带宽分配问题的动态离散化发现 (DDD) 框架

算法整合与模型定义

2025 年 9 月 23 日

1 DDD 框架概述

动态离散化发现 (Dynamic Discretization Discovery, DDD) 是一种为求解大规模连续时间或精细离散时间优化问题而设计的精确算法框架。其核心思想是避免一次性构建和求解一个包含所有时间点的、规模庞大的完整模型。取而代之的是，DDD 从一个仅包含少数关键时间点的“粗糙”模型出发，通过迭代循环，智能地“发现”并添加那些对求解至关重要的时间点，逐步精炼模型。在每一轮迭代中，算法都会计算当前问题的下界 (LB) 和上界 (UB)，直到两者之间的差距收敛到预设精度，从而在可控的计算成本内找到并证明原始问题的最优解。

2 DDD 主算法流程

下面给出了应用于带宽分配问题的完整 DDD 算法框架的伪代码。

Algorithm 1 带宽分配问题的动态离散化发现 (DDD) 算法

输入: 原始问题 (FPP) 参数 $(I, J, T_{\text{fine}}, L_j, D_{it}, C_j, a_{ij})$, 收敛容差 ϵ

输出: FPP 问题的 (ϵ)-最优解 best_solution 及其成本 UB

// 第 0 步: 初始化

T_points $\leftarrow$ {t_start, t_peak, t_end} ▷ 例如: {1, 252, 288}

$LB \leftarrow -\infty$, $UB \leftarrow +\infty$

best_solution $\leftarrow \emptyset$

while $\frac{UB-LB}{UB} > \epsilon$ **do**

// 步骤 A: 求解下界模型

$\mathcal{T}_{\text{intervals}} \leftarrow$ `CreateIntervalsFromPoints`(T_points)

$(z^*_{LBM}, x^{LBM}, y^{LBM}) \leftarrow$ `Solve_LBM`($\mathcal{T}_{\text{intervals}}$) ▷ 见模型 M1

$LB \leftarrow \max(LB, z^*_{LBM})$

// 步骤 B: 修复解并更新上界

$(z_{\text{heuristic}}, \text{solution\_feasible}) \leftarrow$ `Repair_and_Get_UB`(x^{LBM}, y^{LBM}) ▷ 见算法 2

if $z_{\text{heuristic}} < UB$ **then**

$UB \leftarrow z_{\text{heuristic}}$

best_solution $\leftarrow$ solution_feasible

end if

// 步骤 C: 检查收敛

if $\frac{UB-LB}{UB} \leq \epsilon$ **then**

break

end if

// 步骤 D: 精炼离散化

$t_{\text{new}} \leftarrow$ `Refine_Discretization`($x^{LBM}, y^{LBM}, \mathcal{T}_{\text{intervals}}$) ▷ 见算法 3

if t_{new} is not null **then**

T_points $\leftarrow$ T_points $\cup \{t_{\text{new}}\}$

else

break ▷ 若无法找到新的精炼点，则终止

end if

end while

return best_solution, UB

3 核心组件的数学定义

3.1 模型 M1: 下界模型 (LBM)

此模型基于粗粒度时间区间 $\tau \in \mathcal{T}$ 进行建模，其解为原始问题最优解的下界。

$$
\begin{aligned}
\min \quad & \sum_{j \in J} b_j \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_{ij\tau} = D_{i\tau} && \forall i \in I, \forall \tau \in \mathcal{T} \\
& \sum_{\tau \in \mathcal{T}} y_{j\tau} = L_j && \forall j \in J \\
& 0 \le y_{j\tau} \le |\tau| && \forall j \in J, \forall \tau \in \mathcal{T} \\
& \sum_{i \in I} a_{ij} x_{ij\tau} \le C_j \cdot |\tau| && \forall j \in J, \forall \tau \in \mathcal{T} \\
& \sum_{i \in I} a_{ij} x_{ij\tau} \le b_j \cdot (|\tau| - y_{j\tau}) + C_j \cdot y_{j\tau} && \forall j \in J, \forall \tau \in \mathcal{T} \\
& b_j \ge 0, \quad x_{ij\tau} \ge 0, \quad y_{j\tau} \in \mathbb{Z}_{\ge 0}
\end{aligned}
$$

3.2 算法 2: 上界修复与计算

此算法借鉴 UBM 的 “分而治之” 思想，但由 LBM 的解来指导每个区间的免费槽 “预算”。

Algorithm 2 上界修复与计算 (`Repair_and_Get_UB`)

