arXiv:2303.01419v1 [cs.DM] 2 Mar 2023

硬性节点存储约束下的动态离散化发现

Madison Van Dyk[a,*] Jochen Koenemann*

摘要

近期提出的动态离散化发现（DDD）是一种强大的方法，能够使许多时间相关问题更易于处理。尽管 DDD 已应用于各种问题，但其中一项特殊挑战是如何处理存储约束，同时避免在每次迭代中导致松弛过弱。具体而言，目前在连续情形下处理某些硬性存储约束的方法，是在 DDD 的每次迭代中完全移除一部分存储约束。

在这项工作中，我们证明，对于离散问题，这种弱松弛并非必要。具体而言，我们为每次迭代中必须允许的额外存储量给出了界限。我们以有界节点存储条件下的经典通用分组路由问题为例，展示了这些技术；目前，该问题只能通过整数规划求解。我们给出了计算结果，证明 DDD 在求解通用分组路由问题时的有效性。

关键词 — 路由、时间扩展网络、动态离散化发现、节点存储、分组路由。

1 引言

许多具有实际重要性的应用，涵盖配送规划和通信系统等领域，都可以表述为网络设计与路由问题 [32]。实际路由问题往往会因额外的时间因素而变得复杂；决策通常需要考虑运输时间、弧吞吐量和节点存储。

求解带有额外时间考量的路由问题，主要有两类模型：连续形式和时间索引公式。连续形式使用连续变量来模拟时间决策，而时间索引公式则为每个可能做出决策的时间点分别设置变量和约束 [21]。离散时间时序问题的时间索引公式，是通过将时序问题表示为相应时间扩展网络中的静态问题得到的。设$D = (N,A)$为一个具有弧通行时间 τ 的有向图，T 为所考虑的时间范围

*滑铁卢大学组合学与优化系，加拿大安大略省滑铁卢市，邮编 N2L 3G1。本研究由 NSERC探索资助计划资助，项目编号 RGPIN-03956-2017。[a]通讯作者。电子邮箱：madison.vandyk@uwaterloo.ca。

。相应的时间扩展网络$D_T = (N_T, A_T)$由每个节点$v \in N$在每个时间点$t \in [T] := \{0, 1, \ldots, T\}$的一份副本，以及每条弧在每个出发时间的一份副本组成。

虽然连续形式比时间索引公式更加紧凑，但它们需要“大 M”约束，而这会因分支定界树过大而导致求解时间较长 [21]。时间索引公式的线性规划松弛更强，但这种时间扩展常常会使问题难以实际求解，因为网络规模会随T [12, 13]线性增长。

缩减时间扩展网络数学规划 formulation 的规模，一直是理论和应用优化领域的活跃研究方向。Boland 等人 [1, 33] 最近提出了动态离散化发现 (DDD)框架，该框架通过仅使用部分时间索引变量和约束来求解某些类别的时序问题。DDD算法在仅包含完整时间扩展网络中部分节点和弧的网络上，求解一系列整数规划（IP）；这类网络称为部分时间扩展网络。构造部分时间扩展网络时，需确保相应数学规划 formulation 为原问题的最优值提供下界。如果无法将部分时间扩展网络的解转换为原实例中成本相同的解，我们便通过添加节点和弧来细化该部分时间扩展网络。DDD的主要优势在于，它能确定求解问题所需的时间点，而且可能始终无需构建完整的时间扩展网络。

过去几年中，DDD 已被应用于多个问题，包括带时间窗的旅行商问题 (TSP-TW)[33] 以及连续时间服务网络设计（*SND*）问题[1]。对于这些连通性问题，节点没有存储容量约束。因此，下界正确性的证明依赖于这种自由度，使得提前到达的流不会导致不可行。这种对无限节点存储的需求此前使 DDD 模型无法处理许多现实问题，例如某些动态调度问题；对于这些问题的大规模实例，目前只有启发式技术可用 [22]。Boland和Savelsbergh [2]曾指出，消除 DDD 对无界零成本存储的依赖，是未来研究的一个重要方向。Lagos 等人 [21] 最近解决了这一不足，并将 DDD 方法扩展到求解连续库存路径问题 (CIR)，该问题采用往返路线并具有存储容量约束。不过，在其连续情形中，他们在每次迭代中都完全放宽一部分存储约束。在附录A中，我们将说明，对于时间点取自 $\{0, 1, \ldots, T\}$ 的离散设置，这种方法可能导致 DDD 需要 $\Omega(T)$ 次迭代；而采用我们改进的界限，DDD 只需一次迭代即可终止。

虽然 DDD 常用于没有预先指定离散化的连续问题，但许多实际路由问题本质上是离散的，例如 Lara 等人 [22]考虑的动态调度问题。Hewitt 和 Crainic [4] 还指出，在许多应用中，连续问题会被建模为离散问题，其时间粒度根据具体应用选择。例如，实践中通常存在最小时间离散化单位，如一小时或 15 分钟。鉴于此，对于采用离散时间离散化的问题，我们收紧 Lagos 等人 [21]提出的存储约束松弛。具体而言，我们认为每次迭代都不必完全移除一部分节点存储约束，并证明了基于容量

和基础图结构的存储需求界限。解决 Lara 等人 [22] 考虑的动态调度问题，核心挑战在于数据包路由问题。由于本文的主要贡献是进一步发展 DDD 方法，我们聚焦于一个特定问题，即所谓的通用分组路由问题（UPR）。这使我们能够清晰地展示 DDD 方法如何扩展到有界节点存储问题，同时避免引入此前工作已处理过的繁杂约束。此外，与 [21] 中提出的仅适用于具有往返网络结构（即星形图）的图的界限不同，我们的界限适用于任意底层图结构。

在本文中，我们将 DDD 模型扩展至 UPR。UPR 是一种具有有界存储的动态调度问题，涉及在通信网络中对数据包进行路由和调度。由于有界存储这一假设，目前已知解决该问题的唯一方法是采用整数规划。在 UPR 中，给定一个有向图 $D = (N,A)$，我们称之为平面网络或基础网络。每条弧 $a \in A$ 都有一个对应的运输时间 $\tau_a \in \mathbb{N}$，以及一个容量 $u_a \in \mathbb{N}$，表示可同时通过弧 a 的数据包最大数量。令 K 表示由 k 个数据包组成的集合，每个数据包都有一个对应的起点 s_k 和目的地 t_k。如果数据包既不在起点，也不在目的地，我们就称其处于活动状态。此外，每个节点 $v \in N$ 都有存储容量 $b_v \in \mathbb{N}$，这意味着它在任何时刻最多可存储 b_v 个活动数据包。调度的完工时间是任意数据包到达其目的地的最晚时间。UPR 的目标是在遵守任意时刻的弧容量和节点存储量的前提下，将所有数据包路由通过网络，从而使调度的完工时间最小。该问题是 NP困难的；即使是对所有 $v \in N$ 均有 $b_v = 0$ 的特殊情形，其近似难度也不低于顶点着色 [3]。也就是说，假设 $NP \subsetneq ZPP$，对于任意 $\varepsilon > 0$，都难以在 $\Omega(n^{1-\varepsilon})$ 范围内进行近似，其中 n 表示图 [9]中的顶点数。通常，弧容量和节点存储量分别称为带宽和缓冲区。我们可以通过运行贪心算法等方法，假设已知调度最小完工时间的一个上界 T，其中 T^*。

1.1 我们的贡献

本文的贡献兼具算法和理论两方面。我们表明，在许多离散设置中，无需移除下界模型中一部分节点存储约束。相反，我们证明了基于当前时间扩展网络以及底层网络中的弧容量和存储容量，在每个时间节点必须允许的存储量的上界。此外，我们扩展了 DDD 框架，使其能够处理一般图上具有有界节点存储量的时序问题，而非仅限于只有单个节点的度大于 1 的受限往返结构。

在附录B中，我们表明，针对松弛的节点容量所得的界限可以扩展到连续情形，并证明我们的构造推广并收紧了 Lagos 等人提出的车辆等待约束松弛 [21]。具体而言，我们表明，在连续情形中，DDD 的每次迭代中，我们的构造都会导致移除 [21], 中所移除约束的一个子集，并且在某些设置下，该子集是严格子集。我们的结果推广了 Lagos 等人的结果，因为图不再局限于星形图。我们贡献的主要组成部分如下：

1. 我们提出了一种 并为具有有界节点存储的时间索引问题实现下界模型和细化过程 s

；2. 为证明下界模型实际上是一种松弛，我们基于从完全时间扩展网络到部分时间扩展网络的映射所呈现的结构特征，提出论证；3. 我们证明，使用下界和细化过程后，算法至多经过 $|N|T^*$ 次迭代即可终止并得到最优解，其中 N 是基础图中的节点集合，T^* 是最小完工时间；4. 我们在两类实例上实现并测试了 DDD 算法：一类基于美国的人口中心，另一类基于社交网络。对于一类地理实例，当已知上界为 $2T^*$ 时，我们的 DDD 算法所需时间平均为求解完整时间索引公式所需时间的 53%；对于一类几何实例，平均为 49%。我们还表明，底层图越稀疏，DDD 算法的表现越好。

传统上，DDD 研究的是网络设计问题，其任务是通过网络发送流，并在特定时间购买弧上的容量，以便输送该流。对于 UPR问题，输入给定的是带有容量的网络，任务是找到通过该网络的最优路由。我们对 DDD 的应用进一步表明，DDD 有望使各种时间问题变得更易处理。

1.2 其他相关工作

Ford和Fulkerson [12, 13]最早在网络流理论的背景下提出了时序网络设计与路由问题。Ford和Fulkerson 表明，这些“随时间流动”的问题可以改写为相应时间扩展网络中的静态网络流问题。关于时序流的一般背景，读者可参阅 Skutella [32]近期发表的综述。对于多商品流，Hall 等人 [15]给出了困难性证明，并指出了一些多项式时间可解实例。本文研究的 UPR问题是一种随时间变化的多商品流问题，并附加了流量值必须为整数的约束。本文介绍的理论与算法技术可以推广到更一般的 UPR分数变体。

对于允许分数值的时变网络流，Fleischer等人 [10, 11] 给出了这样的保证：当我们允许使用粗粒度网络，并且仅每隔 Δ 单位时间包含顶点副本时，最优解的成本增幅有上界。在这种 Δ压缩方法中，每个节点采用相同的离散化 Δ；而在 DDD 的部分时间扩展网络中，每个节点的离散化则是不均匀的。Wang和Regan [35] 证明，对于 TSP-TW，迭代细化时间窗口离散化将收敛到最优解。类似地，Dash等人 [7]基于预处理方案迭代细化一组时间段，这与 DDD 中的动态方案不同。

动态离散化发现

DDD 框架最初应用于连通性问题，例如最短路径问题 [16] 以及带时间窗的旅行商问题 [33]。在 SND [1] 中，卡车

具有容量，但在任何给定时间，沿指定弧行驶的卡车数量没有上限。Scherr 等人 [31] 和 Hewitt [18] 将 DDD 应用于 SND 的变体时，也采用了相同的假设。这些假设避免了复杂的容量约束；这类约束在将解从完全时间扩展网络映射到部分时间扩展网络（反之亦然）时可能会造成问题。关于 DDD 框架的完整介绍，请读者参阅 Boland和Savelsbergh[2]的综述。

在 Pottel 和 Goel [29], 研究的时变持续时间与资源消耗约束调度问题中，节点上的资源约束可以用弧容量进行编码。Lagos 等人[21] 研究了连续库存路径问题 （CIR），其中一家公司负责管理其客户的库存，并从单一设施配送产品。每次配送仅限服务一位客户，之后卡车必须返回设施。因此，他们的结果仅适用于图为星形图的情况。作者还对两种不同的存储容量约束进行了编码。虽然每位客户都有存储上限，但作者假设，除非安排了配送，否则送达客户位置的产品不会影响存储水平。在许多问题中，包括 UPR，所有静态流量都必须计入某个节点的存储水平。这与 Lagos 等人 [21] 提出的约束更为接近：任一时刻至多只能有一辆车访问某个固定客户。为了在其下界模型中放宽这一约束，作者完全移除了一部分车辆存储约束。

除了建模技术之外，算法工程领域也涌现出越来越多从算法工程角度研究 DDD 的工作 [25, 31, 34]。Marshall 等人 [25] 提出了基于区间的动态离散化发现算法 （DDDI），经验证，该算法求解 SND 实例的速度比传统 DDD 快几个数量级。Scherr 等人 [31] 建议，如果节点和弧不再是获得高质量解所必需的，就将其移除。Hewitt [17] 探讨了 DDD 的加速技术，包括采用 DDD 两阶段实现等改进，以及在每次迭代中添加有效割以强化松弛模型。

分组路由

文献中的分组路由涵盖了广泛的问题，与本文对 UPR 的定义密切相关。在存储转发分组路由（SF-PR）中，弧在任意给定时间只能容纳一个数据包，且运输时间为单位长度。此外，每个数据包都有一条指定路径，必须沿该路径通过底层网络。拥塞 C 表示基础图中使用同一条弧的路径最大数量，而扩张度 D 表示数据包必须经过的路由路径的最大长度。SF-PR 的O（C +D）近似算法最初由 Leighton 等人开发 [23, 24]。在这些初始结果的基础上，多篇论文针对具有任意弧容量和运输时间的 SF-PR 建立了多项式时间近似结果 [28, 30]。然而，对于一般图，这些保证中的常数仍然很大 [28]（一般图为 39（C +D），而当运输时间和弧容量均为单位值时为 23.4（C +D））。Di Ianni 证明了 SF-PR 是 NP困难问题 [8]。Peis 等人 [27, 28] 将 SF-PR 推广到具有任意弧容量和运输时间的情形，并证明该问题是 APX难的。我们注意到，我们的 UPR问题更为一般，因为我们加入了节点存储约束。在本文中，我们将这一变体称为

UPR：其中每个数据包在基础网络中均被指定了一条路径，该问题称为固定路径通用分组路由（UPR-FP）。

当前针对数据包路由问题（路径可变）的近似算法策略，是为每个数据包选择一条合适的路径，将问题转化为一个 UPR-FP 实例。Busch 等人 [3]研究了无缓冲数据包路由问题：数据包一旦注入网络，就不能存储在任何节点上。作者证明，该问题不仅是 NP困难问题，而且其近似难度与顶点着色问题相同。假设 NP $\subsetneq$ ZPP，则顶点着色无法在 $\varepsilon > 0$ 的条件下于 $\Omega(n^{1-\varepsilon})$ 范围内得到近似，其中n 表示图 [9] 中的顶点数。为了确保 UPR 实例的可行解确实包含在某个时间索引公式中，我们必须允许时间索引变量涵盖不超过最小工期某个已知上界的时间，T^*。根据前文讨论，我们不应指望能够知道T 在T^* 上的上界，其中$T \approx T^*$。这说明采用 DDD 求解该问题十分重要，因为我们发现，与在完整的时间扩展网络上求解相比，所提供的上界对 DDD 运行时间的影响小得多。

1.3 路线图

本文其余部分安排如下。第 2 节介绍 UPR 问题及其时间索引公式。本节还将介绍 DDD 的一般框架。第 3 节描述下界模型，并介绍一些附加性质，以确保下界模型确实能为 UPR 实例的最优值提供下界。第4节概述上界和增广步骤，这些步骤对于终止 DDD 算法或生成后续的部分时间扩展网络是必需的。第5节通过一组随机生成的测试实例展示 DDD 算法的有效性。我们还将考察实验结果，并观察给定输入参数时 DDD 算法有效性的变化趋势。

附录A展示了我们对存储界进行的强化松弛如何显著影响 DDD 终止前的迭代次数。附录B概述了第 3 节提出的松弛存储界如何推广 Lagos 等人 [21] 提出的存储松弛。最后，附录C讨论了两阶段 DDD 方法在地理实例上的表现。

2 问题陈述与背景

在定义 数据包路由问题之前，我们先介绍时间扩展网络的概念。

2.1 时间扩展网络

设 $D = (N, A)$ 为一个有向图，其弧标有运输时间 τ，并令 T 为所考虑的时间范围。相应的（完全）时间扩展网络 $D_T = (N_T, A_T)$ 为每个节点 $v \in N$ 在每个时间点 $t \in [T] := \{0, 1, \ldots, T\}$ 复制一个副本，并为每个出发时间复制每条弧。具体而言，$N_T = \{(v,t) : v \in N, t \in [T]\}$，$A_T = \{((v,t), (w, t + \tau_{vw})) : (v,t) \in N_T, vw \in A, t + \tau_{vw} \leq T\}$。例如，给定图 1 中的有向图以及时间范围 $T = 3$，我们得到图 2 所示的时间扩展

