j}) - G_j(v_j|\bar{x}_{k\neq j}).$$

where $t \in \mathbb{R}$ and $j \in N$. By the definition of G_j, jumping to v_j when $s_j > 0$ shrinks the total violation penalty, and do so when $s_j < 0$ increases the total violation penalty.

Lazy adaptive neighborhoods

Only update the neighborhood v_j of x_j that performs a jump. Other v_k remains the same, but update all variables' scores $s_j, j \in N$.

Georgia Institute of Technology

Guide the search

Updating weights

$$w_l \leftarrow \Delta W(w, \bar{x}, t),$$

where ΔW is some weight update function that depends on the current local minimum.

Strategy I

$$\Delta W^*(w, \bar{x}, i) = \begin{cases} w_i & \text{if } \sum_{j \in N} a_{ij} \bar{x}_j \leq b_i, \\ \lambda w_i & \text{otherwise,} \end{cases}$$

Strategy II

$$\Delta W^+(w, \vec{x}, t) = \begin{cases} w_i & \text{if } \sum_{j \in \mathcal{N}} a_{ij} \vec{x}_j \leq b_i, \\ w_i + 1 & \text{otherwise.} \end{cases}$$

ALGO II

Algorithm 2 Feasibility Jump

Input Problem (2), initial assignment $\tilde{x}$
Output: a feasible solution or NULL

$x^* \leftarrow$ NULL ▷ *Initialize best feasible solution*
$\bar{x} \leftarrow \tilde{x}$ ▷ *Initialize incumbent*
$w_i = 1,\ i \in M$ ▷ *Initialize weights*
$v_j = \text{Jump}_j(\bar{x}_{k \neq j}),\ j \in N$ ▷ *Initialize promising values*
$s_j = G_j(\bar{x}_j | \bar{x}_{k \neq j}) - G_j(v_j | \bar{x}_{k \neq j}),\ j \in N$ ▷ *Initialize scores*
$P = \{j \in N : s_j > 0\}$ ▷ *Initialize set of indices with positive score*
while SHOULDTERMINATE() is **false do**
  if $F^w(\bar{x}) = 0$ **then**
    $x^* \leftarrow \bar{x}$
    break ▷ *Found a feasible solution*
  if $P = \emptyset$ **then** ▷ *Whether we have reached a local minimum*
    $U \leftarrow \emptyset$ ▷ *Set of violated constraints*
    for $i \in N : f_i(\bar{x}) > 0$ **do**
      $w_i \leftarrow w_i + 1$ ▷ *Put more emphasis on satisfying this constraint*
      $U \leftarrow U \cup i$
    Update scores $s_j : a_{ij} \neq 0, i \in U, j \in N$
    Update P
    $i^* =$ random choice in U ▷ *Random violated constraint*
    $j^* = \underset{j \in N : a_{i^* j} \neq 0}{\arg\max} s_j$ ▷ *Best move in this constraint*
  else
    $P^* =$ randomly choose up to 25 indices from P
    $j^* = \underset{j \in P^*}{\arg\max}\, s_j$ ▷ *Best move among a random set of moves with $s_j > 0$.*
  $\bar{x}_{j^*} \leftarrow v_{j^*}$ ▷ *Make the move*
  $v_{j^*} = \text{Jump}_{j^*}(\bar{x}_{k \neq j^*})$ ▷ *Recompute jump value*
  Update scores s_j of the neighboring variables
  Update P
return x^*

Georgia Institute of Technology

MIPLIB 2017 benchmark set instances based on the time to feasibility (TF) of FJ versus XPR

Category		# instances		
No solutions found		33	128	240
FJ found no solution but XPR did		84		
FJ found a solution after XPR		11		
FJ found a solution before XPR	during presolve	88	112	
	during root node	24		
	during branching	0		

Figure: In particular, when FJ is faster than XPR, we check whether FJ found a feasible solution (1) during presolve, (2) during root node (either while running the LP solver, other feasibility heuristics, or root cutting heuristics), or (3) during branching

Georgia Institute of Technology