Advances in Applied Mathematics 应用数学进展, 2026, 15(3), 161-173
Published Online March 2026 in Hans. https://www.hanspub.org/journal/aam
https://doi.org/10.12677/aam.2026.153096

Hans汉斯

基于学习预测的最小两条节点不相交路径算法研究

朱世颖*，张淑蓉

太原理工大学数学学院，山西 太原

收稿日期：2026年2月4日；录用日期：2026年2月27日；发布日期：2026年3月6日

摘　要

在任意两个节点之间寻找多条节点不相交路径，是提升网络可靠性的基础任务。Suurballe算法为该问题提供了最优解，但该算法依赖重复的最短路径搜索，导致在大规模网络中计算成本高昂。本文提出一种新颖的预测式加速方法。我们引入一种双阶段策略(预测阶段和训练阶段)：预测阶段通过在反向图上进行双向搜索构建监督数据集，用于预测最优解的上界；训练阶段则通过使用并动态更新最优解的预测范围来剪枝搜索空间。该方法将预测机制与双向搜索整合至Suurballe型框架中，同时保证解的最优性。我们从理论上分析证明了算法的加速效果，并针对节点不相交路径问题设计了专用特征提取的方法与搜索协同策略。该框架为大规模网络中的高效多路径优化提供了新的方法论基础。

关键词

节点不相交路径，图算法，双向搜索，最短路径，互连网络

Accelerating Two Minimum Disjoint Paths Algorithm with Learned Predictions

Shiying Zhu*, Shurong Zhang

College of Mathematics, Taiyuan University of Technology, Taiyuan Shanxi

Received: February 4, 2026; accepted: February 27, 2026; published: March 6, 2026

Abstract

Finding multiple node-disjoint paths is a fundamental task for enhancing reliability in networks. While Suurballe's algorithm provleadan optimal solution for two such paths, its reliance on repeated

*通讯作者。

文章引用: 朱世颖, 张淑蓉. 基于学习预测的最小两条节点不相交路径算法研究[J]. 应用数学进展, 2026, 15(3): 161-173.
DOI: 10.12677/aam.2026.153096

shortest-path searches leads to high computational cost in large-scale networks. This paper proposes a novel prediction-guided acceleration approach. A hybrid two-phase strategy is introduced: in an offline learning phase, bidirectional search on a transformed graph collects dynamic search-state features to build a supervised dataset; during online inference, a lightweight trained model prunes the search space by predicting final path lengths from early-stage states. The method integrates prediction and bidirectional search into a Suurballe-type framework while preserving optimality guarantees. Theoretical analysis under a partial random model demonstrates the acceleration effect, and dedicated feature extraction and search coordination strategies are designed for the node-disjoint path problem. The proposed framework offers a new methodological foundation for efficient multi-path optimization in large-scale networks.

Keywords

Node-Disjoint Paths, Graph Algorithm, Bidirectional Search, Shortest Paths, Interconnection Networks

1. 引言

在通信网络、数据中心互联以及分布式计算系统中，确保关键节点对之间的可靠传输路径是一项基础且具有挑战性的任务。路径冗余[1]-[3]是一种被广泛采用的核心容错机制，它在同一源节点和目标节点之间建立多条逻辑上独立的传输路径。当某条路径因节点故障或网络拥塞而失效时，流量可以无缝切换到备用路径，从而维持服务的连续性。在此背景下，寻找多条除共同的起点和终点外不共享中间节点的节点不相交路径，已成为网络设计与优化的经典问题[4] [5]。其中，寻找总长度最小的两条节点不相交路径是最常见且最具代表性的场景，它不仅能直接服务于双路径保护和负载均衡，也是理解更一般多路径问题的基础。

对于最小成本节点不相交路径问题，Suurballe 于 1974 年提出了一个标志性的算法[6]。该算法以优雅而高效的方式解决问题：首先找到一条从源到目标的单条最短路径，然后构造一个巧妙的辅助图(即转换图)，并在其中寻找一条新的最短路径。这条新路径通过一种称为“交替”的操作与原路径结合，最终得到一对总长度最小的节点不相交路径。Suurballe 算法的理论完备性和实际有效性使其在数十年来成为该领域的标准解决方案[7]。然而，随着现代网络规模的迅速扩张，其固有的计算开销日益凸显。核心瓶颈在于，该算法需要在辅助图上进行一次完整的最短路径搜索，其时间复杂度与标准的 Dijkstra 算法相同。即使采用理论上最优的优先队列实现[8]，对于大型图或需要频繁进行路径计算的动态网络环境，其开销可能变得难以承受。