网络。当允许在节点处存储时，我们还添加滞留弧 $H_T = \{((v,t),(v,t+1)) : v \in V, t \in [T-1]\}$，并记作 $D_T = (N_T, A_T \cup H_T)$。我们将 A_T 中的弧称为移动弧。相对于 D 和 T 的部分时间扩展网络，是任意满足以下条件的有向图 $D_S = (N_S, A_S \cup H_S)$：$N_S \subseteq N_T$，$A_S \subseteq \{((v,t),(w,t')) : (v,t) \in N_T, vw \in A, t' \leq t + \tau_{vw} \leq T\}$，且 H_S 连接 N_S 中每个节点副本与其下一个副本。

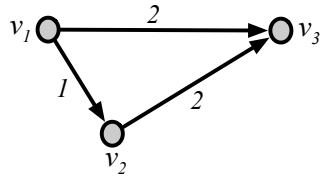

图 1：基础有向图。

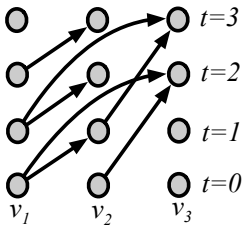

图2：对应的时间扩展图。

使用时间扩展网络对时序问题建模的优点在于，我们可以将其重新表述为静态问题，并应用求解静态网络问题的技术。然而，这种时间扩展通常会使所得的静态问题难以实际求解，因为网络规模会随 T [12, 13] 线性增长。在 DDD 中，我们通过构造部分时间扩展网络来缩小时间扩展网络的规模，这些网络会适当地低估 A_T 中弧的运输时间。本文中，$D_T = (N_T, A_T \cup H_T)$ 表示时间范围为 T 的完全时间扩展网络，而 $D_S = (N_S, A_S \cup H_S)$ 表示任意部分时间扩展网络。我们用 D_∞ 表示 $T = \infty$ 时的无限完全时间扩展网络。请注意，D_∞ 这一抽象概念仅用于定义数据包路由问题。为清晰起见，从现在起，我们将时间扩展网络中的弧和节点分别称为定时弧和定时节点。我们将带有相应时间的定时节点写作 (v,t)，其在基础图中的对应节点为 v。同样，我们将定时弧写作 $e = ((v,t),(w,t'))$，其在基础图中的对应弧为 $a = vw$。此外，我们将时间扩展网络中的路径称为轨迹。

2.2 通用分组路由

设 $D = (N, A)$ 为一个有向图，我们称之为扁平网络或基础网络。每条弧 $a \in A$ 都具有一个关联的运输时间 $\tau_a \in \mathbb{N}$，以及容量 $u_a \in \mathbb{N}$，表示可同时沿弧 a 离开的数据包最大数量。设 K 为数据包集合；对于每个数据包 $k \in K$，令 s_k 和 t{v33>分别表示其对应的起点和目的地。若数据包既不位于起点，也不位于目的地，我们称其为活动数据包。此外，每个节点 $v \in N$ 的存储水平为 $b_v \in \mathbb{N}$，这意味着它在任意时刻最多可存储 b_v 个活动数据包。调度的完工时间是任一数据包到达其目的地的最晚时间。通用分组路由（UPR）的目标是为每个数据包指定一条沿 D_∞ 行进的轨迹，在满足弧容量和节点存储约束的同时，使调度的完工时间最短。我们用 T^* 表示最小完工时间。

我们指出，本文提出的技术也适用于数据包大小任意的情形，以及数据包具有不同释放时间的情形。本文中我们有时会将

数据包称为商品流。

在实践中求解该问题的挑战

该问题可以通过连续变量来刻画时间；然而，为了跟踪两个数据包何时在同一条弧或同一个节点相遇，需要使用“大 M”约束。因此，这些数学规划 formulation 的线性规划松弛较弱，而且从历史经验来看，用于时间网络设计问题的连续形式在实践中的表现一直不佳 [2]。因此，自然的策略是将这些时间相关问题编码为时间扩展图上的静态问题，因为这类数学规划 formulation 的松弛较强。

然而，要将 UPR 的一个实例映射到有限时间扩展网络，我们需要知道最小完工时间的某个上界 T，T^*。遗憾的是，UPR 的近似难度至少与顶点着色一样高 [3]。因此，在实际应用中，我们不得不使用相对较大的 T 值作为上界，以确保能够在相应的完全时间扩展网络 D_T 中找到解。我们在第5节中证明，这会大幅增加求解时间，部分原因是随着T 增大，相应的混合整数规划中会引入额外的对称性。求解实际应用中产生的 UPR 实例时，我们或许过去求解过类似实例，因此知道 T 的一个较好上界 T^*，其中 $T = \alpha T^*$，且 α 的值较小。相反，如果我们对该实例一无所知，那么所能给出的上界很可能需要取较大的值 α。因此，在计算实验中，我们针对上界取 T^*,$1.5T^*$ 和 $2T^*$ 的情况，对所提出的 DDD 算法进行测试，以了解已知上界的紧度不同时，DDD 相较于传统混合整数规划的表现。

完全时间扩展网络的整数规划

对于每个 $v \in N$，令 K_v 表示在 v 处活跃 的商品集合。也就是说，$\mathrm{K}_v = \{k \in \mathrm{K} : s_k \neq v, t_k \neq v\}$。请注意，$\mathrm{K} \setminus \mathrm{K}_v$ 中的商品不会对节点 v 的存储水平产生影响，因为它们要么位于起点，要么位于目的地。我们强调，K_v 不 随时间变化。

令 T 为 T^* 的一个上界，并假设该上界已知。我们希望确定 D_T 中的轨迹，使完工时间最短，同时满足容量限制。首先请注意，由于所有输入数据均为整数，最优解的决策时间属于 $[\mathrm{T}]$。对于每个数据包 $\mathrm{k} \in \mathrm{K}$ 以及每条定时弧 $\mathrm{e} \in \mathrm{A}_T \cup \mathrm{H}$，我们设置一个二元变量 x_e^k；若数据包 k 按其在 D_T中的指定轨迹沿定时弧 e 行进，则该变量等于 1。我们直接根据 D 中的弧容量和节点容量，指定 D_T 中的定时弧容量。具体而言，对于每条定时弧 $\mathrm{e} = ((\mathrm{v},\mathrm{t}), (\mathrm{w},\mathrm{t}')) \in \mathrm{A}_T$，定义 $\mathrm{u}_e := \mathrm{u}_{vw}$。类似地，对于 $\mathrm{e} = ((\mathrm{v},\mathrm{t})(\mathrm{v},\mathrm{t}')) \in \mathrm{H}_T$，定义 $\mathrm{b}_e := \mathrm{b}_v$。

令 $\delta^+\mathrm{D}_T(\mathrm{v},\mathrm{t})$ 和 $\delta^-\mathrm{D}_T(\mathrm{v},\mathrm{t})$ 分别表示 D_T 中 (v,t) 处的出向和入向定时弧。也就是说，$\delta^+_{D_T}(\mathrm{v},\mathrm{t}) = \{\mathrm{e} \in \mathrm{A}_T \cup \mathrm{H}_T : \mathrm{e} = ((\mathrm{v},\mathrm{t}),(\mathrm{w},\mathrm{t}'))\}$，而 $\delta^-\mathrm{D}_T(\mathrm{v},\mathrm{t}) = \{\mathrm{e} \in \mathrm{A}_T \cup \mathrm{H}_T : \mathrm{e} = ((\mathrm{w},\mathrm{t}'),(\mathrm{v},\mathrm{t}))\}$。以下整数规划模型

给定上界 T 时的 UPR。

$$\min \quad \bar{T} \tag{UPR(D_T)}$$

$$\text{s.t.} \quad t' \cdot x_e^k \le \bar{T} \quad \forall k \in \mathcal{K},\ \forall e = ((v,t),(w,t')) \in A_T \tag{1}$$

$$\sum_{e \in \delta^+_{D_T}(v,t)} x_e^k - \sum_{e \in \delta^-_{D_T}(v,t)} x_e^k = \begin{cases} 1 & (v,t) = (s_k, 0) \\ -1 & (v,t) = (t_k, T) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_T \tag{2}$$

$$\sum_{k \in \mathcal{K}} x_e^k \le u_e \quad \forall e \in A_T \tag{3}$$

$$\sum_{k \in \mathcal{K}_v} x_e^k \le b_e \quad \forall\ e \in H_T \tag{4}$$

$$x_e^k \in \{0,1\} \quad \forall k \in \mathcal{K},\ \forall e \in A_T \cup H_T. \tag{5}$$

约束 (1) 表示时间范围等于所有数据包中的最晚到达时间。约束 (2) 确保轨迹集合满足流守恒。最后，约束 (3) 和 (4) 分别确保轨迹满足弧容量约束和节点存储约束。

此外，我们将加入以下约束，以强化线性规划松弛。在应用附录 C 中提出的两阶段 DDD 方法时，这一点至关重要。对于每个 $k \in$K，令 $A_{final\,T,k} = \{((v,t),(w,t')) \in A_T : w = t_k\}$ 为进入数据包 k 目的地的定时移动弧集合。由于每个数据包在可行整数解中只有一条轨迹，因此有以下约束。

$$\sum_{e=((v,t),(w,t')) \in A^{final}_{T,k}} t' \cdot x_e^k \le \bar{T} \quad \forall k \in \mathcal{K}. \tag{6}$$

2.3 DDD 框架

一般而言，DDD 算法旨在求解定义在某个完全时间扩展网络 D_T 上的最小化问题 P，其对应的混合整数规划（MIP）为 IP(D_T)。在 DDD 中，我们构造并逐步细化部分时间扩展网络，记作 D_S，使其相较于完整的时间扩展网络更加稀疏。此外，每个部分网络的构造方式确保了由 D_S 导出的混合整数规划（记作 IP(D_S)）的最优解能够为 IP(D_T) 的最优值提供下界。若满足这一关系，我们称 D_S 是 D_T 的一种松弛。

在部分时间扩展网络中，会对弧通行时间进行低估，使得D_T 中的每条轨迹都能映射到D_S 中的一条轨迹，且底层路径相同，不过定时弧会缩短。每次迭代中，我们求解D_S，并检查所找到的D_S 的解能否映射为D_T 中成本相同的解。由于D_S 中的弧并非都具有实际长度，因此可能无法在D_T中获得对应的解。如果无法在D_T中获得对应的解，我们就通过延长较短的定时弧，并向部分时间扩展网络中添加定时节点和定时弧来进行细化

D_S。Boland 等人 [1] 提出了以下关键性质，以确保D_S 是一个松弛问题；原问题的时间范围为T，且每个商品$k \in$ K的释放时间为r_k，截止时间为d_k。

标准 DDD 性质：

(P1) 对于所有商品 k $\in$ K，节点 (s_k,r_k) 和 (t_k,d_k) 均属于 N_S。（P2）每条弧 ((v,t),(w, $\bar{t}$)) $\in A_S$ 的 $\bar{t} \leq t + \tau_{vw}$。（P3）对于平面网络中的每条弧 a = (v,w) $\in$ A，以及部分时间扩展网络 $D_S = (N_S, A_S \cup H_S)$ 中满足 $t + \tau_{vw} \leq T$ 的每个节点 (v,t)，A_S 中都有一条从 (v,t) 开始的 a 的定时副本。（P4）如果弧 ((v,t),(w,t′)) $\in A_S$，则 N_S 中不存在节点 (w,t″)，且 $t' < t'' \leq t + \tau_{vw}$。

对于通用分组路由，我们假设给定了最小所需完工时间的一个上界 T，并且对于所有 $k \in$ K，设置 $r_k = 0$ 和 $d_k = T$。请注意，在通用分组路由情况下，由于弧容量和节点容量约束，这些性质不足以使 D_S 成为 D_T 的下界模型。例如，如果将 D_S 中的定时弧全部设置为与其底层弧相同的容量，那么 D_S 的总弧容量将小于 D_T。我们将在第 3.1 节和第 3.2 节中描述必要的附加性质。我们指出，本文提出的技术具有很强的通用性，可以轻松扩展到 r_k 和 d_k 会发生变化的情形，以及商品具有非单位需求量、且单个商品的流可以以分数形式沿多条轨迹发送的情形。

从D_T 到D_S的映射

要证明D_S 是D_T的松弛，我们需要将 IP(D_T) 的一个解映射为 IP(D_S) 的一个解，且成本不增加。以下映射 $\mu : A_T \to A_S$ 是 DDD 文献中证明下界模型是松弛时使用的标准工具。对于任意定时弧$e \in A_T$，μ 将弧e 上的流映射到A_S中的弧 $\mu(e)$。具体而言，

$$e = ((v,\bar{t}),(w,\bar{t}')) \quad \to \quad \mu(e) = ((v,\hat{t}),(w,\hat{t}')), \tag{7}$$

其中 $\hat{t} = \max\{t : t \leq \bar{t}, (v,t) \in N_S\}$，且 $\hat{t}' = \max\{t : t \leq \hat{t} + \tau_e, (v,t) \in N_S\}$。请注意，$\hat{t}'$ 取决于 $\hat{t}$，而非 $\bar{t}'$。对于 UPR，我们用 UPR(D_T) 和 UPR(D_S)（见第 3 节）分别替代 IP(D_T) 和 IP(D_S)。

2.4 通用分组路由的 DDD 方法概述

如上所述，DDD 方法的关键组成部分包括下界模型、上界/终止步骤以及增广/细化步骤。在第 3 节中，我们介绍另外两个性质，以确保当底层问题是通用分组路由时，D_S 是D_T 的松弛。我们的结果使 DDD 能更有效地求解带存储约束的流问题，因为改进后的节点存储松弛减少了某些实例中 DDD 终止前所需的迭代次数（见附录A）。在第4节中，我们将介绍上界模型和增广步骤。随后，我们将在第5节中展示计算结果。

概述草　　通用分组路由的 DDD算法如下。我们给出精确的　　e

第5节末尾的算法。子程序以及 UPR(D_S) 的定义见第4节和第5节。

算法1 求解 UPR-DDD(D，K，T)

输入：基础网络 $D=(N, A)$、数据包集合 K，以及最优完工时间的上界 T

1：创建满足下界性质的初始部分时间扩展网络D_S
2：**当** 未求解**时执行**
3: 求解 UPR(D_S)
4: 判断该解能否转换为 UPR(D_T) 的一个解，且不增加时间范围
5: **如果** 部分时间扩展网络的解可以转换**那么**
6: 停止。已找到 UPR(D_T) 的一个最优解。
7: 通过修正部分解支撑集中的至少一条弧，扩充当前定时节点集合N_S
该解要么过短，要么超出了弧容量或存储容量

3 下界模型

DDD迭代方法的关键组成部分之一是下界模型。具体而言，给定适当的定时节点子集 N_S，我们希望获得一个部分时间扩展网络 $D_S=(N_S, A_S \cup H_S)$，以及一个数学规划 formulation UPR(D_S)，其最优值不超过 UPR(D_T) 的最优值。按照 DDD 中的标准做法，我们依据 (P1)–(P4) 构造A_S 。弧的具体选择在本节末尾的算法 3中给出。现在，我们陈述与部分时间扩展网络D_S相对应的下界数学规划 formulation，并在第 3.1 节和第 3.2 节中定义 u' 和 b' 。

$$\min \quad \bar{T} \tag{UPR(D_S)}$$

$$\text{s.t.} \quad (t+\tau_{vw})\cdot x_e^k \le \bar{T} \quad \forall k \in \mathcal{K},\ \forall e=((v,t),(w,t')) \in A_S \tag{8}$$

$$\sum_{e=((v,t),(w,t'))\in A_{\text{first},k}} (t+\tau_{vw})\cdot x_e^k \le \bar{T} \quad \forall k \in \mathcal{K}. \tag{9}$$

$$\sum_{e\in\delta^+_{D_S}(v,t)} x_e^k - \sum_{e\in\delta^-_{D_S}(v,t)} x_e^k = \begin{cases} 1 & (v,t)=(s_k,0) \\ -1 & (v,t)=(t_k,T) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_S \tag{10}$$

$$\sum_{k\in\mathcal{K}} x_e^k \le u'_e \quad \forall e \in A_S \tag{11}$$

$$\sum_{k\in\mathcal{K}_v} x_e^k \le b'_e \quad \forall e \in H_S \tag{12}$$

$$x_e^k \in \{0,1\} \quad \forall k \in \mathcal{K}, e \in A_S \cup H_S. \tag{13}$$

除了修改弧容量和节点容量外，我们还将 UPR(D_T) 中的约束 (1) 和 (6) 替换为 (8) 和 (9)。注意，在部分时间扩展网络中，对于某些 $((v,t),(w,t')) \in A_S$，可能存在 $t' < t + \tau_{vw}$。因此，约束 (8) 和 (9) 比约束 (1) 和 (6) 更严格。在第4节 4.2 中，我们将证明，只要对于所有 $a \in A$ 都有 $\tau_a \ge 1$，在整个 DDD 过程中，我们永远不会添加

具有 $t > T^*$ 的时间节点 (v,t)。

我们需要为D_S 中的弧容量和节点存储量进行分配，以确保 UPR(D_S) 是 UPR(D_T) 的松弛。按照惯例，为证明 UPR(D_S) 的最优值至多为 UPR(D_T) 的最优值，我们使用映射 μ（如第2.3节所定义），将 UPR(D_T) 中完工时间为 $\bar{T}$ 的可行解映射为 UPR(D_S) 中完工时间至多为 T 的解。