输入: LBM 解 x^{LBM}, y^{LBM}
输出: 可行解成本 $z_{\text{heuristic}}$, 可行解 solution_feasible
local_costs $\leftarrow \{\}$, feasible_sub_solutions $\leftarrow []$
for each coarse interval τ **do**
    $(b^*_{j\tau}, x^*_{\text{sub}}, y^*_{\text{sub}}) \leftarrow$ `Solve_Subproblem_MIP`$(\tau, y^{LBM}_{j\tau})$ ▷ 见模型 M2
    local_costs$[j, \tau] \leftarrow b^*_{j\tau}$ for all j
    feasible_sub_solutions.append$((x^*_{\text{sub}}, y^*_{\text{sub}}))$
end for
solution_feasible $\leftarrow$ `Concatenate`(feasible_sub_solutions)
for each provider $j \in J$ **do**
    $\hat{b}_j \leftarrow \max_\tau$\{local_costs$[j, \tau]$\}
end for
$z_{\text{heuristic}} \leftarrow \sum_{j \in J} \hat{b}_j$
return ($z_{\text{heuristic}}$, solution_feasible)

3.2.1 模型 M2: 上界修复子问题

此模型在每个粗糙区间 τ 内进行精细化的求解。

$$
\begin{aligned}
\min \quad & \sum_{j \in J} b_{j\tau} \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_{ijt} = D_{it} && \forall i \in I, \forall t \in \tau \\
& \sum_{i \in I} a_{ij} x_{ijt} \le C_j && \forall j \in J, \forall t \in \tau \\
& \sum_{i \in I} a_{ij} x_{ijt} \le b_{j\tau} + C_j \cdot y_{jt} && \forall j \in J, \forall t \in \tau \\
& \sum_{t \in \tau} y_{jt} = y_{j\tau}^{LBM} && \forall j \in J \\
& x_{ijt} \ge 0, \quad b_{j\tau} \ge 0, \quad y_{jt} \in \{0, 1\}
\end{aligned}
$$

3.3 算法 3: 离散化精炼

此算法通过求解可行性检验子问题来严谨地发现下一个需要被添加到离散化集合中的关键时间点。

Algorithm 3 离散化精炼 (`Refine_Discretization`)

输入: LBM 解 x^{LBM}, y^{LBM}, 粗糙区间集合 $\mathcal{T}_{\text{intervals}}$
输出: 新的关键时间点 t^*
for each coarse interval $\tau \in \mathcal{T}_{\text{intervals}}$ **do**
    status $\leftarrow$ `Solve_Feasibility_Check_MIP`(τ, x^{LBM}, y^{LBM}) ▷ 见模型 M3
    if status is INFEASIBLE **then**
        IIS $\leftarrow$ `Get_IIS_from_solver()` ▷ 获取不可约不一致子系统
        $t^* \leftarrow$ `Extract_Time_Point_from_IIS(IIS)`
        return t^*
    end if
end for
return `null`

3.3.1 模型 M3: 可行性检验子问题

此模型的目标仅为检验 LBM 的解在一个区间 τ 内是否可以被细化为一个满足所有原始约束的解。

$$
\begin{aligned}
\min \quad & 0 \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_{ijt} = D_{it} && \forall i \in I, \forall t \in \tau \\
& \sum_{i \in I} a_{ij} x_{ijt} \le C_j && \forall j \in J, \forall t \in \tau \\
& \dots && \text{(所有其他必须满足的精细约束)} \\
& \sum_{t \in \tau} x_{ijt} = x^{LBM}_{ij\tau} && \forall i \in I, j \in J \\
& \sum_{t \in \tau} y_{jt} = y^{LBM}_{j\tau} && \forall j \in J \\
& x_{ijt} \ge 0, \quad y_{jt} \in \{0,1\}
\end{aligned}
$$

4 最优性保证

该 DDD 框架是一个精确算法，能够保证在有限次迭代后找到原始问题的最优解（或在容差 ϵ 内的近似最优解）。其保证基于以下三点：

1. **有限终止性**: 在每次未收敛的迭代中，算法都会从有限的精细时间点集合 T_{fine} 中添加至少一个新点到 T_points。因此，算法最多迭代 $|T_{\text{fine}}|$ 次后必然终止。

2. **边界收敛性**: LBM 作为松弛模型，其目标值 (LB) 在迭代过程中单调不减。而 UB 作为可行解的成本，在迭代中单调不增。上下界必然会相互逼近。

3. **有效精炼性**: 关键的精炼步骤 (算法 3) 通过求解可行性检验问题，能严格证明当前 LBM 解的不可行性，并找到导致该不可行性的关键时间点 t^*。将 t^* 加入离散化集合，可以确保下一轮的 LBM 不会再犯完全相同的 “错误”，从而保证算法不断取得进展，最终收敛到最优解。