传统的加速方法，如双向搜索[9]和启发式剪枝[10]，在特定场景下可以提高效率，但往往缺乏理论保证或普适性[11]。新兴的“带预测的算法”范式为这一问题提供了新的视角[12]-[14]。该范式允许算法利用机器学习提供的预测来指导搜索过程，在预测准确时获得远超传统算法的性能，同时在预测错误时仍能保证基本的最坏情况性能。然而，现有的预测增强算法主要关注单源节点与目标节点集之间的最短路径问题[15]，尚无专门针对节点不相交路径问题定制的预测增强方案。类似地，尽管学习增强方法已在匹配等其他组合优化问题中展现出潜力[16]，但其在多路径路由中的应用仍待探索。

近年来，机器学习和预测机制已被引入经典算法的设计中以提升其效率。Feijen 和 Schäfer 提出的“带预测的 Dijkstra”算法[15]即为一例。该算法利用早期迭代中的路径特征来预测最终的最短路径长度，并基于此预测修剪不必要的边，从而减少优先队列操作次数。实验表明，该算法在随机图模型中显著降低了计算开销。同时，双向搜索作为一种经典的启发式策略[9]，已广泛应用于单源单目标最短路径问题。针对多目标场景的类似启发式方法也已被探索[17]，大幅缩小了搜索范围并提升了算法效率。

如何将学习增强方法的高效性与 Suurballe 算法求解节点不相交路径的框架进行深度融合，仍然是一个开放性问题。直接将预测机制应用于 Suurballe 算法的关键步骤面临独特挑战：辅助图的结构与权重动态地依赖于每次迭代中找到的路径，使得搜索过程的特征更加复杂多变。

因此，本文聚焦于最小两条节点不相交路径这一具体问题，主要贡献如下。

我们提出了一个新颖的框架，将学习增强的预测机制与 Suurballe 算法相融合，以加速求解总长度最小的两条节点不相交最短路径。在预测阶段，该框架采用双向搜索策略，高效地从源节点和目标节点两个方向收集路径状态数据。与传统的单向搜索相比，此方法在数据获取阶段显著减少了所需搜索空间，同时为预测模型的训练提供了更丰富、更均衡的拓扑特征数据集。在训练阶段，该框架能够早期识别并剪枝无希望的搜索分支。这种预测引导的搜索策略在保持原始 Suurballe 算法最优性保证的同时，显著提升了整体搜索效率，实现了兼具理论保证的实用加速。我们对所提框架的加速效果进行了理论分析。随后证明，即使是在最坏情况下，算法性能也会退化为经典的 Suurballe 算法，其时间复杂度仍为 $O(m+n\log n)$，其中 m 表示边数，n 表示节点数。该分析为设计具有可证明效率的学习增强路径优化算法提供了理论基础。

后续章节结构安排如下：第 2 章回顾必要的理论背景，涵盖 Suurballe 算法的关键步骤、双向搜索策略以及学习增强算法的基本原理。第 3 章详述我们提出的双阶段框架，包括训练阶段的双向搜索数据收集、预测模型设计以及应用阶段集成的预测搜索流程。第 4 章对所提方法的时间复杂度和加速效果进行理论分析。最后，第 5 章总结全文并提出未来研究方向。

2. 问题描述与分析

给定一个带权有向图 $G=(V,E)$，其中 V 是节点集，$|V|=n$；E 是边集，$|E|=m$；每条边 $e\in E$ 具有非负权重 $w(e)\in\mathbb{R}_{\geq 0}$。设源节点为 $s\in V$，目标节点为 $t\in V$。传统的最短路径问题旨在找到一条从 s 到 t 的路径 P，使得总权重 $w(P)=\sum_{e\in P}w(e)$ 最小。本文专注于最小两条节点不相交路径问题。

Table 1. Main symbols and description
表 1. 主要符号及解释

符号	解释
S	交错路径
$L_f[v]$，$L_b[v]$	在图 G 中从 s 到 v ($L_f[v]$)以及从 t 到 v ($L_b[v]$)的最短距离
$d_f(u)$，$d_b(u)$	节点 u 的前向/后向 d 值
B	不相交路径对总长度的上界，在搜索过程中更新
P	用于剪枝的交错路径预测长度
α，β	用于调整预测值 P 的膨胀因子
S_f，S_b	前向与后向双向搜索中已确定节点的集合

本节详细阐述最小两条节点不相交路径问题的模型与分析，重点在于如何通过引入 Suurballe 算法、

双向搜索及机器学习预测技术来优化路径计算。结合这些方法，我们旨在提高计算第二条路径的效率，尤其是在大规模图结构中。本文使用的主要符号如表 1 所示。

2.1. 基于预测的最小两条节点不相交路径问题

Suurballe 算法是求解节点不相交最短路径对问题的经典算法。其核心思想是构建第一条最短路径，然后利用一条交错路径来构建第二条最短路径，从而确保两条路径互不重叠，参见图 1。