回顾式（7），对于任意定时弧 e $=((v,\bar{t}),(w,\bar{t}')) \in A_T$，$\mu(e)=((v,\hat{t}),(w,\hat{t}'))$，其中 $\hat{t} = \max\{t : t \le \bar{t}, (v,t) \in N_S\}$，且 $\hat{t}' = \max\{t : t \le \hat{t} + \tau_{vw}, (v,t) \in N_S\}$。考虑到这一映射 μ，我们将说明如何定义 u′ 和 b′，使得 UPR(D_S) 是 UPR(D_T) 的松弛。令 $\bar{x}$ 为 UPR(D_T) 的一个可行解。我们将 $\hat{x}$ 定义为如下二进制向量：对于所有 e $\in A_S$ 和 k $\in$ K,

$$\hat{x}_e^k = \max\{\bar{x}_f^k : \mu(f) = e\}.$$

也就是说，对于所有 $e \in A_S$ 和 $k \in$ K，若对于任意满足 $\bar{x}_f^k = 1$ 的定时弧 $f \in A_T$，都有 $\hat{x}_e^k = 1$ if $\mu(f) = e$。在本文中，我们滥用记号，将以这种方式从 $\bar{x}$ 得到的 $\hat{x}$ 记为 $\mu(\bar{x})$。

在 Boland 等人首次提出 DDD 方法的开创性工作中，作者针对具有流守恒约束的问题证明了以下引理（[1] 中的定理 2）。

引理 3.1。若 $\bar{x}$ 是满足 *UPR*(D_T) 中的流约束和整数性约束（约束 *(2)* 和 *(5)*）的向量，则 $\hat{x} = \mu(\bar{x})$ 满足 *UPR*(D_S) 中相应的约束（*(10)* 和 *(13)*）。

3.1 弧容量

将弧容量纳入 DDD 框架的技术已在 [21] 和 [29] 中提出。为求清晰，并便于在第 3.2 节的新颖工作中使用，我们在本节完整介绍这项工作。

在图 3 所示的示例中，我们假设弧容量均为单位容量。注意，μ 会将两条$u \to v \to w$ 轨迹（以蓝色虚线表示）都映射到D_S中的同一条轨迹，这会超过原有的单位弧容量。因此，有必要在性质 (P1)–(P4) 的基础上添加条件，以便为弧容量问题提供一个下界。

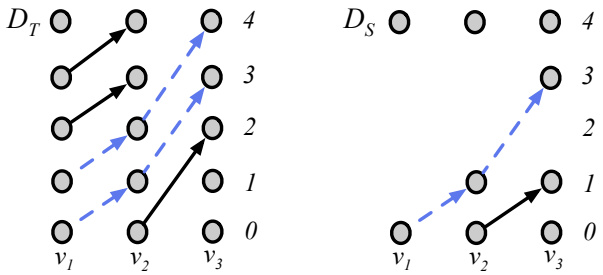

图 3：D_T 中的两条轨迹通过 μ映射到D_S 中的同一条轨迹。

对于每个时间节点（v,t）$\in N_S$，令 $n_S(v,t)$ 表示v 在N_S中t 之后首次出现的时间。也就是说，

$$n_S(v,t) = \min\{t' : t' > t, (v,t') \in N_S\}.$$

令 $e = ((v,t),(w,t')) \in A_S$。那么，当 $t_1 \in \{t, t+1, \ldots, n_S(v,t)-1\}$ 时，μ映射 $f = ((v,t_1),(w,t_2)) \in A_T$ 到 e。为刻画此区间的长度，我们定义

$$m_S(v,t) := n_S(v,t) - t,$$

它表示 v 在 N_S 中下一次出现之前的时间单位数。注意，对于任意 $(v,t) \in N_T$，都有 $n_T(v,t) = t+1$，且 $m_T(v,t) = 1$。请注意，稍后我们将用到这样一个事实：即使 $(v,t) \notin N_S$，$m_S(v,t)$ 仍有定义。对于任意定时弧 $((v,t),(w,t')) \in A_S$，A_T 中有 $m_S(v,t)$ 条定时弧根据映射 μ 映射到 $((v,t),(w,t'))$。这证明了下列性质 (P^{arcs}) 足以使 $\hat{x}$ 满足 UPR(D_S) 的弧容量约束。

$$(P^{arcs})\ \text{For any arc } e = ((v,t),(w,t')) \in A_S,\ u'_e = u_e \cdot m_S(v,t)$$

具体而言，我们证明了以下引理。为使论证完整，我们给出一个简短的形式化证明。

引理 3.2. 设 D_S 是一个满足性质 $(P1)-(P4)$ 和 (P^{arcs}) 的部分网络，且 $\bar{x}$ 是 $UPR(D_T)$ 的一个解。那么，$\hat{x} = \mu(\bar{x})$ 满足约束 ***(11)***。

证明。设 $e = ((v,t),(w,t')) \in A_S$，并考虑 $\sum_{k\in\mathcal{K}} \hat{x}^k_e$。如果对于某个商品 $k \in K$，有 $\hat{x}^k_e = 1$，那么对于一条定时弧 $a = ((v,\bar{t}),(w,\bar{t}+\tau_{vw})) \in A_T$，有 $\bar{x}^k_a = 1$，且 $\mu(a) = e$。根据映射 μ 的定义，我们知道 $\bar{t} \in \{t, t+1, \cdots, n_S(v,t)-1\}$。因此，集合 $\{a \in A_T : \mu(a) = e\}$ 的大小为 $m_S(v,t)$。因此，

$$\sum_{k\in\mathcal{K}} \hat{x}^k_e \le \sum_{\substack{a\in A_T:\\ \mu(a)=e}} \sum_{k\in\mathcal{K}} \bar{x}^k_a \le u_a \cdot m_S(v,t),$$

其中，最后一个不等式成立，是因为 $\bar{x}$ 是 UPR(D_T) 的可行解。由于$u_e = u_a$，因此结论成立。

□

3.2 存储限制

首先，我们介绍通用分组路由中的存储约束与 Lagos 等人在求解连续时间库存路径问题（CIR）时所处理的存储约束之间的差异。 [21] 在 CIR 问题中，一家公司管理其客户的库存，并从单一设施配送产品。Lagos 等人增加了一项限制，即在任意时间点，一个客户位置只能有一辆车。本质上，这相当于对客户停车场的存储容量施加硬性限制。Lagos 等人仅研究星形图（“往返”路线），这简化了解空间，使得平面网络中的每个客户节点只有一条入弧。为了在给定部分时间扩展网络的情况下获得 CIR 实例的下界，他们“放宽”一部分存储

约束。对于某个时间节点（v，t）上的每个被放宽的存储约束，作者都会移除该存储约束（车辆限制），从而在该次迭代中实际上允许（v，t）处拥有无限存储容量。

然而，我们的研究表明，在离散设置中，我们可以对存储容量采用更保守的松弛。在附录A中，我们证明，将 Lagos 等人的方法应用于离散问题会导致 $\Omega(T)$ 次迭代，而采用我们更紧的松弛则可使 DDD 在一次迭代中终止。此外，我们的研究处理的是更复杂的平坦网络，其中允许任意路线，而不仅仅是往返路线。这一点具有重要意义，因为目前尚不清楚 Lagos 等人的方法在往返路线框架之外是否也能表现良好。

与弧容量的情况类似，我们不能简单地将基础图中的节点容量赋给 N_S 中的节点。本节将说明如何为每条定时弧 $e \in H_S$ 赋予滞留弧存储量 b'_e，以确保 UPR(D_S) 是 UPR(D_T) 的松弛。

回顾 $\mathrm{m_S}(v,t)$ 的定义，它表示在 N_S 中，时间t 之后 v 首次出现前的时间单位数。请注意，即使 $(v,t) \notin N_S$，$\mathrm{m_S}(v,t)$ 仍有明确定义。接下来，我们考察 D_S 中时间节点的邻居。对于 $(v,t) \in N_S$，令 $N_S^-(v,t)$ 表示 D_S 中 (v,t) 的入邻居。具体而言，

$$N_S^-(v,t) = \{(w,t') : \exists\, e = ((w,t'),(v,t)) \in A_S\}.$$

设 $\bar{x}$ 为 UPR(D_T) 的一个可行解，并令 $\hat{x}$ 为根据本节开头所述的映射 $\mu : A_T \to A_S$ 得到的向量。我们希望了解映射 μ 会如何影响某个时间节点 $(v,t) \in N_S$所需的存储量。

对于所有 $k \in K$，令 Q_k 为数据包 k 根据 $\bar{x}$ 在 D_T 中经过的轨迹。Q_k 由一组有序的移动带时间弧 $\{e_1^k, e_2^k, \ldots, e^k{}_{l_k}\}$ 以及额外的滞留弧组成。令 P_k 为底层图 D 中对应的路径。我们定义 $\bar{t}_u^{k\,out}$ 和 $\bar{t}_v^{k\,in}$，使得 $((u, \bar{t}_u^{k\,out},), (v, \bar{t}_v^{k\,in},)) = e_j^k$，并以类似方式定义 $\hat{t}_u^{k\,out}$ 和 $\hat{t}_v^{k\,in}$。也就是说，

$$e = ((u, \bar{t}_u^{k,out}), (v, \bar{t}_v^{k,in})) \quad \rightarrow \quad \mu(e) = ((u, \hat{t}_u^{k,out}), (v, \hat{t}_v^{k,in}))$$

考虑轨迹 Q_k 上 D_T 中连续的两条移动带时间弧 $e_{uv} = ((u, \bar{t}_u^{k\,out},), (v, \bar{t}_v^{k\,in},))$ 和 $e_{vw} = ((v, \bar{t}_v^{k\,out},), (w, \bar{t}_w^{k\,in},))$。$e_{uv}$ 和 e_{vw} 上的流分别映射为 $\mu(e_{uv}) = ((u, \hat{t}_u^{k\,out},), (v, \hat{t}_v^{k\,in},))$ 和 $\mu(e_{vw}) = ((v, \hat{t}_v^{k\,out},), (w, \hat{t}_w^{k\,in},))$。

$(v,\ t)$ 是 N_S 中 v 的一个定时副本。以下显而易见的事实将用于理解映射 μ 对数据包k 在 $(v,\ t)$ 处存储的影响。

(F1) $\hat{t}^{k,out}(l) = \max\{t : t \le \bar{t}^{k,out}(l), (l,t) \in N_S\}$，其中 $l \in \{u, v\}$；(F2) $\hat{t}_v^{k\,in} = \max\{t : t \le \hat{t}_u^{k\,out} + \tau_{uv}, (v,t) \in N_S\}$；(F3) $\hat{t}_v^{k\,in} \le \bar{t}_v^{k\,in}$，且 $\hat{t}_v^{k\,out} \le \bar{t}_v^{k\,out}$。

（F1）根据 μ（方程（7））的定义得出，而（F2）则由（F1）以及以下事实得出：在 A_S 中，存在一条从（u，$\hat{t}_u^{k\,out}$，）出发且长度尽可能长的 uv 定时弧（性质（P2）和（P4））。（F3）由此得出

由（F1）和（F2），以及以下事实得出：$\bar{t}_v^{k\,in} = \bar{\cdot}, t_u^{k\,out} + \tau_{uv}, $。

如果根据 $\bar{x}$ ($\bar{t}_v^{k\,in} \leq, t < \bar{t}_v^{k\,out}, $)，数据包 k 先前已存储在 (v,t)，那么 μ 不能新增数据包 k 的存储。因此，假设数据包 k 在 $\bar{x}$ 中未存储于 (v,t)。如果 $\bar{t}_v^{k\,in} \leq$, t，那么由于该数据包未存储于 (v,t)，我们也有 $\bar{t}_v^{k\,out} \leq$, t。在这种情况下，根据事实 (F3) 可知 $\hat{t}_v^{k\,out} \leq \bar{}, t_v^{k\,out} \leq$, t，因此数据包 k 在 $\hat{x}$ 中未存储于 (v,t)。不过，如果 $\bar{t}_v^{k\,in} >$, t，且根据 $\hat{x}$，数据包 k 到达 v 的时间早于其按照 $\bar{x}$ 的调度到达时间，那么数据包 k 在 (v,t) 的存储量可能会增加。也就是说，$\hat{t}_v^{k\,in} < \bar{}, t_v^{k\,in}, $。这可能发生在以下任一情形：

1. 流在同一时间从 u 出发（$\hat{t}_u^{k\,out} = \bar{}$，$t_u^{k\,out}$，）， 但（v $\bar{t}_v^{k\,in}$，） $\notin N_S$。例如，图5展示了一个部分时间扩展网络，其中 (u, 1) $\in N_S$，但 (v, 2) $\notin N_S$，因此 μ((u, 1), (v, 2)) = ((u, 1), (v, 1))；
2. 流被迫提前离开前一个节点（$\hat{t}_u^{k\,out} < \bar{}$，$t_u^{k\,out}$，）。例如，图6展示了一个部分时间扩展网络，其中 (u, 1) $\notin N_S$，因此流必须提前离开 u，且 μ((u, 1), (v, 2)) =((u, 0), (v, 1))。

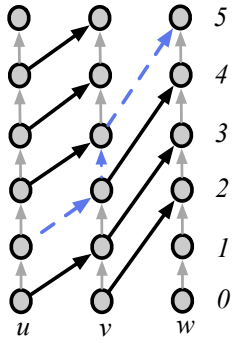

图4：原始轨迹

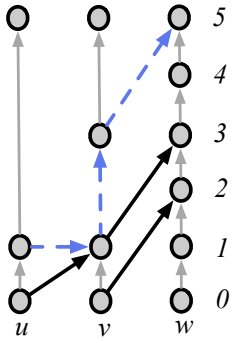

图5：场景1

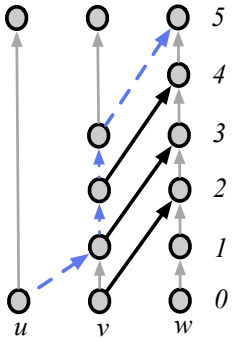

图6：场景2

令 K(v) 表示根据 $\bar{x}$，沿着包含 v 的一个定时副本的轨迹行进的数据包集合。令 $K_{\hat{x}}^1$(v) 表示 K(v) 中在 $\hat{x}$ 和 $\bar{x}$ 中均于同一时间离开 v 前一节点的数据包集合。类似地，令 $K_{\hat{x}}^2$(v) 表示 K(v) 中在 $\hat{x}$ 中比在 $\bar{x}$ 中更早离开前一节点的数据包集合。注意，K(v) = $K_{\hat{x}}^1$(v) $\cup K_{\hat{x}}^2$(v)。令 e = ((v,t),(v,t′)) 为 H_S 中离开 (v,t) 的定时弧，令 f = ((v,t), (v,t + 1)) 为 H_T 中离开 (v,t) 的定时弧。

引理3.3。 若 D_S 满足性质 $(P1)-(P4)$，则任何数据包 $k \in \mathcal{K}_{\hat{x}}^1(v)$，若其在 $\bar{x}$ 中未存储于 v，则在 $\hat{x} = \mu(\bar{x})$ 中也不会存储于 v。

证明。假设数据包 k $\in K_{\hat{x}}^1$(v) 未存储在 $\bar{x}$ 中的节点 v 处。也就是说，$\bar{t}_v^{k\,in} = \bar{}$，$t_v^{k\,out}$，。令 u 为路径 P_k in $\bar{x}$ 上的前一个节点。由于 k $\in K_{\hat{x}}^1$(v)，我们知道 $\hat{t}_u^{k\,out} = \bar{}$，$t_u^{k\,out}$，。因此，在 D_S 中，数据包 k 沿着某条弧 ((u $\bar{t}_u^{k\,out}$，),(v, $\hat{t}_v^{k\,in}$，)) 传输。根据事实（F2），$\hat{t}_v^{k\,in} =$，max{t :t $\leq \hat{t}_u^{k\,out} + \tau_{uv}$，,(v,t) $\in N_S$}。类似地，根据事实（F1），$\hat{t}_v^{k\,out} =$，max{t :t $\leq \bar{t}_v^{k\,out}$，,(v,t) $\in N_S$}。由于 $\bar{t}_v^{k\,out} = \bar{}$，$t_u^{k\,out} + \tau_{uv}$，且 $\hat{t}_u^{k\,out} = \bar{}$，$t_u^{k\,out}$，我们可知 $\hat{t}_v^{k\,in} =$，$\hat{t}_v^{k\,out}$，。因此，节点 v 处没有新增数据包 k 的存储，且 $\hat{x}_e^k = \bar{x}_f^k = 0$。 □

我们首先在引理 3.4 中给出一个简单界限，说明在 (v, t) 处容纳 $K_{\hat{x}}^1(v)$ 中的数据包所需的额外存储量。随后，我们在引理 3.5 中进一步收紧这一论证。在附录A中，我们将说明，这种收紧

可能对某些问题实例中 DDD 的有效性至关重要。

引理 3.4。 如果 D_S 满足性质 $(P1)-(P4)$，那么

$$\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \le \sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \bar{x}^k_f + (\mathrm{m_S}(v,t)-1)b_v.$$

证明。假设数据包 k 在 $\bar{x}$ ($\bar{t}^{k\ in}_v < \bar{,}\ t^{k\ out}_v,$) 中的 v 处被存储。我们知道，如图6所示，如果 $\hat{t}^{k\ in}_v \le$, $t < \bar{t}^{k\ in}_v,$，则可以在 (v,t) 引入存储。由于数据包 k 在 $\bar{x}$ 和 $\hat{x}$ 中于同一时间离开前一个节点，因此可得 $\bar{t}^{k\ in}_v \in$, $\{t+1, t+2, \cdots, n_S(v,t)-1\}$。因此，$K^1_{\hat{x}}(v)$ 中每个在 (v,t) 引入存储的数据包，都必须先前在该区间内的某个时刻存储于 v 处，而该区间的长度为 $m_S(v,t)-1$。因此，$\sum_{k\in\mathcal{K}^1_{\hat{x}}}(v)\ \hat{x}^k_e \le \sum_{k\in\mathcal{K}^1_{\hat{x}}}(v)\ \bar{x}^k_f + (m_S(v,t)-1)b_v$。 □

在下面的引理中，我们会看到，这个界可以显著收紧。收紧这一界所依据的论证是：如果 $K^1_{\hat{x}}(v)$ 中的两个数据包 j 和 k 在 $\bar{x}$ 中的同一时间未存储于 v，那么在 $\hat{x}$ 中也是如此。

引理 3.5。 若 D_S 满足性质 $(P1)-(P4)$，则

$$\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \le b_v.$$

此外，若 $n_S(v,t) = t+1$，则

$$\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \le \sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \bar{x}^k_f.$$

证明。 假设数据包 k 在 $\bar{x}$ 中的 ($\bar{t}^{k\ in}_v < \bar{,}\ t^{k\ out}_v,$) 时刻存储于 v。同样，如果 $\hat{t}^{k\ in}_v < \bar{,}\ t^{k\ in}_v,$，也可能引入存储。不过，我们将证明：根据 $\bar{x}$，若 $K^1_{\hat{x}}(v)$ 中的商品流 k 和 j 未在同一时间存储于 v，那么对于 $\hat{x}$ 也是如此。这将证明 $\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \le b_v$，因为 $\bar{x}$ 是可行的。

不失一般性，我们可以假设 $\bar{t}^{k\ out}_v \le \bar{,}\ t^{j\ in}_v,$，如图 7 所示（$\bar{t}^{k\ out}_v = 3,$以及 $\bar{t}^{j\ in}_v = 3,$），因为必有一个数据包不晚于另一个数据包的到达时间离开节点 v。我们断言 $\hat{t}^{k\ out}_v \le, \hat{t}^{j\ in}_v,$。

假设 $\hat{t}^{j\ in}_v \ge \bar{,}\ t^{k\ out}_v,$。那么由于根据（F3）有 $\hat{t}^{k\ out}_v \le \bar{,}\ t^{k\ out}_v,$，可推出 $\hat{t}^{k\ out}_v \le, \hat{t}^{j\ in}_v$。或者，假设 $\hat{t}^{j\ in}_v < \bar{,}\ t^{k\ out}_v,$。根据（F1），

$$\hat{t}^{k,out}_v = \max\{t : t \le \bar{t}^{k,out}_v, (v,t) \in N_S\}.$$

令 u 为根据 $\bar{x}$，数据包 j 在 v 之前经过的节点。由 (F2) 且由于 $\hat{t}^{j\ out}_u = \bar{,}\ t^{j\ out}_u,$(S1),

$$\hat{t}^{j,in}_v = \max\{t : t \le \hat{t}^{j,out}_v + \tau_{uv} = \bar{t}^{j,in}_v, (v,t) \in N_S\}.$$

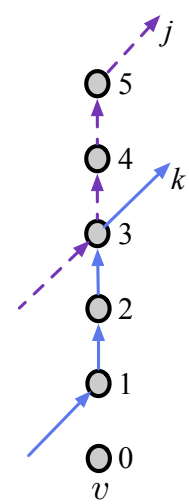

图 7：不相交的存储

由于 $\hat{t}_{\text{inv}} < \bar{t}_v^{k\,out}$，且 $\bar{t}_v^{k\,out} \le \bar{}$, t_{inv}，可得 $\hat{t}_v^{k\,out} =$, $\hat{t}_{\text{inv}} = \max\{t : t \le \bar{t}_v^{k\,out}$, , $(v,t) \in N_S\}$，如所需。这证明 $\hat{x}^1$ 在所有 $(v,t) \in N_S$ 处均满足存储约束。

最后，当 $(v,t+1) \in N_S$ 时，$K_{\hat{x}}^1(v)$ 中存储在 $\hat{x}$ 的 (v,t) 处的数据包集合，恰好就是存储在 $\bar{x}$ 的 (v,t) 处的数据包集合。因此，如果 $n_S(v,t) = t+1$，那么 $\sum_{k\in K_{\hat{x}}^1(v)} \hat{x}_e^k \le \sum_{k\in K_{\hat{x}}^1(v)} \bar{x}_f^k$。

□

现在我们对 $K_{\hat{x}}^2(v)$ 中数据包的存储量进行界定。要在时间 t 存储于 v，数据包必须在时间 t 之前到达 v。这给出了数据包从 D_S 中的前一节点离开的时间上界，由此可得数据包从 $\bar{x}$ 中前一节点离开的时间也有相应的上界。类似地，我们确定了 $K_{\hat{x}}^2(v)$ 中的数据包在 D_T 中从前一节点离开的时间下界：如果它未存储在 $\bar{x}$ 的 (v,t) 处，却存储在 $\hat{x}$ 的 (v,t) 处。

为便于记号表示，我们引入一个额外的定义。给定部分时间扩展网络 D_S 和完全时间扩展网络 D_T，对于每个 $e = ((v,t),(v,t')) \in H_S$，我们定义

$$U_e(D_S, D_T) := \sum_{(w,t')\in N_T^-(v,t)\cup N_S^-(v,t)} u_{wv} \cdot (n_S(w,t') - 1).$$

当 D_S 和 D_T 显而易见时，我们记作 U_e。

引理 3.6. 如果 D_S 满足性质 $(P1)$ –$(P4)$，那么

$$\sum_{k\in K_{\hat{x}}^2(v)} \hat{x}_e^k \le \sum_{k\in K_{\hat{x}}^2(v)} \bar{x}_f^k + U_e.$$

证明。令 $w \in N^-(v)$。我们知道，A_T 中 (w,v) 的多个定时弧会映射到 A_S 中的同一个定时弧 $((w,t_1),(v,t_2))$，如第 3.1 节所述，这可能会在 v 处引入额外的存储。在整个证明过程中，我们将参考图 8。缺失的节点用白色方块标出，灰色虚线弧表示如果 $(w, t-\tau_{wv})$ 位于当前部分时间扩展网络中（可能是，也可能不是），我们将得到的弧。

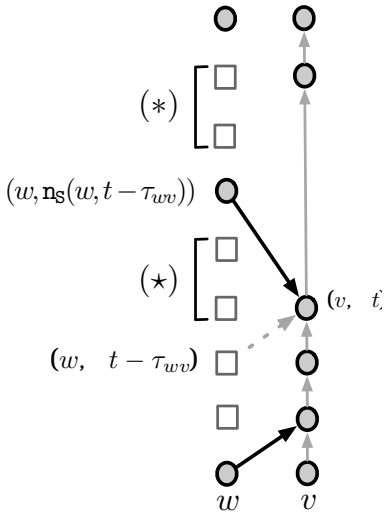

图 8

假设 k 是 $K_{\hat{x}}^2(v)$ 中的一个数据包，根据解 $\bar{x}$，w 是它到达 v 之前经过的节点。也就是说，数据包 k 沿着 D_T 中的定时弧 $((w\ \bar{t}_w^{k\,out},), (v\ \bar{t}_v^{k\,in},))$ 行进。首先，考虑 $\bar{t}_v^{k\,in} \le$, t 的情况。如果数据包 k 在解 $\bar{x}$ 中最初存储于 (v,t)，那么在解 $\hat{x}$ 中，数据包 k 在 (v,t) 所需的存储量不会超过解 $\bar{x}$ 中的存储量。反之，如果数据包 k 未存储在 v 处，则 $\bar{t}_v^{k\,out} \le$, t，因此 $\hat{t}_v^{k\,out} \le$, t 也成立。因此，数据包 k 不会存储在解 $\hat{x}$ 中的 (v,t) 处。

因此，只有当 $\bar{t}_v^{k\,in} > t$ 且 $\hat{t}_v^{k\,in} \le t$ 时，才可能在 $\hat{x}$ 中的 (v,t) 处为数据包 k 引入存储。第一个

条件等价于 $\bar{t}_w^{k\,out} > t - \tau_{wv}$。现在我们考虑第二个条件，即 $\hat{t}_v^{k\,in} \le t$。首先注意，如果 $\bar{t}_w^{k\,out} < \mathrm{n_S}(w, t-\tau_{wv})$，那么 $\hat{t}_v^{k\,in} \le t$。因此，可能出现这种情况：

$$\bar{t}_w^{k,out} \in \{t-\tau_{wv}+1, t-\tau_{wv}+2, \cdots, \mathrm{n_S}(w, t-\tau_{wv})-1\}.$$

该区间的长度为 $\mathrm{m_S}(w, t-\tau_{wv})-1$，我们在图 8 中用 $(\star)$ 标记这些出发时间。注意，$(w, t-\tau_{wv}) \in N_T^-(v, t)$。

或者，我们也可以有 $\bar{t}_w^{k\,out} \ge \mathrm{n_S}(w, t-\tau_{wv})$，并且如果 $\hat{t}_v^{k\,in} \le t$，则性质（P4）蕴含 $\hat{t}_v^{k\,in} = t$。也就是说，$\mu((w, \bar{t}_w^{k\,out}), (v, \bar{t}_v^{k\,in}))$ 是 D_S 中 (v,t) 处的一条入向定时弧。因此，$(w, \hat{t}_w^{k\,out}) \in N_S^-(v,t)$。该集合在图 8 中用 $(\star)$ 标记。因此，为容纳 $K_{\hat{x}}^2(v)$ 中的数据包，(v,t) 处所需的额外存储至多为

$$\sum_{(w,t') \in N_T^-(v,t) \cup N_S^-(v,t)} u_{wv} \cdot (\mathrm{m_S}(w,t') - 1) = U_e.$$

□

因此，我们引入以下性质。引理3.7的证明可由引理3.3、3.5和3.6以及以下事实推出：对于每个 $e \in H_S$, $\sum_{k \in \mathcal{K}_v} \hat{x}_e^k = \sum_{k \in \mathcal{K}_{\hat{x}}^1(v)} \hat{x}_e^k + \sum_{k \in \mathcal{K}_{\hat{x}}^2(v)} \hat{x}_e^k$。

(P^{storage})：对于任意 $e = ((v,t),(v,t')) \in H_S$,

$$b'_e \ge \begin{cases} b_v + U_e & \text{if } (v, t+1) \in N_S \\ 2b_v + U_e & \text{if } (v, t+1) \notin N_S \end{cases}$$

引理3.7。设 D_S 是满足性质 $(P1)$–$(P4)$ 和 (P^{storage}) 的部分时间扩展网络，并设 $\bar{x}$ 是 $UPR(D_T)$ 的一个解。那么 $\hat{x} = \mu(\bar{x})$ 满足约束（*12*）。

最后，我们证明以下定理：UPR(D_S) 确实是一个下界。

定理3.1。若 D_S 满足性质 $(P1)$–$(P4)$、(P^{arcs}) 和 (P^{storage})，则 $UPR(D_S)$ 的最优解的目标值不大于 $UPR(D_T)$ 的最优解的目标值。

证明。设 $\bar{x}$ 是 UPR(D_T) 的一个解，并令 $\hat{x} = \mu(\bar{x})$。根据引理 3.1，$\hat{x}$ 满足 UPR(D_S) 的流约束和整数性约束。根据引理 3.2 和引理3.7，$\hat{x}$ 满足 UPR(D_S) 的弧容量约束和存储容量约束。最后，由于 μ 将 D_T 中的轨迹映射为 D_S 中低估原始长度的轨迹，因此，UPR(D_S) 中 $\hat{x}$ 的目标值不大于 $\bar{x}$ 的目标值。 □

需要注意的是，当 $N_S = N_T$ 时，UPR(D_S) 等价于 UPR(D_T)。此外，随着我们向 N_S 添加定时节点，UPR(D_S) 的目标值单调不减。然而，我们最终希望在不必使用 $N_S = N_T$ 的情况下求解 UPR(D_T)。在接下来的章节中，我们将说明如何判断 UPR(D_S) 的一个解能否转换为 UPR(D_T) 的一个具有相同完工时间的解；若不能，则说明如何选择要添加至N_S的定时节点。

满足 $(P1)-(P4)$、(P^{arcs}) 和 (P^{storage}) 的最小定时节点集合，是所有时刻 0 和 T 的节点集合。我们以这组定时节点作为算法的起点。

算法 2 生成-初始-$N_S(D=(N,A),\mathrm{K},T)$

输入：基础网络 $D=(N,A)$、数据包集合 K，以及最优完工时间的上界 T

1: **对所有** $v\in N$ **执行**

2: 将定时节点 $(v,0)$ 和 (v,T) 添加到 N_S

3: **返回**N_S

在算法 3 中，我们将当前定时节点集合用作输入，并生成定时弧 A_S 和 H_S，以及容量 u' 和 b'，使得 $D_S=(N_S,A_S\cup H_S),u',b'$ 满足 $(P1)-(P4)$、(P^{arcs}) 和 (P^{storage})。我们定义这些容量，使其以等式形式满足 (P^{arcs}) 和 (P^{storage})。定时弧的构造在 DDD 文献中已是标准做法。

算法 3 生成 $A_S\cup H_S(N_S, D, T)$ 输入：基础网络 $D=(N, A)$，以及一组定时节点 N_S1：对所有 $(v,t)\in N_S$ 执行 2：$e\leftarrow((v,t),(v,n_S(v,t)))$,3：$U_e(D_S, D_T)\leftarrow\sum_{(w,t')\in N_T^-(v,t)\cup N_S^-(v,t)} u_{wv}\cdot(m_S(w,t')-1)$ 4：$b'_e:=\begin{cases} b_v+U_{v+e}^{2}\ \ {}_{S}\ \ {}_{T}\ \ ,\ +1\notin\ (D\ ,D\ )\ 若\ (v\ t\ +1)\ \in N\ b\ U\ (D\ ,D\ )\ 若\ (v\ t\ ) \end{cases}$

N_S5：若 $t<T$，则向 H_S 添加定时弧 e，其存储容量为 b'_e6：对所有 $vw\in A$ 执行 7：向 A_S 添加定时弧 $f=((v,t),(w,t'))$，其容量为 $u'_f=u_{vw}\cdot m_S(v,t)$，其中 t' 是满足 $(w,t')\in N_S$ 且 $t'\le t+\tau_{vw}$ 的最大值

8：**返回** A_S，H_S

4 上界模型与增强

4.1 上界模型

给定下界模型的一个解，我们希望确定它是否可以转换为 UPR(D_T) 的最优解。假设我们正在处理一个满足 (P1) - (P4)、(P^{arcs}) 和 (P^{storage}) 的部分时间扩展网络 $D_S=(N_S,\ A_S\cup H_S)$。令 $\hat{x}$ 为 UPR(D_S) 的一个最优解，其目标值为 $\hat{T}$。我们想知道 $\hat{x}$ 能否转换为 UPR(D_T) 的一个解，且具有相同的值（即完工时间）。

注意，$\hat{x}$ 为每个数据包指定了D_S 中的一条轨迹，每条轨迹都对应于底层静态图D中的一条路径。此外，给定一个候选完工时间$\hat{T}$。因此，如果我们在D_T 中求解 UPR，并附加限制条件，要求数据包沿着由$\hat{x}$指定的D 中的底层路径移动，就可以为 UPR(D_T)生成一个上界。具体而言，我们得到一个 UPR-FP 实例，其中对于每个$k\in$K，P_k是由$\hat{x}$ 为数据包k所确定的D 中的路径。令A_T^k和H_T^k 表示可用于以下情况的定时弧集合：

底层路径为P_k的轨迹。也就是说，对于每个$k \in$K,

$$A_T^k = \{((v,t),(w,t')) \in A_T : vw \in A(P_k)\} \qquad \text{and} \qquad H_T^k = \{((v,t),(v,t')) \in H_T : v \in N(P_k)\}.$$