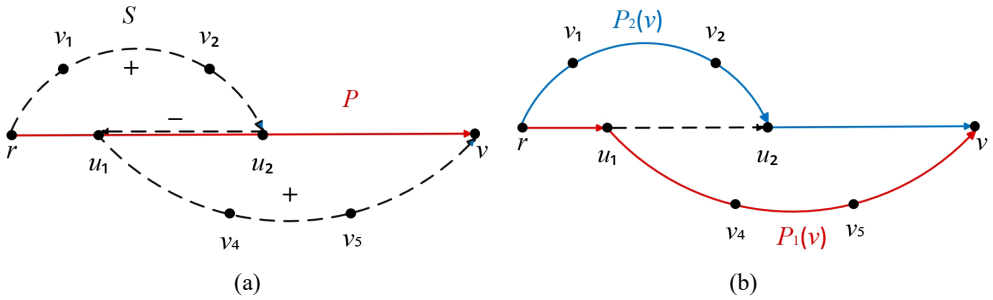

Figure 1. Interlacing path S and paths correction
图 1. 交错路 S 与路径修正

定义 2.1 (最小两条节点不相交路径)。给定一个带权图 $G=(V,E,w)$，其中 V 是顶点集，$E\subseteq V\times V$ 是弧集，$w:E\to\mathbb{R}_{\geq 0}$ 是非负权重函数。对于指定的源顶点 $s\in V$ 和目标顶点 $t\in V$，最小两条节点不相交路径(2-MNDP)是指从 s 到 t 的一对节点不相交路径 P_1 和 P_2，它们使得总长度 $w(P_1)+w(P_2)$ 最小，其中 P_1 和 P_2 除 s 和 t 外不共享任何顶点。

定义 2.1 (交错路径)。给定从源 s 到目标 t 的两条节点不相交路径 P_1 和 P_2，一条交错路径 S 是 s 和 t 之间的一条带符号标记的路径，满足以下条件：

1. S 中属于 P_1 或 P_2 的每条边都被赋予一个负号。
2. S 中位于 S 与 $\{P_1,P_2\}$ 交集上的每个顶点，在 S 中必须至少与一条带负号的边相关联。

Suurballe 算法的执行过程可以直观地理解为：基于已有路径，通过特定的“修正”操作来构建第二条不相交路径。具体而言，算法首先在图 G 中寻找一条从 s 到 t 的最短路径 P_1。这一步使用标准的 Dijkstra 算法实现；设 P_1 的长度为 L_1。

该算法的关键在于第二步：如何找到另一条与 P_1 不共享中间顶点，同时又能最小化 $L_1+w(P_2)$ 的路径 P_2。Suurballe 提出了一种基于路径差的构建方法。考虑两条路径 P_1 和一条候选路径 P_2，它们的对称差可以表示为一个有向子图，其中 P_2 中的边标记为正，P_1 中的边标记为负，而两条路径共享的边相互抵消。这个对称差图包含一条从 s 到 t 的有向路径，其长度等于 $w(P_2)-w(P_1)$ 的绝对值。

该算法的计算复杂度主要取决于两个搜索过程：第一个是时间复杂度为 $O(m+n\log n)$ 的标准 Dijkstra 搜索；第二个是改进的路径搜索。假设使用基于斐波那契堆的 Dijkstra 算法，每轮最短路径搜索的时间复杂度为 $O(m+n\log n)$，其中 $m=|E|$，$n=|V|$。因此，寻找 K 条路径的总时间复杂度为 $O\left(K\left(m+n\log n\right)\right)$。

2.2. 双向搜索策略

双向搜索作为图论中的经典算法增强技术，通过从源节点和目标节点同时启动搜索过程，打破了传统单向搜索的固有模式。在 Suurballe 算法的理论框架内，我们创新性地将该策略引入改进的路径搜索阶段，为求解节点不相交路径这一经典组合优化问题的效率提升开辟了新途径。

双向 Dijkstra 算法同时从源节点 s 和目标节点 t 开始搜索，直到两个搜索前沿相遇。设 $G=(V,E,w)$

为有向加权图，其反向图为$G'=(V,E',w')$，其中$E'=\{(v,u)\mid(u,v)\in E\}$，$w'(v,u)=w(u,v)$。搜索过程维护两个距离标签函数：$d_f:V\to\mathbb{R}_{\geq 0}\cup\{\infty\}$和$d_b:V\to\mathbb{R}_{\geq 0}\cup\{\infty\}$，分别表示从$s$出发的前向距离估计和从$t$出发的后向距离估计。同时还维护两个已确定节点的集合$S_f,S_b\subseteq V$，满足以下不变性：

1. 若$v\in S_f$，则$d_f(v)$为从s到v的最短距离。
2. 若$v\in S_b$，则$d_b(v)$为从v到t(在G中)的最短距离。

当存在顶点$u\in S_f\cap S_b$时，我们称u为相遇顶点。此时，可以构造出一条长度为$d_f(u)+d_b(u)$的s-t路径。特别地，若$u^*=\arg\min_{u\in S_f\cap S_b}\left[d_f(u)+d_b(u)\right]$，则对应的路径长度为当前已知最优上界。在结构均匀的图中，双向策略可将计算复杂度降低至单向搜索的约$1/2^{\alpha}$，其中指数α取决于图本身的拓扑特性。此外，双向框架自然地提供了早期剪枝的机会；从搜索两侧获得的中间结果可以相互共享和验证，使得潜在的搜索分支能够被更早地排除。

3. 启发式算法

本节在经典 Suurballe 算法[6]及近年来学习增强算法进展[15]的启发下，我们提出一个新颖的双阶段框架以加速计算最小两条节点不相交路径。该方法将双向搜索策略与机器学习预测相结合，旨在降低计算开销的同时保持最优性保证。

3.1. 算法框架

我们将通过以下步骤构建最小两条节点不相交路径，这些步骤将使用算法 1~5。

步骤 1. 我们首先采用算法 1，使用 Dijkstra 算法计算从源节点s到目标节点t的第一条最短路径P_1。令$L_1=w(P_1)$表示其长度。随后，我们在转换图 G'上同时从s和t执行双向搜索。在此搜索过程中，我们维护并更新每个节点v的d值，该值定义为当前路径长度与L_1的偏差。在每一步中，我们选择具有最小d值的节点进行扩展，优先选择那些保持接近最优解的路径。

步骤 2. 在算法 2 中，我们对前i_0次迭代执行双向搜索，始终选择具有最小d值的节点。后续操作遵循标准的 Suurballe 流程。当遇到一个节点u同时出现在前向和后向搜索前沿时，我们获得第二条不相交路径的长度。对于多组源–目标节点对，我们收集训练样本(X,y)，其中X包含记录的d值以及其他搜索状态特征。随后，我们基于此数据训练回归模型M。

步骤 3. 在预测阶段，我们采用算法 3 中预测引导的单向搜索。在每一步中，我们提取与训练阶段相同的特征X (尤其是d值)。查询训练好的模型M以获得预测的交错路径长度$\hat{L}$。随后，我们使用$\hat{L}$对搜索空间进行剪枝：跳过任何当前路径长度超过$\hat{L}$的节点v。每一步内的节点探索过程与 Suurballe 算法保持一致。搜索持续进行，直到找到一条长度接近$\hat{L}$的完整交错路径，或者剪枝后的搜索空间被完全探索。

步骤 4. 算法 4 构成了整个算法的核心松弛操作。对于每个新发现或距离更新的节点，该过程首先判断其距离是否改善了当前记录值。若是，则根据节点距离与预测值P的关系，将该节点分配至优先队列 PQ(若其距离未超过P)或储备集R(若其距离超过P)。对于已位于R中的节点，若其距离因后续搜索进展而降至P以下，则将其重新激活并移回 PQ。

步骤 5. 算法 5 用于处理初始预测可能不足的情况(例如，搜索完成但未找到有效的交错路径)。它通过将$\hat{L}$乘以系数$\beta>1$来动态调整预测，并重新激活被剪枝的节点。当获得第二条节点不相交路径且其总长度位于预测边界内时，算法终止。

3.2. 初始化

该初始化算法为后续的双向搜索预测阶段构建了必要的数学框架与数据结构。由于标准的 Dijkstra 算

法在找到目标节点 t 后不会立即停止，而是继续运行直至优先队列为空，从而自然计算出从源节点 s 到所有节点的最短距离。这一特性使得我们可以在一次搜索中同时获得 P_1 和 $L_f[\cdot]$。为此，算法 1 首先通过两次独立的 Dijkstra 搜索建立全局距离信息：第一次搜索(第 2 行)从源节点 s 出发，在计算到目标节点 t 的最短路径 P_1 及其长度 L_1 的同时，计算出从 s 到图中所有节点 v 的最短距离 $L_f[v]$；第二次在反向图 G_{rev} 上的搜索(第 3 行)从 t 出发，计算从每个节点 v 到 t 的最短距离 $L_b[v]$。这两个距离数组 L_f 和 L_b 分别刻画了节点在前向与反向拓扑结构中的可达性代价。

基于最短路径 P_1 的结构，算法随后(第 6~11 行)识别并排除关键节点：若路径 P_1 包含至少三个节点($|P_1| \geq 3$)，则将 s 之后的第一个节点作为 s_1，将 t 之前的最后一个节点作为 t_1；否则(在直接连接的特殊情况下)，设 $s_1 = t$，$t_1 = s$。