尽管这个 UPR-FP 实例仍然比具有相同时间范围T的原始 UPR 实例更容易求解，但它仍然是 NP困难 [27]。因此，我们无法在每次迭代中获得上界，转而将时间范围限制为$T\prime = \lceil(1+\alpha)\,\hat{}T\rceil$，其中 $\alpha \geq 0$（我们的计算采用了 $\alpha = 0.01$）。限制时间范围使我们能够检测 $\hat{}x$ 是否可以转换为 UPR(D_T) 的一个解，且其值至多为$T\prime$。令$D_{T\prime} = (N_{T\prime}, A_{T\prime} \cup H_{T\prime})$为时间范围为$T'$的完全时间扩展网络。现在给出以下上界数学规划 formulation。注意，弧容量和节点存储量与原始 UPR(D_T) 实例中给定的数值一致。

$^{-}$分钟 T

$$(\text{UPR-FP}(\{D_{T\prime}^k\}_{k\in\mathcal{K}}, D_{T\prime}))$$

$$\text{s.t. } t' \cdot x_e^k \leq \bar{T} \quad \forall k \in \mathcal{K},\ \forall e = ((v,t),(w,t')) \in A_{T\prime}^k \tag{14}$$

$$\sum_{e=((v,t),(w,t'))\in A_{T\prime,k}^{k,final}} t' \cdot x_e^k \leq \bar{T} \quad \forall k \in \mathcal{K} \tag{15}$$

$$\sum_{e\in\delta^+_{D_{T\prime}^k}(v,t)} x_e^k - \sum_{e\in\delta^-_{D_{T\prime}^k}(v,t)} x_e^k = \begin{cases} 1 & (v,t) = (s_k, 0) \\ -1 & (v,t) = (t_k, T') \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_{T\prime} \tag{16}$$

$$\sum_{k\in\mathcal{K}} x_e^k \leq u_e \quad \forall e \in A_{T\prime} \tag{17}$$

$$\sum_{k\in\mathcal{K}_v} x_e^k \leq b_e \quad \forall e \in H_{T\prime} \tag{18}$$

$$x_e^k \in \{0,1\} \quad \forall k \in \mathcal{K}, e \in A_{T\prime}^k \cup H_{T\prime}^k. \tag{19}$$

由此得到以下上界过程。算法4以当前部分时间扩展网络的一个最优解 $\hat{}x$ 作为输入，该网络的完工时间为 $\hat{}T$。我们还将最优性因子容差 $\alpha \geq 0$ 以及当前已知的 T^* 的最佳上界作为输入，记为 UB。$\hat{}x$ 为每个 k_k K 定义一条轨迹 $\hat{}Q_k$，将其投影到底层基础图后，为每个 $k \in$ K 定义一条路径 P_k。然后，我们求解具有固定路径且上界为 $T' = \lceil(1+\alpha)\,\hat{}T\rceil$ 的 UPR问题，其中每个 $k \in$K 都必须遵循底层路径为 P_k 的轨迹。如果该问题可行，我们就检查其值V 是否小于当前最佳上界；若是，则输出当前可行解。否则，返回原上界。

算法4 Compute-UB(D, ˆx, ˆT α, UB) 输入：基础网络 D =(N, A)，值为 ˆT 的最优部分网络解 ˆx、参数$\alpha \geq 0$，以及当前 T* 值的上界 UB 1: T′ =⌈(1 + α) ˆT ⌉，且 ¯x =∅2: $D_{T'}$ =($N_{T'}$, $A_{T'}$ ∪ $H_{T'}$) 3: 令 ˆQ ={ $\hat{Q}_k$}$_{k\in\mathcal{K}}$ 表示由 ˆx 给出的轨迹集合 4: 对每个 k ∈K，令 P_k 表示轨迹 $\hat{Q}_k$在 D 中的底层路径 5: $A^k_{T'}$ ={((v,t), (w,t′)) ∈$A_{T'}$:vw ∈A(P_k)}，且 $H^k_{T'}$ ={((v,t), (v,t′)) ∈$H_{T'}$:v ∈N(P_k)} 6: $D^k_{T'}$ =($N_{T'}$, $A^k_{T'}$ ∪$H^k_{T'}$) 7: 求解 UPR-FP($\{D^k_{T'}\}_{k\in\mathcal{K}}$, $D_{T'}$) 8: 如果 UPR-FP($\{D^k_{T'}\}_{k\in\mathcal{K}}$, $D_{T'}$) 可行且其最优值为 ¯T′，则 9: 令 ¯x 为 UPR-FP($\{D^k_{T'}\}_{k\in\mathcal{K}}$, $D_{T'}$) 的一个最优解 10: UB =min{UB ¯T′} 11: 否则 12: UB =UB 13: 返回 UB ¯x

4.2 增广步骤

现在考虑这样一种情况：使用上界模型无法将部分解 $\hat{x}$ 转换为 UPR(D_T) 的等成本解（算法4中执行了第12行）。本节将详细说明如何扩充集合N_S。

由于松弛过程，我们知道，$\hat{x}$ 可能无法转换为具有相同完工时间的 UPR(D_T) 解，原因是D_S中存在缩短的弧、弧容量被松弛，以及节点存储量被松弛。

设 e =((v，t)，(w，t')) ∈$A_S \cup H_S$ 是 $\hat{x}$支撑集中的一条定时弧。修正A_S [1]中的短弧有一种成熟的方法：

$$\text{if } t' < t + \tau_{vw}, \text{ we add the timed node } (w, t + \tau_{vw}).$$

现在我们继续处理超出弧容量和存储容量的弧。对于每条定时弧 e =((v，t)，(w，t')) ∈$A_S \cup H_S$，令 $\hat{x}_e$ 表示根据 $\hat{x}$分配给弧e 的总活跃流。也就是说，

$$\hat{x}_e = \begin{cases} \sum_{k\in\mathcal{K}} \hat{x}^k_e & e \in A_S \\ \sum_{k\in\mathcal{K}_v} \hat{x}^k_e & e \in H_S \end{cases}$$

若 e ∈A_S 且 $\hat{x}_e > u_e$，则根据 u' 的构造，(v,t + 1) ∉N_S。因此，我们将把 (v,t + 1) 加入 N_S。

若 e ∈H_S 且 ˆx$_e$ > b$_e$，根据 b′$_e$ 和 m$_S$(v,t) 的定义可知，存在某个 (z ¯t) ∈N^-_T(v,t) 或 (z ¯t) ∈N^-_S(v,t)，使得 m$_S$(z ¯t) > 1。对于每个满足 m$_S$(z ¯t) > 1 的 (z ¯t) ∈N^-_S(v,t)，我们将 (z ¯t + 1) 加入 N_S。如果 (v,t + 1) 尚未在集合中，我们也会将其加入 N_S，从而确保在下一次迭代中 N^-_S(v,t) ⊆N^-_T(v,t)。

下一页的算法 5 重述了这些过程中的每一个。

算法 5 扩充$N_S(D_S, D, \mathcal{K}, \hat{x})$

输入：当前的部分时间扩展网络 $D_S=(N_S, A_S\cup H_S)$、基础图 $D=(N, A)$、数据包集合 K，以及 UPR(D_S) 的最优解 $\hat{x}$

$N'_S \leftarrow N_S$

对于 $e=((v, t), (w, t')) \in \text{supp}(\hat{x}) := \{e \in A_S \cup H_S : \hat{x}_e > 0\}$ **执行**

  如果 $e \in A_S$ **则**

    根据 $\hat{x}$，$\hat{x}_e := \sum_{k\in\mathcal{K}} \hat{x}_e^k$，计算分配给定时弧 e 的流。

    如果 $t' < t + \tau_{vw}$ **then**

      $N'_S \leftarrow N'_S \cup \{(w, t + \tau_{vw})\}$

    如果 $\hat{x}_e > u_e$ **then**

      $N'_S \leftarrow N'_S \cup \{(v, t + 1)\}$.

  如果 $e \in H_S$ **那么**

    根据 $\hat{x}$、$\hat{x}_e := \sum_{k\in\mathcal{K}_v} \hat{x}_e^k$，计算分配给定时弧$e$ 的相关流。

    如果 $\hat{x}_e > b_v$ **then**

      对于所有 $(z, \bar{t} = t - \tau_{z,v}) \in N_T^-(v,t) : \mathbb{m}_S(z, \bar{t}) > 1$ **执行**

        $N'_S \leftarrow N'_S \cup \{(z, \bar{t} + 1)\}$.

    $N'_S \leftarrow N'_S \cup \{(v, t + 1)\}$.

    返回 N'_S

命题 4.1. 给定一个最小完工时间为 T^* 的 *UPR* 实例，算法 *5* 仅向 N_S 添加时间节点(v,t)，其中 $t \le T^* + 1$。

证明。 在任意一次迭代中，UPR(D_S) 的最优解 $\hat{T}$ 至多为 T^*，因为 UPR(D_S) 是 UPR(D_T) 的松弛。当由于存储容量超限而修正弧 $e=((v,t),(v,t')) \in H_S$时，我们知道 $t' \le T^*$，因为 v 不是对 $\hat{x}_e$有贡献的商品流的目的地，且根据约束 (8)。此外，对于此修正，我们添加节点 (w,t)，其中 $t \le t' + 1$，因为 $\tau \ge 0$。

现在考虑对 A_S 中弧 $((v,t),(w,t'))$ 的修正。如果该弧超出容量 u，则将节点 $(v,t+1)$ 添加到 N_S。由于 $t \le T^*$，显然 $t + 1 \le T^* + 1$。最后，如果该弧太短，则将节点 $(w, t + \tau_{vw})$ 添加到 N_S。由于我们用约束 (8) 替代了约束 (1)，可知 $t + \tau_{vw} \le \hat{T} \le T^*$。

□

沿着证明的思路，我们很容易得到推论 4.1。

推论 4.1。 给定一个最小完工时间为 T^* 的 *UPR* 实例，其中 $\tau_a > 0$ 对所有 $a \in A$ 都成立，算法 *5* 仅向 N_S 添加节点 (v,t)，其中 $t \le T^*$。

命题 4.1 指出了 DDD 方法相较于求解 UPR(D_T) 的优势：当给定的上界 T远大于 T^* 时，DDD 方法在整个算法过程中都能维持一个小得多的时间扩展网络。

在 DDD 应用于 SND [1], 的原始方法中，上界模型的解决定了增广步骤中需要校正哪些定时弧。然而，在我们的模型中，我们会校正 UPR(D_S) 最优解支撑集中的每条定时弧，只要它的长度不足，或超出了原始弧和存储

容量。因此，无需在每次迭代中都运行上界过程；只有当连续两次迭代报告的完工时间相近时才运行该过程，或许能节省时间。虽然在实验中我们每次迭代都求解了上界模型，但值得测试这种替代方法。

算法 6 求解 UPR-DDD(D,K,T α)

输入：基础网络 $D=(N,A)$、商品集合 K、一个关于最优完工时间的上界 T，以及一个最优性参数 $\alpha \geq 0$

1：$N_S \leftarrow$生成-初始-N_S(D,K,T)
2：$\bar{x} = \varnothing$
3：上界 $\leftarrow T$
4：下界 $\leftarrow 0$
5：间隙 =(上界 −下界)/上界
6：**当** 间隙 $> \alpha$ 或 $\bar{x} \neq \varnothing$ **执行**
7：　A_S，$H_S \leftarrow$生成-$A_S \cup H_S$(N_S，D，T)
8：　$D_S \leftarrow (N_S, A_S \cup H_S)$
9：　求解 UPR(D_S)，并令 $\hat{x}$ 为一个最优解，其值为 $\hat{T}$
10：　下界 $\leftarrow$ 最大{LB，$\hat{T}$}
11：　上界 $\bar{x} \leftarrow$ Compute-UB(D, $\hat{x}$, $\hat{T}$ α,上界)
12：　间隙 =(上界 − 下界)/上界
13：　**如果** 间隙 $\leq \alpha$ 且 $\bar{x} \neq \varnothing$ **则**
14：　　停止。已找到一个与最优解相差不超过 α 的解，适用于 UPR(D_T)。
15：　　**返回** $\bar{x}$,UB
16：　**else**
17：　　$N_S \leftarrow$ 增广-N_S(D_S，D，K，$\hat{x}$)

现在我们证明算法的正确性，并界定迭代次数。需要注意的是，我们可以根据T^*来界定迭代次数，而不仅仅是根据T。

定理 4.1。 算法 *UPR-DDD*(D,$\mathcal{K}$,T α) 终止时得到的解，其完工时间至多为$(1+\alpha)T^*$，且最多经过$|N|T^*$次迭代。

证明。 首先回顾一下，由于所有输入数据均为整数，且已知$T^* \leq T$，最优解的决策时间属于 $[T]$。

考虑算法的一次迭代，此时部分时间扩展网络为$D_S=(N_S, A_S \cup H_S)$，松弛容量由u'和b'给出。令$\hat{Q}$为D_S中给出最小工期路由的轨迹集合。对于每个$k \in$K，令$\hat{Q}_k$表示$\hat{Q}$中数据包k的轨迹。令$\hat{T}$表示$\hat{Q}$在$\bar{D}_S$中的完工时间。令Q为通过求解相应上界得到的轨迹集合，并假设两个完工时间之间的因子间隙大于 α。

因此，在时间范围为 $\alpha \hat{T}$ 的情况下，我们无法在D_T中获得与Q具有相同底层路径的轨迹，同时满足原有的容量u和b。具体而言，显然不可能$\hat{}$仅将每个数据包的轨迹指定为Q_k，并使其位于D_T中。因此，必定是Q中的某条定时弧过短，或者超出了弧容量或节点存储量。正如我们在第4节2中所论述的，在每种情形下，必定存在

一个时间节点（v,t）$\in N_T \setminus N_S$，可以将其加入N_S。

此外，我们在命题 4.1 中证明，我们的算法只会向v,tN_S 添加满足$t \leq T^* + 1$的定时节点。由于每次迭代至少添加一个定时节点，且第一次迭代时每个节点至少有一个副本，因此 DDD算法会在至多 $|N|T^*$ 次迭代内终止。 □

推论 4.2. 算法 $UPR - DDD(D, \mathcal{K}, T\ \alpha = 0)$ 至多经过$|N|T^*$ 次迭代即可终止并得到最优解。

5 计算结果

为展示 DDD算法的有效性，我们比较了 DDD算法与原始完整整数规划 UPR(D_T) 在地理实例和几何实例上的运行时间。对于地理实例，我们依据美国的人口中心来选择节点和弧。我们构造几何实例来模拟社交网络。

对于每个问题实例，我们首先求解时间范围T足够大的 DDD 实例，使得$T^* < T$。这次初始求解得到T^*的值。随后，为比较 DDD 和完整整数规划的求解时间，我们将每种算法的时间范围上界设为 T^*,$1.5T^*$和 $2T^*$，并分别运行至多两小时。因此，每个实例总共求解七次。请注意，我们仅在分析中将上界T 设为T^* 的倍数。在实际应用中，我们会选择足够大的T 值，以确保所有数据包都能在时间T内完成路由。每种算法均使用 Python 3.6.9 编写，并采用 Gurobi 8.1.1 [14] 作为优化求解器。使用求解器的确定性选项，将运行时间限制设为 7200 秒（两小时），并将实例求解至距最优性 1% 以内。实例在配备 256GB 内存、运行 Linux操作系统的 64 核 2.6GHz Xeon Gold 6142处理器上运行。每个实例的运行限制为 5 个核心。生成的实例可在 https://github.com/madisonvandyk/UPRlib 找到。

5.1 地理实例

数据集

对于基础图，我们使用美国人口最多的前n 个城市的位置。我们随机选择m 条弧来构成A，并将 τ_a 设为以百英里为单位的距离，向上取整到最近的整数。我们计算每对顶点之间的最短有向路径。随后，我们从有向图D 中选择k 个随机起讫点对，要求起点到目的地之间存在有向路径，且最短路径至少包含 δ 条弧，其长度不超过任意起讫点对最短路径最大长度的 γ 倍。我们以这种方式构造起讫点对，以确保最小工期并不简单地等于最短路径的最大长度。对于弧容量和节点容量，我们采用Crainic等人 [5, 6]方法的离散版本，该方法是作为SND的严格测试集而开发的。此后，这种数据集构造方式经过修改，用于分析DDD算法的性能 [1, 21]。对于节点存储容量，我们从端点为 $[\alpha_1, \alpha_2]$ 和 $[\beta_1, \beta_2]$ 的离散均匀分布中选择容量。Crainic等人 [5, 6] 引入了容量比率 $C = |A|k/\sum_{e\in A} u_e$。当$C$趋近于1时，网络的容量约束较弱；随着$C$增大，拥堵水平会提高

。Crainic在求解SND时研究了$C \in \{1,2,8\}$的情形。然而，由于在UPR中，我们不像SND那样会受到激励去合并数据包流，因此需要更加严格的拥堵条件才能生成具有研究价值的问题（拥堵程度最低时，所有数据包都可以沿最短路径进行路由，而无需使用节点存储）。下面列出我们的参数集。