算法 1　初始化算法

输入：图 $G = (V, E, w)$，源节点 s，目标节点 t

输出：前向距离 L_f，后向距离 L_b，首尾节点 s_1, t_1，节点集 V_f, V_b，d 值 d_f, d_b

1: $P_1, L_1, L_f \leftarrow \text{Dijkstra}(G, s, t)$

2: $L_b[v] \leftarrow \text{Dijkstra}(G_{\text{rev}}, t), \forall v \in V$

3: if $|P_1| \geq 3$ then

4: $\quad s_1 \leftarrow v_1$

5: $\quad t_1 \leftarrow v_k$

6: else

7: $\quad s_1 \leftarrow t$

8: $\quad t_1 \leftarrow s$

9: end if

10: $V_f \leftarrow V \setminus s, s_1$

11: $V_b \leftarrow V \setminus t, t_1$

12: for each $u \in V_f$ then

13: $\quad$ if $u \in \partial(s)$ then

14: $\quad\quad d_f(u) \leftarrow w((s,u)) - L_f(u)$

15: $\quad$ else

16: $\quad\quad d_f(u) \leftarrow \infty$

17: $\quad$ end if

18: end for

19: for each $u \in V_b$ then

20: $\quad$ if $u \in \partial(t)$ then

21: $\quad\quad d_b(u) \leftarrow w((t,u)) - L_b(u)$

22: $\quad$ else

23: $\quad\quad d_b(u) \leftarrow \infty$

24: $\quad$ end if

25: end for

此构造得到两个节点集：前向集(包含除 s 和 s_1 外的所有节点)；以及后向集 $V_b = V \setminus \{t, t_1\}$ (包含除 t 和 t_1 外的所有节点)。这 $V_f = V \setminus \{s, s_1\}$ 两个集合定义了后续双向搜索中允许探索的节点范围。最后，算法初始化 V_f 和 V_b 中节点的 d 值(第 13~22 行)：对于 $u \in V_f$，若 u 是 s 的直接邻居，则其前向 d 值 $d_f(u) = w((s,u)) - L_f(u)$ (边权重 $w((s,u))$ 与前向最短距离 $L_f(u)$ 的差值，反映直接边与最短路径的成本

偏移)；否则，$d_f(u)=\infty$。类似地，对于$u\in V_b$，若$u\in\partial(t)$，则其后向d值$d_b(u)=w\big((t,u)\big)-L_b(u)$；否则，$d_b(u)=\infty$。这些$d$值将在后续搜索中作为启发式信息，引导算法优先探索偏离基线路径L_1最小的节点。

3.3. 双向搜索

本文提出的算法2实现了一种完全对称的探索策略，在每次迭代中交替进行前向和后向搜索。$L_{f'}[v]$和$L_{b'}[v]$用于跟踪更新后的距离，S_f和S_b则记录每个方向上已处理的节点。搜索最多进行i_0次迭代，除非更早发现相遇节点，否则每次迭代都执行前向和后向扩展步骤。算法从V_f中选择具有最小d_f值的节点u_f，计算其更新后的前向距离$L_{f'}[u_f]=d_f[u_f]+L_f[u_f]$，并将其加入前向已处理集合$S_f$ (第13~24行)。

算法2　双向搜索算法

输入：图$G=(V,E,w)$，s，t，前向距离L_f，后向距离L_b，d_f,d_b，V_f,V_b，s_1,t_1，早期停止迭代次数i_0

输出：记录的特征$X=d,B$

1: $\forall v\in V$，$L_{f'}[v]\leftarrow\infty$，$L_{b'}[v]\leftarrow\infty$，$L_{f'}[s]\leftarrow 0$，$L_{b'}[t]\leftarrow 0$

2: $S_f\leftarrow\varnothing$，$S_b\leftarrow\varnothing$，$X\leftarrow\varnothing$，$i\leftarrow 0$，$B\leftarrow\infty$