参数

- $n = 20$ – 节点数量；
- $m \in \{30,45,60\}$– 弧的数量；
- $k \in \{200,250,300\}$– 数据包的数量；
- $(\alpha_1, \alpha_2) \in \{(1,\lceil 0.01k \rceil),(1,\lceil 0.0175k \rceil),(1,\lceil 0.025k \rceil)\}$– 弧容量的界限；
- $(\beta_1, \beta_2) \in \{(0,\lceil 0.01k \rceil),(0,\lceil 0.0175k \rceil),(0,\lceil 0.025k \rceil)\}$ – 存储容量的界限；
- $\delta = 3$， $\gamma = 0.90$。

α_1， α_2和 β_1， β_2 的选择可提供一系列拥堵水平，同时确保生成的实例始终可行。m 的三个取值经过选择，以确保我们能够考察DDD算法在连通性各异的网络上的表现，同时保持实例为容量受限——过于稠密的图会使许多顶点对能够通过少数弧相连，从而降低拥塞。类似地，k 的取值范围使我们能够分析DDD算法在不同密度下的有效性。

我们简要概述上述参数如何影响最优时间跨度T^* 以及UPR(D_T)的总体求解时间，其他所有参数保持不变。随着m增大，T^*减小，因为数据包可以通过更短的直接路径传输，且被迫重叠的数据包更少。虽然较大的m 意味着生成UPR(D_T)需要更长时间，但由于T^*显著减小，在我们的实验中总体求解时间反而减少。自然地，随着k增大，T^*增大。正如预期，随着 α和 β增大，T^*减小。

结果

我们首先展示在固定T、m和k时， α、 β所有设置下的平均运行时间（秒）。需要指出的是，“ratio”列表示运行时间的平均比率，而非平均运行时间的比率。标记为 * 的平均值表示至少有一个实例未能在时间限制内终止。

	$k = 200$			$k = 250$			$k = 300$		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	267	878	2.82	288	693	2.02	674	1,463*	1.98
$1.5T^*$	709	1,293	1.85	1,582	1,135	0.67	1,761	1,602*	0.88
$2T^*$	1,891*	959*	0.85*	2,714*	1,368*	0.42*	2,879*	1,838*	0.51*

表1：$m = 30$。

	k = 200			k = 250			k = 300		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	144	303	2.08	210	338	1.73	219	345	1.68
1。5T^*	406	467	1.25	882	734	0.93	975	814	0.79
2T^*	1,491	580	0.40	2,232	652	0.34	3,100*	1,024	0.42*

表2：m = 45。

	k = 200			k = 250			k = 300		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	135	315	2.25	152	320	2.13	189	467	2.44
1。5T^*	390	517	1.41	485	652	1.27	800	1,958	2.04
2T^*	1,374	532	0.55	1,513	726	0.55	2,814*	2,162*	0.76*

表3：m = 60。

在所有情形中，我们都观察到一个明显趋势：当上界T 相对于T^*增大时，UPR(D_T)的运行时间增幅远大于DDD的运行时间增幅。这一结果符合直觉，因为随着T 增大，UPR(D_T)会变大，网络中也会引入额外的对称性——在最优解中，大多数数据包可能拥有越来越多的可行轨迹。由于部分时间扩展网络较为稀疏，许多这类额外的对称性得以避免。

图8至图13报告了当T = 1.5T^* 且T = 2T^*时各项实验的运行时间。每张图都展示了在给定时间限制内求解的实例数量。这些图表所呈现的结论与上表相同。随着m 减小、T 增大，DDD相较于完整整数规划的性能优势愈发明显。

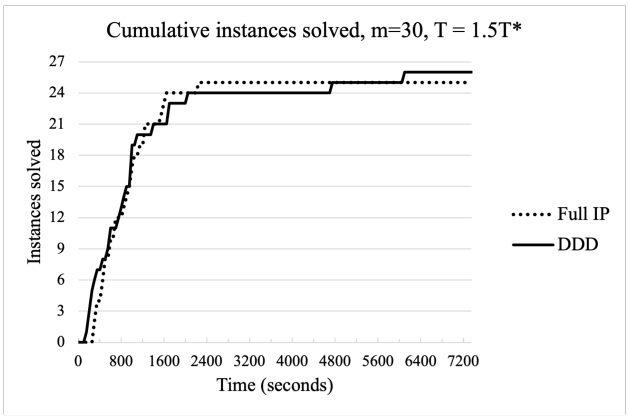

图9：m = 30，T = 1.5T^*

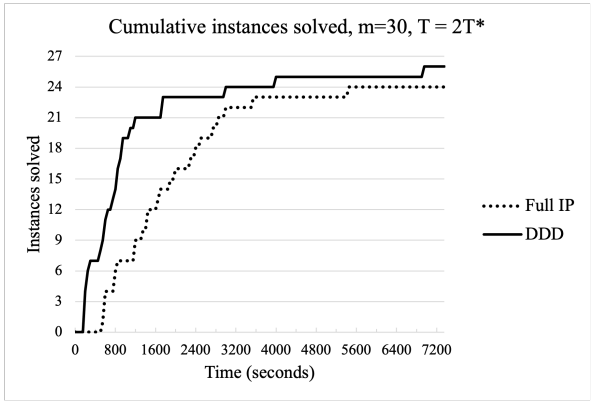

图10：m = 30，T = 2T^*

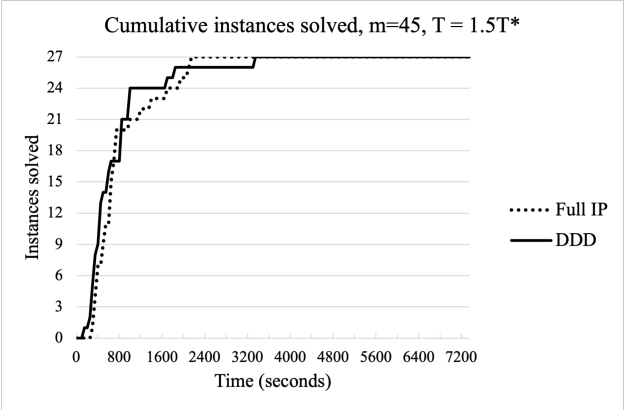

图11：$m = 45$，$T = 1.5T^*$

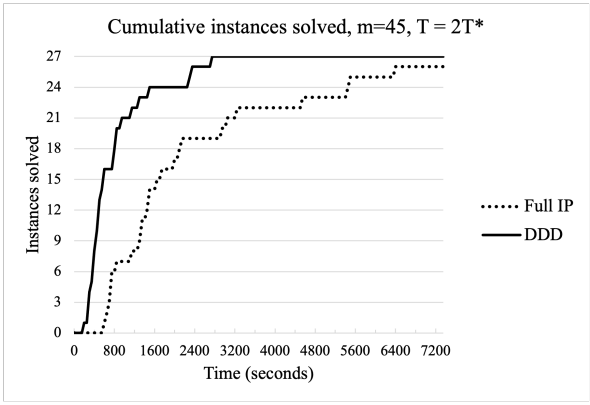

图12：$m = 45$，$T = 2T^*$

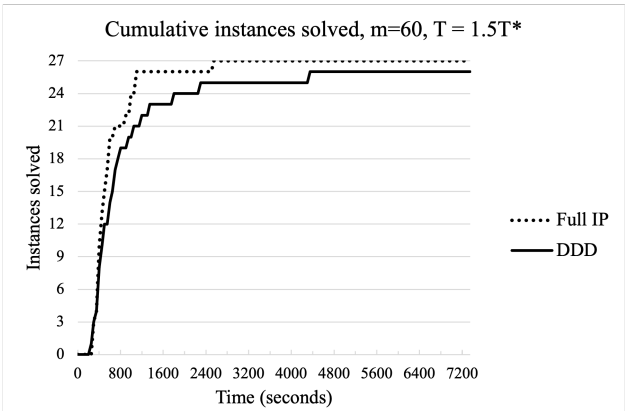

图13：$m = 60$，$T = 1.5T^*$

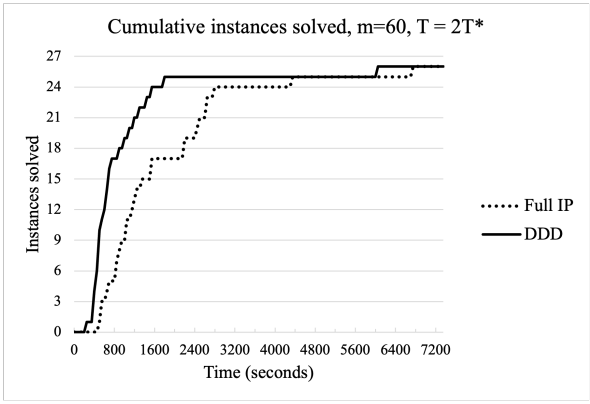

图14：$m = 60$，$T = 2T^*$

迭代规模

当$T = T^*$时，UPR(D_T)的表现优于DDD。这并不意外，因为尽管DDD求解的整数规划规模小于UPR(D_T)，但要使算法终止，我们仍需要相当稠密的部分时间扩展网络。下表给出了DDD终止所需的平均迭代次数，以及最终定时节点集相对于完整定时节点集的平均规模。

	$m = 30$		米 = 45		米 = 60	
因子	迭代次数	$\|N_S^{\mathtt{final}}\|/\|N_T\|$	迭代次数	$\|N_S^{\mathtt{final}}\|/\|N_T\|$	迭代次数	$\|N_S^{\mathtt{final}}\|/\|N_T\|$
T^*	9.67	0.43	7.00	0.60	6.93	0.72
$1.5T^*$	9.85	0.31	6.93	0.40	6.78	0.47
$2T^*$	9.89	0.23	7.07	0.31	7.07	0.37

表4：平均迭代次数，以及$N_S^{\mathtt{final}}$的相对大小。

平均迭代次数会随着弧的数量 m 增加而减少。这并不令人意外，因为弧的数量越多，数据包轨迹所需的定时弧就越少，因此 DDD 能以更少的迭代次数生成可行

轨迹。同时，$|N_S^{\texttt{final}}|/|N_T|$ 的平均比率会随 m 增加，这解释了为什么当m取值较高时，DDD 的表现并未改善。该比率较高的一个原因是存储容量的细化过程：在修正节点v处超出的存储量时，新增的时间节点数量在一定程度上取决于v的度。我们将在第 5.2 节进一步探讨稀疏性对 DDD 性能的影响。

细化趋势

在每次迭代中，我们求解定义在部分时间扩展网络上的整数规划，并得到一个解 $\hat{x}$。如果 $\hat{x}$ 无法转换为D_T中的最优解，那么 $\hat{x}$ 的支撑集中必定存在定时弧，它们要么过短，要么超过了原始吞吐量或存储水平。下文的图中，我们考察这些违反约束类型中的每一种如何影响每次迭代中的细化过程。

在图 15 中，我们展示了每次迭代中，$\hat{x}$ 的支撑集中不可行弧的平均比例，并按弧过短、超过吞吐量或超过存储水平分类。在图 16 中，我们展示了每次迭代中，为纠正每种违反约束类型而添加的定时节点的平均比例。我们纳入了 DDD 至少经过 7 次迭代才终止的测试实例，并且只考察前 7 次迭代。如下文所示，这足以观察到明显的趋势。

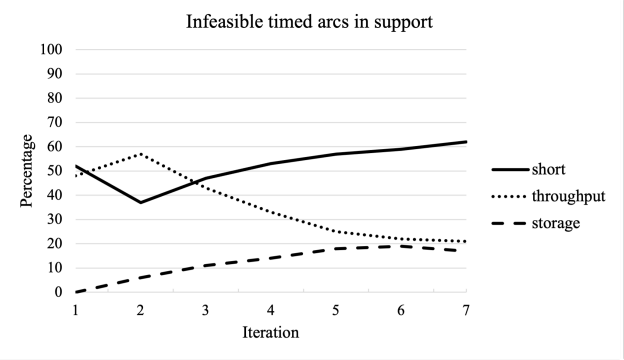

图 15：不可行定时弧的比例。

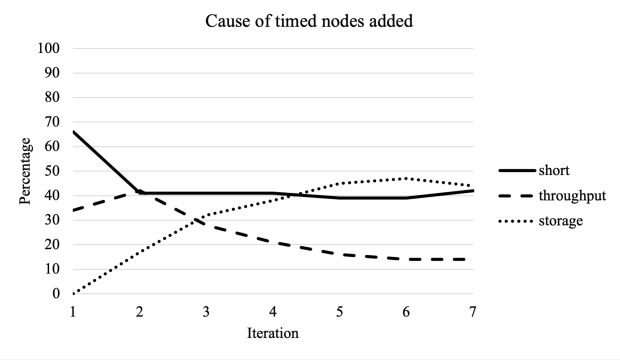

图 16：添加时间节点的原因

请注意，在第一次迭代中，$\hat{x}$ 的支撑集中没有超过存储容量的定时弧。如果不存在吞吐量容量（或至少不存在具有限制性的吞吐量容量），则无需存储，因此这一点并不令人意外。出于同样的原因，在每次迭代中，超过吞吐量的定时弧数量多于超过存储容量的定时保持弧数量也是自然的。随着迭代的进行，$\hat{x}$的支撑集规模增大，并将数据包流分散到更多的定时弧上。因此，超过吞吐量的不可行定时弧所占比例降低是自然的。在图 22 中，我们看到，由于超过存储容量而添加的时间节点比例超过了由于超过吞吐量而添加的时间节点比例。这是因为，在纠正超过吞吐量的问题时，我们只添加一个时间节点；而在纠正节点 v处的存储容量超限问题时，最多可以添加 $\deg_D(v)+1$ 个时间节点。

5.2 几何实例

数据集

我们的实例由随机几何图组成，这类图已被广泛用于建模人类社交网络 [19, 26]。我们首先从一个 $l \times l$ 网格中随机选择 n 个节点。然后，如果任意一对节点之间的 L1 范数距离不超过 p，就用一条弧连接这对节点。这是随机几何图的离散版本。由于社交网络的直径较小，我们按照 Kleinberg [19] 的经典构造扩充弧集。具体而言，对于每个节点 $v \in N$，我们独立随机添加 q 条“远程”弧（v, w），其中，从 v 出发的第 i 条有向弧的端点为 w 的概率与 $\|v - w\|_1^{-r}$ 成正比。对于为构成 A 而生成的每条弧，我们将运输时间设为端点之间的 L1 范数距离。

随后，我们按照与地理实例相同的流程，从有向图 D 中随机选取 k 个起讫点对。与地理实例相同，我们从离散均匀分布中选取容量，其端点为（α_1，α_2）和（β_1，β_2），分别对应弧容量和节点存储。现在列出我们的参数集。

参数

- $l = 25$ – 网格长和宽；
- $n = 20$– 节点数；
- $k \in \{200, 225, 250\}$– 数据包数量；
- $p \in \{3, 4\}$– 局部连接的半径；
- $q \in \{1, \{1, 2\}\}$ – 每个节点的长程连接数量；
- $r = 0.5$ – 用于选择长程连接的缩放因子；
- $(\alpha_1, \alpha_2) \in \{(1, \lceil 0.01k \rceil), (1, \lceil 0.02k \rceil)\}$ – 弧容量的界限；
- $(\beta_1, \beta_2) \in \{(0, \lceil 0.01k \rceil), (0, \lceil 0.02k \rceil)\}$– 存储容量的界限。

当 $q = \{1, 2\}$ 时，我们以相同的概率为每个节点选择 1 条或 2 条长距离弧。我们将 p 和 q 设得足够小，使实例容量受限，同时仍能呈现不同程度的局部和全局连通性。图 17 展示了使用 n 和 l（如前所述）得到的稀疏网络，其中 $r = 0.5$、$p = 3$ 和 $q = 1$。相比之下，图 18 展示了使用参数 $r = 0.5$、$p = 4$ 和 $q = \{1, 2\}$ 得到的稠密网络。在每幅图中，节点的位置都对应于 $l \times l$ 网格中的位置。

当 $r = 0$ 时，长程连接是均匀随机选取的；当 $r = 1$ 时，连接的选取概率与距离倒数成正比。由于我们希望实现一定程度的全局连通性，但在社交网络中，长距离连接比短距离连接少见，因此我们使用 $r = 0.5$。如前所述，为了构造有意义的问题，我们要求 UPR 中的拥堵程度高于 SND。 α_1, α_2 和 β_1, β_2 的取值范围能够产生不同的拥堵水平，同时确保所得实例可行。