3: $meeting_{found}\leftarrow$ false

4: $meeting_{node}\leftarrow$ null

5: while $i<i_0$ and not $meeting_{found}$ then

6: $i\leftarrow i+1$

7: if $V_f\neq\varnothing$ then

8: $u_f\leftarrow\arg\min_{v\in V_f}d_f[v]$

9: $V_f\leftarrow V_f\setminus u_f$

10: $L_{f'}[u_f]\leftarrow d_f[u_f]+L_f[u_f]$

11: $S_f\leftarrow S_f\cup u_f$

12: 运行 Suurballe 算法步骤

13: $X.\text{add}(\text{"前向"},u_f,d_f[u_f])$

14: if $u_f\in S_b$ then

15: $D\leftarrow d_{f'}[u_f]+d_{b'}[u_f]$

16: $B\leftarrow L_{f'}[u_f]+L_{b'}[u_f]$

17: $meeting_{found}\leftarrow$ true

18: $meeting_{node}\leftarrow u_f$

19: $X.\text{add}(\text{"前向"},u_f,D)$，$X.\text{add}(\text{"B值"},B)$

20: 跳出循环

21: endif

22: endif

23: if $V_b\neq\varnothing$ and not $meeting_{found}$ then

24: $u_b\leftarrow\arg\min_{v\in V_b}d_b[v]$

25: $V_b\leftarrow V_b\setminus u_b$

26: $L_{b'}[u_b]\leftarrow d_b[u_b]+L_b[u_b]$

27: $S_b\leftarrow S_b\cup u_b$

28: 运行 Suurballe 算法步骤

29: $X.\text{add}(\text{"后向"},u_b,d_b[u_b])$

30: if $u_b \in S_f$ then
31: $D \leftarrow d_{f'}[u_b] + d_{b'}[u_b]$
32: $B \leftarrow L_{f'}[u_b] + L_{b'}[u_b]$
33: $meeting_{found} \leftarrow$ true
34: $meeting_{node} \leftarrow u_b$
35: X.add("相遇节点",u_b,D)，X.add("B值",B)
36: 跳出循环
37: end if
38: end if
39: end while

随后，它调用 Suurballe 算法过程，根据当前搜索状态调整边权重并管理路径转换。在处理 u_f 之后立即检查相遇条件：如果 u_f 已存在于 S_b 中，则从前向视角看已经相遇。此时，算法计算 $D = d_{f'}[u_f] + d_{b'}[u_f]$ 和完整的交替路径长度 $B = L_{f'}[u_f] + L_{b'}[u_f]$，将这些值连同相遇节点信息记录到特征集 X 中，并终止搜索。

如果在向前阶段未检测到相遇，算法则进入向后阶段：从 V_b 中选择具有最小 d_b 值的节点 u_b，将其向后距离更新为 $L_{b'}[u_b] = d_b[u_b] + L_b[u_b]$，并将其加入 S_b (第 26~37 行)。应用相同的 Suurballe 算法\cite{Suurballe1974}过程，并进行对称的相遇检查：如果 u_b 已存在于 S_f 中，则从后向视角看已经相遇。算法随后计算相应的 D 和 B 值，记录相遇信息并终止。

3.4. 训练阶段

算法 3 在给定的图 G、源节点 s 与目标节点 t 上运行。它首先执行标准的 Dijkstra 算法以计算第一条最短路径 P_1 及其长度 L_1。主要的搜索过程在一个循环中展开，只要优先队列非空且队列中的最小优先级不超过当前预测值 P，该循环就会持续。

在每次迭代中，从队列中提取具有最小距离标签的节点 u。如果 u 是目标节点 t，算法将记录发现的路径长度 L_{found} 并跳出循环，这标志着已找到一条从 s 到 t 的完整路径。在前 i_0 次迭代中，算法记录搜索轨迹特征：对于每次迭代，当前节点的距离 $d(u)$ 和当前上界 B 作为一个数据对存储在特征数组 $X[i]$中。当迭代计数器恰好达到 i_0 时，算法调用预测函数 PREDICTION，基于已收集的特征 X 生成初始路径长度预测。该值随后乘以膨胀因子 α $(\alpha \geq 1)$以得到正式的预测值 P。

接着，对于当前节点 u 的每条出边 (u,v)，算法计算暂定距离 $tent = d(u) + L(u)$。随后调用 Suurballe 算法步骤以执行 Suurballe 算法特有的边权重调整和交错路径构建——这是确保最终找到节点不相交路径的关键步骤，与标准的 Suurballe 方法保持一致。如果 $tent$ 超过了当前上界 B，则对该边进行剪枝，因为它无法产生比当前已知最优路径更短的解。如果目标节点 t 可通过此边到达，则更新上界：$B = \min\{tent, B\}$。算法随后运行算法 4，该算法根据 $tent$ 与预测值 P 的关系，决定节点 v 应插入优先队列 PQ 还是储备集 R：若 $tent \leq P$，则插入 PQ；否则，放入 R。在循环结束时，算法 5 过程处理预测值低估真实路径长度的情况。

搜索结束后，获得真实的最短路径长度 L_{true}。特征数组 X 与此真实长度 $y = L_{\text{true}}$ 配对形成训练样本 (X, y)，并添加至训练数据集 D_{train}。通过对大量不同的图实例和源－目标节点对重复此过程，积累了丰富的训练数据。最后，在完整数据集 D_{train} 上训练回归模型 M。训练好的模型 M 随后在预测阶段被部署，以快速估计新实例的交错路径长度，从而实现算法的整体加速。