我们简要概述上述参数如何影响最优时间范围 T^* 以及

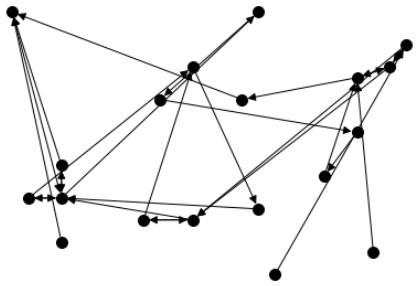

图 17：局部和全局稀疏

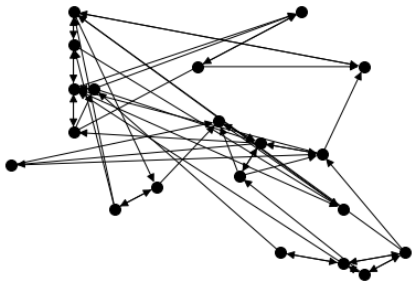

图 18：局部和全局稠密

其他参数保持不变时，UPR(D_T)的整体求解时间。随着p和q增大，T^*会缩短，因为数据包可以沿更短的直接路径传输，且被迫重叠的数据包会更少。随着弧容量和存储容量增大，T^* 会缩短，因为网络能够容纳更多活动数据包。

结果

总体而言，当 $T = 2T^*$ 时，DDD 比求解完整整数规划更快；对于较慢的稀疏实例，当 $T = 1.5T^*$ 时也更快。平均来看，我们发现，当 $T = 2T^*$ 时，DDD 完成所需的时间为运行 UPR(D_T) 所需时间的 49%。请注意，这一估计偏高，因为运行 UPR(D_T) 时，有些实例未能在允许的时间内完成。

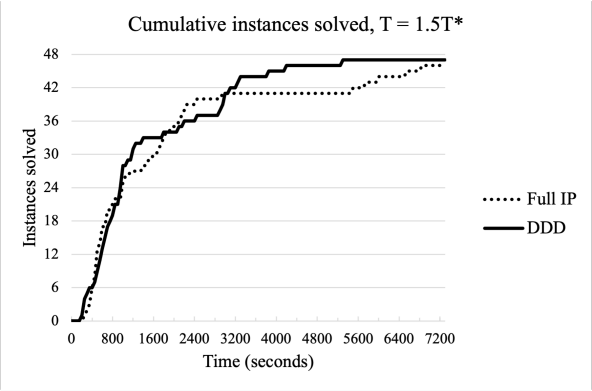

图 19：累计求解实例，$T = 1.5T^*$

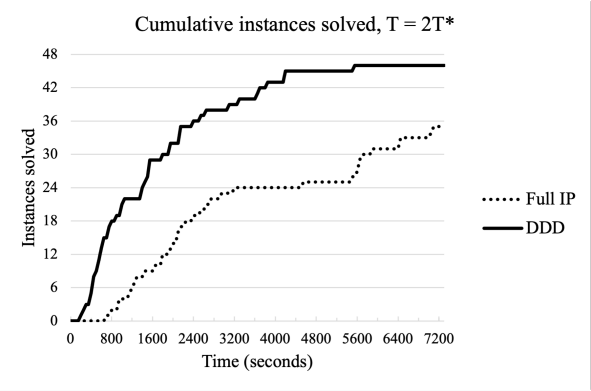

图 20：累计求解实例，$T = 2T^*$

为了更仔细地考察结果，我们根据底层网络的局部和全局连通性水平将实例分组。同样，“比率”列给出的是 UPR(D_T) 与 DDD 运行时间之比的平均值。同样，标有 * 的平均值表示至少有一个实例未能在时间限制内结束。因此，如果所有实例都运行至结束，同一行中的比率可能会更低。

低局部连通性和低全局连通性

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	363	1,946	6.17	477	1,069	2.17	651	1,921	4.41
$1.5T^*$	2,705	3,220	1.18	5,066*	2,702	1.15*	3,644	1,523	0.47
$2T^*$	6,780*	3,084	0.44*	6,191*	3,188	0.79*	5,791*	1,663	0.27*

表5：($p = 3, q = 1$)。

低局部连通性和高全局连通性

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	234	821	3.20	285	550	2.97	245	893	4.09
$1.5T^*$	1,205	768	0.87	2,465*	1,235	2.13*	1,029	2,672	3.61
$2T^*$	2,803	2,904*	0.75	3,591*	1,391	0.59*	6,284*	1,584	0.37*

表 6：($p = 3$，q ={1，2)。

高局部连通性和低全局连通性

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	186	478	2.48	248	734	2.76	281	791	2.70
$1.5T^*$	1,343	1,581	1.48	1,092	1,263	1.23	1,728	1,687	1.05
$2T^*$	4,726*	2,538*	0.40*	3,776*	1,867	0.49*	5,926*	1,880	0.40*

表7：($p = 4, q = 1$)。

高局部和高全局连通性

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率	UPR(D_T)	DDD	比率
T^*	152	318	2.23	187	382	2.12	202	438	2.20
$1.5T^*$	868	533	0.82	796	539	0.82	590	834	1.41
$2T^*$	2,502	1,093	0.58	3,925*	624	0.31*	1,803	896	0.52

表8: ($p = 4, q \in \{1,2\}$).

总体而言，我们得出了与第5.1节相同的结论。随着上界相对于 T^* 增大，DDD 相较于完整整数规划的性能有所提升。对于稀疏实例，即使 $T = 1.5T^*$，DDD 的表现也优于 UPR(D_T)（DDD 的平均运行时间为完整整数规划的 93%）。从下表可以看出，底层图越稀疏，DDD 终止前的平均迭代次数就越多（“迭代次数”列）。尽管平均迭代次数增加，$|N_S^{\mathtt{final}}|/|N_T|$ 的平均比率却有所下降，因此 DDD 在较稀疏的实例上仍表现得相对更好。未来的研究方向之一是考察局部和全局连通性如何影响 DDD算法的整体性能。

	$p = 3,\ q = 1$		$p = 3,\ q = \{1,2\}$		$p = 4,\ q = 1$		$p = 4,\ q = \{1,\ 2\}$	
因子	迭代。	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	次迭代	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	次迭代	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	次迭代	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$
T^*	9.75	0.57	8.42	0.64	7.33	0.64	6.17	0.66
$1.5T^*$	8.42	0.37	8.58	0.45	7.50	0.44	6.25	0.46
$2T^*$	8.58	0.28	8.25	0.34	7.67	0.34	6.58	0.35

表 9： $\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$ 的平均迭代次数和平均值。

细化趋势

我们观察到，DDD 中细化步骤呈现出与第5.1节相同的趋势。随着迭代次数的增加，纠正违反存储约束的情况对细化过程的影响越来越大，而违反吞吐量约束的情况则越来越少。纠正短弧仍然是导致大多数定时弧不可行的主要原因，而新增时间节点的原因则由纠正较短的定时弧和纠正超出限制的存储弧共同构成。

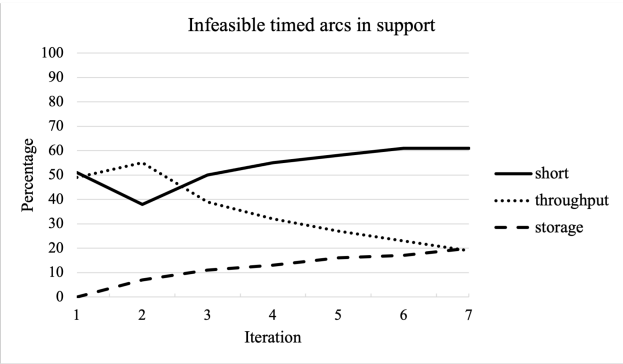

图 21：不可行定时弧的比例。

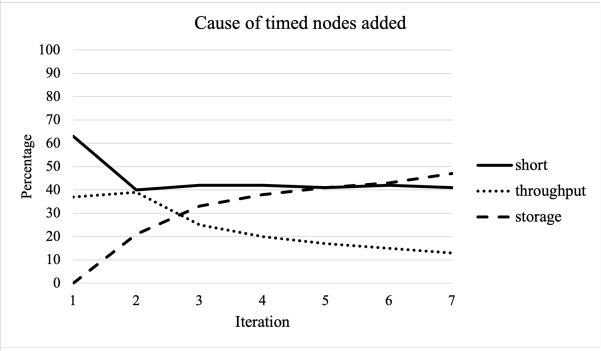

图 22：添加定时节点的原因

6 结论

本文研究了通用分组路由问题，并开发了一种用于精确求解该问题的新型 DDD算法。据我们所知，本文首次为 DDD 框架中存储水平所需的松弛提供了有限界限。为证明我们的下界模型是一种松弛，我们给出了从完全时间扩展网络到部分时间扩展网络的映射的结构性结果。这些结构性结果将有助于扩展 DDD 框架，以求解具有库存持有成本的问题。

我们介绍了一种实现，并给出了一些说明性计算结果。在地理实例和几何实例中，我们都观察到，随着所提供的上界 T 增大，完整整数规划的运行时间比 DDD 算法增长得更快。在此前应用 DDD 的许多问题中，最优解的支撑集相对于完全时间扩展网络而言较小。但在 UPR 中情况不再如此，因为弧容量和存储水平会增大解的支撑集。

一项重要贡献可以是调整 DDD 方法，使不同商品流使用不同的部分时间扩展网络。其挑战在于确保我们在每次迭代中仍能获得最优解的下界。

最后，我们观察到，随着所提供的上界相对于T^*增大，完整整数规划的性能会下降。人们通常认为，连续形式的表现不如时间索引公式 [21],，但当上界$T >> T^*$达到某个值时，情况很可能不再如此。此外，当路径预先给定时（例如 UPR-FP），连续形式可能更有效。未来的一个研究方向是：对于在底层静态图中加入指定路径的问题，研究连续形式与时间扩展公式之间的运行时间差异。了解连续形式与时间索引公式之间的这种权衡或许很有意义，并可用于改进 DDD算法中的上界和增广步骤。

致谢

我们感谢 Cristiana L. Lara 提供宝贵的讨论和反馈。

参考文献

[1] N. Boland、M. Hewitt、L. Marshall 和 M. Savelsbergh。连续时间服务网络设计问题。*OperationsResearch*，65(5):1303–1321，2017年。[2] N. Boland 和 M. Savelsbergh。关于时间依赖模型整数规划的观点。*TOP-invitedpaper*，27:147–173，2019年。[3] C. Busch、M. Magdon-Ismail、M. Mavronicolas 和 P. G. Spirakis。直接路由：算法与复杂性。载于*ESA*，2004年。[4] T. Crainic 和 M. Hewitt。网络设计及其在运输与物流中的应用，第12章，第347–382页。编者 T.G. Crainic、M. Gendreau 和 B. Gendron，Springer，2021年。[5] T. G. Crainic。货运运输中的服务网络设计。*EuropeanJournal ofOperationsResearch*，122(2):272–288，2000年。[6] T. G. Crainic、A. Frangioni 和 B. Gendron。多商品容量约束固定费用网络设计的基于束的松弛方法。*DiscreteApplied Mathematics*，112(1):73–99，2001年。[7] S. Dash、O. Gunluk、A. Lodi 和 A. Tramontani。带时间窗的旅行商问题的时间桶公式。*INFORMS Journalon Computing*，24(1):132–147，2012年。[8] M. Di Ianni。高效延迟路由。*TheoreticalComputerScience*，196:131–151，1998年。[9] U. Feige 和 J. Kilian。零知识与色数。*JournalofComputer and SystemSciences*，1998年。

[10] L. Fleischer 和 M. Skutella。无中间存储的最小成本时变流。*ACM-SIAM*离散算法年度研讨会论文集，第66–75页，2003年。[11] L. Fleischer 和 M. Skutella。最快时变流。*SIAM*计算杂志，36(6):1600–1630，2007年。[12] L. Ford 和 D. Fulkerson。由静态流构造最大动态流。*Operations Research*，第419–433页，1958年。[13] L. Ford 和 D. Fulkerson。网络流。普林斯顿大学出版社，1962年。[14] Gurobi Optimization, LLC。Gurobi优化器参考手册，2023年。[15] A. Hall、S. Hippler 和 M. Skutella。多商品时变流：高效算法与复杂性。《计算机科学讲义》（含《人工智能讲义》和《生物信息学讲义》子系列），2719:397–409，2003年。[16] E. He、N. Boland、G. Nemhauser 和 M. Savelsbergh。用于最短持续时间时变最短路径问题的动态离散发现算法。约束规划、人工智能与运筹学集成国际会议（*CPAIOR*），第289–297页，2018年。[17] M. Hewitt。连续时间负载设计问题的增强型动态离散化发现。《交通科学》，53:1731–1750，2019年。[18] M. Hewitt。灵活定班服务网络设计问题。《交通科学》，56:1000–1021，2022年。[19] J. Kleinberg。小世界现象：算法视角。计算理论研讨会（*STOC*），第163–170页，2000年。[20] F. Lagos、N. Boland 和 M. Savelsbergh。连续时间库存路径问题。《交通科学》，54(2)，2020年。[21] F. Lagos、N. Boland 和 M. Savelsbergh。通过往返路线求解连续时间库存路径问题的动态离散化发现。《计算机与运筹学》，141，2022年。[22] C. L. Lara、J. Koenemann、Y. Nie 和 C. C. de Souza。通过拉格朗日分解实现可扩展的时序感知网络设计，http://www.optimization-online.org/DB_HTML/2022/02/8788. html，2022年。[23] F. Leighton、B. Maggs 和 S. Rao。以 o(拥塞+扩张度) 步完成分组路由与作业车间调度。《组合数学》，14:167–186，1994年。[24] F. Leighton、B. Maggs 和 A. Richa。快速算法：寻找 o(拥塞+扩张度) 分组路由调度方案。《组合数学》，196:375–401，1999年。

[25] M. Marshall、M. Hewitt、N. Boland 和 M. Savelsbergh。基于区间的动态离散化发现方法，用于求解连续时间服务网络设计问题。《交通科学》，55 (1)，2020年。[26] S. Milgram。小世界问题。《今日心理学》，1(61)，1967年。[27] B. Peis、M. Skutella 和 A. Wiese。分组路由：复杂性与算法。近似与在线算法研讨会，《计算机科学讲义》，第217–228页，2009年。[28] B. Peis 和 A. Wiese。具有任意带宽和运输时间的通用分组路由。*G¨unl¨uk O.，Woeginger G.J.*（编）整数规划与组合优化（*IPCO*）*2011*，6655：362–375，2011年。[29] S. Pottel 和 A. Goel。具有时变持续时间和资源消耗的活动调度。*European Journal of Operations Research*，2021年。[30] C. Scheideler。互连网络的通用路由策略。《计算机科学讲义》，1390：57–71，1998年。[31] Y. O. Scherr、M. Hewitt、B. A. Neumann Saavedra 和 D. C. Mattfeld。针对混合自主车队服务网络设计问题的动态离散化发现。《交通研究*B*辑》，141：164–195，2020年。[32] M. Skutella。时间网络流导论。组合优化研究趋势：波恩 *2008*，第451–482页，2009年。[33] D. M. Vu、M. Hewitt、N. Boland 和 M. Savelsbergh。通过动态离散化发现求解带时间窗的时变旅行商问题。《交通科学》，54(3)：703–720，2019年。[34] D. M. Vu、M. Hewitt 和 D. D. Vu。通过动态离散化发现求解时变最短行程持续时间问题和送货员问题。*European Journalof OperationalResearch*，2022年。[35] X. Wang 和 A. Regan。关于一种新的带时间窗约束旅行商问题时间窗离散化方法的收敛性。《计算机与工业工程》，56(1)：161–164，2009年。

A 严格存储上限的重要性

为展示在离散时间设置下收紧存储松弛的重要性，我们将考察服务网络设计（SND）问题 [1]，并在其中加入严格的节点和弧容量限制。在 SND 的离散时间实例中，给定一个有向图 $D = (N,A)$，其中每条弧 $a \in A$ 都具有相应的运输时间 $\tau_a \in \mathbb{N}_{\geq 0}$、单位流量成本 $c_a \in \mathbb{R}_{\geq 0}$、固定成本 $f_a \in \mathbb{R}_{\geq 0}$和容量 $u_a \in \mathbb{N}_{>0}$。此外，我们还加入严格的节点和弧容量限制。具体而言，给定一个上限 h_a，表示在任意（整数）时间点沿弧 a 发送的卡车数量不得超过该上限；每个节点 $v \in N$ 在任意时刻最多可存储 b_v 个单位。

令 K 表示一个商品集合，其中每个商品具有一个源点 $s_k \in N$ 和汇点 $t_k \in N$，以及必须沿着从 s_k 到 t_k 的单条轨迹进行路由的需求量 q_k（流不可拆分）。对于商品 $k \in$K，令 r_k 和 d_k 分别表示其释放时间和截止时间。若一条 *feasible* 的 s_k,t_k轨迹从s_k 出发的时间不早于r_k，并且到达t_k 的时间不晚于d_k，则称其对商品 k 是可行的。SND 的目标是为每个商品确定一条可行轨迹，从而在满足严格节点和弧容量限制的同时，使固定成本和可变成本之和最小。

我们按如下方式构造 SND 实例。基础图 D 是图 23 所示的有向图。图中的每条弧也标注了其运输时间和卡车容量。节点v 和w 标注了其存储容量，而节点s 和t 没有存储上限。令$T \geq 4$。