算法 3　训练阶段算法

输入：图 $G=(V,E,w)$，源节点 s，目标节点 t

输出：训练数据集 $D_{\text{train}}=(X_j,y_j)$，其中 X_j 为特征，y_j 为真实最短路径长度

$P_1,L_1 \leftarrow \text{Dijkstra}(G,s,t)$

$d(s)=0$，$d(v)=\infty$ 对于所有 $v\in V\setminus s$

$D_{\text{train}} \leftarrow \varnothing$，$B=\infty$，$P=\infty$，$X=[\,]$，$i=0$

PQ.INSERT$(s,d(s))$，$R=\varnothing$

while PQ.非空() 且 PQ.最小优先级() $\le P$ then

　$u=$ PQ.REMOVE-MIN()

　if $u=t$ then

　　$L_{\text{found}}=d(u)$

　　跳出循环

　end if

　$i=i+1$

　if $i\le i_0$ then

　　$X[i]=(d(u),B)$

　end if

　if $i=i_0$ then

　　$P=\alpha\cdot\text{PREDICTION}(X)$

　end if

　for each $(u,v)\in E$ then

　　$tent=d(u)+L(u)$

　　运行 Suurballe 算法步骤

　　if $tent>B$ then 继续 end if

　　if $v=t$ then

　　$B=\min tent,B$

　　end if

　　运行算法 4

　end for

　运行算法 5

end while

$y=L_{\text{true}}$

$D_{\text{train}} \leftarrow D_{\text{train}}\cup(X,y)$

使用回归方法在 D_{train} 上训练预测模型 M

3.5. RELAX-PREDICTION 过程

RELAX-PREDICTION 过程是训练算法中的核心松弛操作，它根据暂定距离 *tent* 和当前预测值 P 动态管理节点的插入与更新策略。当通过边 (u,v) 发现的暂定距离 *tent* 改进了节点 v 当前记录的距离 $d(v)$ 时，该过程被触发。它更新 v 的距离信息并调整其在搜索数据结构中的位置。

首先，将 *tent* 与 $d(v)$ 进行比较。若 $tent<d(v)$，则执行更新：记录旧距离值 d_{old}，并将 $d(v)$ 更新为 *tent*。随后根据 v 之前的状态采取不同的操作。如果 v 是首次被发现（$d_{\text{old}}=\infty$），则根据 *tent* 与预测值 P 的关系决定其放置位置。若 $tent\le P$，则将 v 插入优先队列 PQ 以继续参与当前轮次的搜索；否则，将其插入储备集 R 进行暂存。

算法 4　RELAX-PREDICTION 算法

1: if $tent < d(v)$ then
2: $d_{\text{old}} \leftarrow d(v)$
3: $d(v) \leftarrow tent$
4: if $d_{\text{old}} = \infty$ then
5: if $tent \le P$ then
6: PQ.INSERT$(v, tent)$
7: else
8: R.INSERT$(v, tent)$
9: end if
10: else
11: if $v \notin R$ then
12: PQ.DECREASE-PRIORITY$(v, tent)$
13: else
14: if $tent > P$ then
15: R.DECREASE-PRIORITY$(v, tent)$
16: else
17: R.REMOVE(v)
18: PQ.INSERT$(v, tent)$
19: end if
20: end if
21: end if
22: end if

如果 v 是一个已被发现的节点($d_{\text{old}} \neq \infty$)，则进一步区分其当前位置。当 v 不在储备集 R 中时，直接调用 PQ.DECREASE-PRIORITY 来更新其在优先队列中的优先级。如果 v 已经在 R 中，则根据更新后的 $tent$ 与 P 的关系进行处理：当 $tent > P$ 时，仅更新该节点在 R 中的优先级；当 $tent \le P$ 时，表明该节点已进入预测路径长度范围，则将其从 R 中移除并插入 PQ，使其能够重新加入主搜索过程。

3.6. UPDATE-PREDICTION 过程

UPDATE-PREDICTION 过程在优先队列变为空而搜索尚未完成时被调用，这表明当前的预测值 P 可能低估了真实路径长度。该过程通过增大预测值并重新激活储备集中符合条件的节点，使搜索得以继续。

算法 5　UPDATE-PREDICTION 算法

1: $P \leftarrow \beta \cdot P$
2: for each 节点 $v \in R$ then
3: if $d(v) \le B$ then
4: R.REMOVE(v)
5: PQ.INSERT$(v, d(v))$
6: end if
7: end for

首先，将当前预测值 P 乘以一个膨胀因子 β ($\beta > 1$)，得到一个新的预测值。随后，遍历储备集 R 中的所有节点，并检查每个节点 v 的距离标签 $d(v)$ 是否满足激活条件：当且仅当 $d(v) \le B$ 时，即节点的当

前距离不超过已知的最优上界 B，该节点被认为可能构成更优路径的一部分。满足此条件的节点将从 R 中移除，并被插入优先队列 PQ，从而重新加入主搜索过程。

3.7. 时间复杂性分析

本文提出的加速 Suurballe 算法在时间复杂度方面保持了严格的理论保证。

定理 1. 在最坏情况下，即预测完全不准确时，该算法退化为经典的 Suurballe 算法，其时间复杂度保持为 $O(m+n\log n)$，其中 m 表示边数，n 表示节点数。