- $f_a = 1$，$c_a = 0$，以及$h_a = 1$ 对所有$a \in A$均成立；
- $(s1,t1) = (s2,t2) = (s,t)$;
- $qk = 50$，$rk = 0$ 以及$dk = T$ 对于$k \in \{1,2\}$。

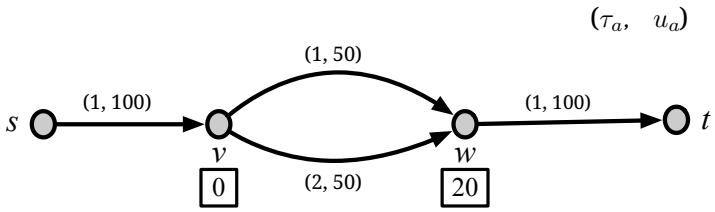

图 23

注意，虽然我们希望在弧sv 和wt上各购买一辆卡车，但这并不可行，因为两条vw 弧的运输时间不同，而且v 和w 的存储容量不足以容纳沿较快路线行驶的单位。因此，解的最小成本为 5。

考虑 DDD 的一次迭代，其中包括s的所有时间副本、v的所有副本，以及节点w仅在时间 0、1、2 和T的副本。需要指出的是，通过添加具有限制性释放时间和截止时间的商品流，不难使其成为部分时间扩展网络所允许的最小定时节点集合。

将 Lagos 等人的存储松弛应用于此离散设置时， [20] 会规定将（w,2）的存储容量设为无穷大，因为（w,3）$\notin N_S$，从而在部分时间扩展网络中得到成本为 4 的解。在 Lagos 等人的细化过程中，当某个定时节点（w,t）发生存储容量超限时，就会将（w,$t+1$）添加到N_S中。然而，只有当w 的 Ω（T）个副本都加入解后，DDD算法才会终止，因为必须纳入商品流 1 和 2 可行到达w 的每个时间，以及其后的一个小时，以防止存储容量松弛。这将导致 DDD算法进行 Ω（T）次迭代。

如果没有引理 3.5 的证明，结合引理 3.4 和 3.6 将只能得到以下较弱的松弛容量性质（就松弛的强度而言）：

($P^{\text{storage-relaxed}}$)：对于任意$e = ((v,t),(v,t')) \in H_S$,

$$b'_e \geq \mathrm{m_S}(v,t) \cdot b_v + U_e,$$

其中

$$U_e = \sum_{(w,t')\in N_T^-(v,t)\cup N_S^-(v,t)} u_{wv}\cdot(\mathtt{m_S}(w,t')-1).$$

然而，这种存储松弛对 DDD算法中迭代次数的界仍然同样不理想。由于相邻节点v 的每个定时副本都包含在N_S、$U_e = 0$ 中，因此我们可以将时间节点（w,t）的存储容量设为 $\mathtt{m_S}$（w,t）$\cdot b_w$。当 $\mathtt{m_S}$（w,t）≥ 3 时，存储量至少为 50，从而使部分时间扩展网络允许商品流 1 和 2 在sv 和wt上重叠。因此，只有当通过商品流 1 和 2 可以到达的w 的每个副本都满足 $\mathtt{m_S}$（w,t）≤ 2 时，DDD算法才会终止，因此该算法仍需要 $\Omega(T)$ 次迭代。

采用（$P^{\mathtt{storage}}$）给出的更紧存储上限后，（w,2）处的容量仅为 40，从而避免了不可行解。因此，按照（$P^{\mathtt{storage}}$）选择存储容量时，DDD算法经过一次迭代后终止。

B 连续情形下的改进

本节将推广并收紧连续情形下静态存储约束的松弛。前文提到，Lagos 等人 [21]研究了具有往返路线的 CIR，其中每个客户设施在任意时刻至多只能有一辆等待车辆。我们将此称为场站约束。设v 为网络中的一个客户节点，w 为其入邻居（在他们研究的往返网络结构中，这是唯一的供应设施）。

在 [21], 所提出的下界模型中，仅当定时节点（v，$t+\varepsilon$）和（w，$t-\tau_{wv}+\varepsilon$）位于当前部分时间扩展网络中时，才会在客户位置v 的时间t 处纳入场站约束。现在我们将说明，本文所证明的界限推广了 [21] 中存储约束松弛的结果，使其适用于任意平面网络，并且即使在连续情形下，也收紧了先前的结果。

为清晰起见，我们在此重述存储分配。（P $^{\mathtt{storage}}$）：对于任意$e = ((v,\ t),\ (v,\ t')) \in H_S$,

$$b'_e \geq \begin{cases} b_v + U_e & \text{if } (v,t+1)\in N_S \\ 2b_v + U_e & \text{if } (v,t+1)\notin N_S. \end{cases}$$

在 Lagos 等人的研究中，ε 的值定义了时间范围的离散化。此外，由于 CIR 中没有硬性吞吐量约束，因此对于所有弧$u_{wv} = \infty$ wv，我们均设置为如此。最后，作者考虑的场站约束具有单位容量，因此对于所有客户顶点$b_v = 1$ v，我们均设置为如此。

当（v，$t+\varepsilon$）$\in N_S$ 时，可知 N_S^-（v，t）$= N_T^-$（v，t）。接着，如果（w，$t-\tau_{wv}+\varepsilon$）$\in N_S$，则可得到 $U_e = 0$。因此，对于 [21], 中所考虑的实例，我们得到以下界。

（$p^{\text{storage-CIR-OB}}$）：对于任意 $e = ((v, t), (v, t')) \in H_S$，其中 v 是客户节点，

$$b'_e \geq \begin{cases} 1 & \text{if } (v, t+\varepsilon) \in N_S \text{ and } (w, t-\tau_{wv}+\varepsilon) \in N_S \\ 2+U_e & \text{if } (v, t+\varepsilon) \notin N_S \end{cases}$$

注意，当 $(w, t+\varepsilon) \in N_S$ 对所有 $(w, t) \in N_S^-(v,t) \cup N_T^-(v,t)$ 都成立时，可得 $U_e = 0$。在这种情况下，只需将 (v,t) 处的堆场容量设为 2，而不必在 $(v, t+\varepsilon) \notin N_S$ 时设为 ∞。因此，此处给出的界限推广并收紧了 Lagos 等人 [21] 提出的界限。

C 地理场景下的两阶段 DDD

Hewitt [17] 最近提出了两阶段 DDD 方法，用以加快 DDD 的初始迭代。在第一阶段，DDD 范式求解混合整数规划的线性规划松弛。在该阶段的每次迭代中，为节省计算时间，求解的是基于部分时间扩展网络定义的 MIP 的线性规划松弛，而不是 MIP 本身。第一阶段最终以线性规划松弛的最优解以及最终的部分时间扩展网络 D_S^{LP} 结束。在第二阶段，DDD 求解原始 MIP，并将部分时间扩展网络初始化为 D_S^{LP}。对于 SND 的变体 [17, 31]，结果表明，与单阶段 DDD 方法相比，这种两阶段方法能够更快地求得最优解。

尽管两阶段 DDD 是一种颇具前景的加速策略，但这种方法的一个缺点是，线性规划松弛最优解的支撑集可能大于混合整数规划最优解的支撑集，因此，与单阶段方法的部分时间扩展网络相比，迭代中构建的部分时间扩展网络可能更大。

本节给出计算结果，比较两阶段方法与原始 DDD 方法在第5.1节所述地理实例集上的表现。对于每次迭代的下界，我们求解由 D_S 诱导的线性规划松弛，并得到 UPR(D_S) 的一个解 $\hat{x}$，其目标值为 $\hat{T}$。请注意，由于允许分数变量，在求解线性规划松弛时，$\hat{x}$ 支撑集中定时移动弧的最终到达时间（记为 final($\hat{x}$)）可能超过 $\hat{T}$。因此，在第一阶段，我们不再像原始 DDD 方法那样设置 $T' = \lceil (1+\alpha)\hat{T} \rceil$，而是设置 $T' = \lceil (1+\alpha)\text{final}(\hat{x}) \rceil$。此外，在线性规划松弛中，不再强制数据包沿单条轨迹行进。令 $\hat{Q}_k = \{\hat{Q}_k^1, \cdots, \hat{Q}_k^{r_k}\}$ 表示 $\hat{x}$ 支撑集中数据包 k 的轨迹集合，并令 $P_k = \{P_k^1, \cdots, P_k^{r_k}\}$ 表示 D 中对应的路径集合。对于每个 $k \in K$，我们重新定义 A_T^k 和 H_T^k 如下

$$A_T^k = \{((v,t),(w,t')) \in A_T : vw \in P_k^i \in \mathcal{P}_k\}, \text{ and}$$
$$H_T^k = \{((v,t),(v,t')) \in H_T : v \in N(P_k^i), P_k^i \in \mathcal{P}_k\}.$$

第一阶段正确性的证明可直接由原始 DDD 方法正确性的证明得出。不过，我们注意到，推论 4.1 不再适用于两阶段方法；相反，在第一阶段中，可能会添加时间节点 (v,t)，其中 $t > T^* + 1$。不过，我们仍然观察到，对于任何

添加的时间节点（v,t），必有 $t \leq T$，其中 T 是给定的上界。因此，阶段1中的最大迭代次数至多为 $|N|T$（这是对定理 4.1 和推论 4.2 的弱化）。

总体而言，我们发现，两阶段 DDD 方法的平均速度慢于 使用原始单阶段 DDD 方法求解实例的速度。具体来说，两阶段 DDD 算法的平均速度比原始 DDD 算法慢 20.4%。如前所述，这并不完全令人意外，原因有几点。虽然只求解线性规划松弛可以加快初始迭代的速度，但这些迭代对应的部分时间扩展网络非常稀疏，因此算法的运行时间自然主要耗费在后续迭代上。在下表中可以看到，两阶段方法平均需要总计 8.72 次迭代，而单阶段 DDD 方法平均在 7.91 次迭代后终止。此外，我们发现使用两阶段方法时，$|N_{\text{最终}}|/|N_T|$（最终定时节点集与完整定时节点集的比值）的平均值增加了 10%。

结果

在表 10 和表 11 中，我们比较两阶段 DDD 与单阶段 DDD（原始 DDD）在运行时间、迭代总次数以及 $|N_{\text{最终}}|/|N_T|$ 平均值方面的表现。

	平均运行时间（秒）			平均 $\lvert N\,final\,S\rvert/\lvert N_T\rvert$		
UB	原始	两阶段	两阶段 原始	原始	两阶段	两阶段 原始
T^*	569	616	1.08	0.58	0.63	1.08
$1.5T^*$	1,019	1,206	1.18	0.39	0.44	1.13
$2T^*$	1,094	1,422	1.30	0.30	0.33	1.10

表10：$|N_{\text{最终}}|/|N_T|$ 的平均运行时间和平均值。

在所有上界因子（$T \in \{T^*, 1.5T^*, 2T^*\}$）下，平均而言，原始 DDD 方法比两阶段方法更快终止。其中一个影响因素是，运行两阶段 DDD 时，最后的定时节点集平均规模更大，从而导致最后几次迭代速度更慢。此外，在表11中，我们可以看到，运行两阶段方法时，迭代次数总计有所增加。使用 DDD 求解线性规划松弛后，两阶段方法在第二阶段平均需要约 2 次迭代。由于后续迭代占据了 DDD 运行时间的主要部分，且两阶段方法导致最后几次迭代速度更慢，这解释了为什么对于该特定问题和 DDD 实现，两阶段方法并未比原始 DDD 方法带来改进。

	两阶段 DDD			原始 DDD
UB	阶段1	阶段2	总计	总计
T^*	6.62	2.04	8.65	7.86
$1.5T^*$	6.63	2.27	8.90	7.85
$2T^*$	6.57	2.04	8.60	8.01

表11：各阶段的平均迭代次数。

Th以下图表展示了在$T = 1.5T^*$和$T = 2T$条件下累计求解的实例 *.

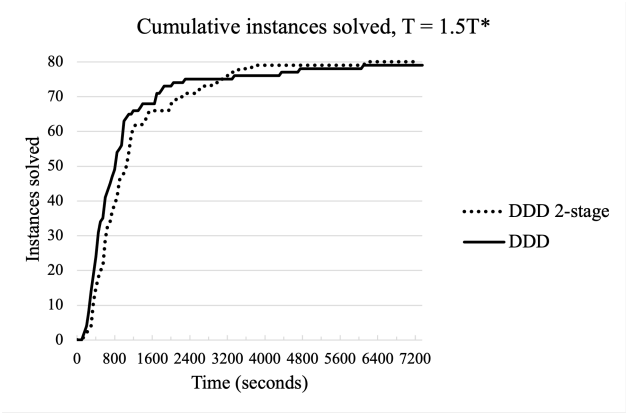

图24：$T = 1.5T^*$

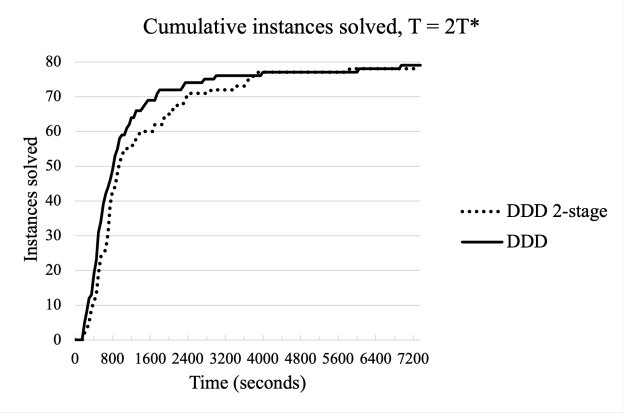

图25：$T = 2T^*$

下表按 m 的各个取值，对两阶段 DDD 的性能进行了细分，其中 m 表示基础图中的弧数量。表中的每个条目都等于两阶段方法中该特征的平均值除以原始单阶段 DDD 方法中该特征的平均值。例如，在满足 $m = 30$，$k = 200$ 且 $T = T^*$ 的实例中，两阶段 DDD 的平均迭代次数为 11.44，而原始 DDD 方法的平均迭代次数为 10.56，因此比率为 11.44/10.56 = 1.08。

	$k = 200$			$k = 250$			$k = 300$		
UB	time	迭代次数	$\frac{\lvert N^{final} \ s\rvert}{\lvert N_T\rvert}$	time	迭代次数	$\frac{\lvert N^{final} \ s\rvert}{\lvert N_T\rvert}$	time	迭代次数	$\frac{\lvert N^{final} \ s\rvert}{\lvert N_T\rvert}$
T^*	0.69	1.08	1.06	0.89	1.00	1.03	1.12	1.15	1.38
$1.5T^*$	0.88	1.10	1.05	1.56	1.15	1.27	1.20	1.24	1.14
$2T^*$	1.89	1.02	1.10	1.46	1.09	1.24	1.27	1.19	1.06

表 12：$m = 30$。

	$k = 200$			$k = 250$			$k = 300$		
UB	time	迭代次数	$\frac{\lvert N^{final} \ s\rvert}{\lvert N_T\rvert}$	time	迭代次数	$\frac{\lvert N^{final} \ s\rvert}{\lvert N_T\rvert}$	time	迭代次数	$\frac{\lvert N^{final} \ s\rvert}{\lvert N_T\rvert}$
T^*	1.34	1.12	1.07	1.36	1.14	1.08	1.43	1.19	1.11
$1.5T^*$	1.23	1.07	1.08	1.58	1.17	1.13	1.75	1.20	1.20
$2T^*$	1.01	1.03	1.07	2.03	1.17	1.15	1.32	1.10	1.10

表13：$m = 45$。

	k = 200			k = 250			k = 300		
UB	time	迭代次数	$\frac{\lvert N^{final}\ s\rvert}{\lvert N_T\rvert}$	time	迭代次数	$\frac{\lvert N^{final}\ s\rvert}{\lvert N_T\rvert}$	time	迭代次数	$\frac{\lvert N^{final}\ s\rvert}{\lvert N_T\rvert}$
T^*	1.08	1.02	0.99	1.51	1.20	1.10	1.08	1.05	1.00
$1。5T^*$	1.14	1.06	1.12	1.52	1.15	1.12	0.66	1.05	1.11
$2T^*$	1.16	0.98	1.08	1.15	1.02	1.06	0.90	1.05	1.12

表14：$m = 60$。

一致地，我们看到，两阶段 DDD 方法的平均迭代次数和 $|N_{final\,S}|/|N_T|$ 的平均值都在增加。迭代次数也呈现相同的趋势。

请注意，即使对于给定的混合整数规划，DDD算法定义良好，两阶段方法也未必定义良好。具体而言，如果不添加约束 (9)，向 D_S 添加定时节点可能会导致 UPR 的线性规划松弛值下降 。此外，细化过程可能会影响两阶段方法的有效性，因为在给定支撑集较大的分数解时，积极的细化过程若要修正解的支撑集中的所有短弧，可能会添加更多定时节点。因此，了解数学规划 formulation 和细化过程如何影响两阶段 DDD 方法的有效性，将会很有意思。