证明： 当预测值 P 完全错误(例如 $P=0$)或预测机制被禁用时，该算法退化为结合了 Dijkstra 剪枝技术的标准 Suurballe 算法。此时，第一次 Dijkstra 搜索计算第一条最短路径 P_1 的时间为 $O(m+n\log n)$。构建辅助图 G'需要遍历所有边，耗时 $O(m)$。第二次搜索在 G'上执行标准的 Dijkstra 算法，同样耗时 $O(m+n\log n)$。UPDATE-PREDICTION 过程最多被调用 $O(\log W)$ 次，其中 W 为最大边权重，每次调用处理 $O(n)$ 个节点，总成本为 $O(n\log W)$。由于在边权重多项式有界的假设下 $\log W=O(\log n)$，因此最坏情况下的总时间复杂度为：

$$T_{\text{worst}}=2\cdot O(m+n\log n)+O(m)+O(n\log n)=O(m+n\log n).$$

优先队列的斐波那契堆实现支持每次 Dijkstra 搜索在 $O(m+n\log n)$ 时间内完成[18]，而 Suurballe 算法需要进行两次这样的搜索。尽管 UPDATE-PREDICTION 过程可能引入额外开销，但储备集 R 的大小永远不会超过节点总数 n，且每次更新的成本可被限定为 $O(|R|)$。综合这些因素，该算法保持了与经典 Suurballe 算法相同的最坏情况时间复杂度 $O(m+n\log n)$。

4. 理论分析

本节提供关于解最优性的理论证明。Suurballe 算法作为最小和节点不相交路径问题的经典精确解，自 1974 年以来其正确性已被严格证明。我们所引入的预测加速机制，本质上是将一种基于机器学习预测的智能剪枝策略整合到了 Suurballe 算法的搜索过程中。为分析该策略在随机网络拓扑下的预期性能，我们采用经典的随机图模型[19]来描述图的生成分布，并在此基础上讨论算法的期望加速效果。

4.1. 算法的最优性保证

令源节点为 $s\in V$，目标节点为 $t\in V$，图 $G=(V,E,w)$ 的边权重函数为 $w:E\to\mathbb{R}_{\geq 0}$。记 P_1^*,P_2^* 为总权重最小的节点不相交路径对，其目标函数值为：

$$L_{\min}=\min_{P_1,P_2\in\mathcal{P}_{s,t}^{\text{disj}}}\left[w(P_1)+w(P_2)\right],$$

其中 $\mathcal{P}_{s,t}^{\text{disj}}$ 表示所有 s-t 节点不相交路径对的集合。

该算法使用一个预测值 P 来指导搜索，该预测值满足条件 $P\geq D$，其中 $D=w(P_1^*)+w(P_2^*)-w(P_1)$ 是第二条路径的可能最小长度(第一条路径 P_1 已通过标准 Dijkstra 预先计算)。令 α 为膨胀因子；实际使用的预测边界为 $P'=\alpha\cdot P$。对于任何边 $(u,v)\in E$，其暂定距离 $tent(u,v)=d(u)+w(u,v)$ 在满足以下条件时被剪枝：

$$tent(u,v)>\min\{B,P'\},$$

其中 B 是当前最佳上界。由 $P'\geq D$ 可知，满足 $tent(u,v)\leq D$ 的边将不会被剪枝，并且 P_1^* 和 P_2^* 上的所有边均满足此条件(根据 Suurballe 算法的构建原理)。因此，该算法保证在有限步内找到最优路径对：

$$\lim_{k\to\infty}\Pr\left(\left(P_1^{(k)},P_2^{(k)}\right)=\left(P_1^*,P_2^*\right)\right)=1,$$

其中 $\left(P_1^{(k)}, P_2^{(k)}\right)$ 是在第 k 次迭代中获得的路径对。

4.2. 预测错误下的鲁棒性分析

预测机制的鲁棒性对于算法的实际效用至关重要。我们的设计采用多层保护机制，以确保即使在预测不准确时，算法依然可靠。首先，通过膨胀因子 α 对初始预测值进行保守调整，避免因低估实际距离而导致过早剪枝。其次，算法持续监控搜索状态，并在怀疑预测值不足时自动触发 UPDATE-PREDICTION 过程，通过因子 β 逐步放宽预测边界。这种渐进调整策略能够在处理预测误差的同时，避免搜索空间的急剧扩张。更重要的是，即使在极端情况下，算法最终将回退到不使用预测的标准搜索过程，从而保证了解的存在性和最优性。这种分层鲁棒性设计使得算法能够适应不同质量的预测信息，在预测准确时实现最大加速，在预测较差时保持基线性能。

4.3. 计算效率与对比指标的理论分析

为全面评估所提算法的性能，我们引入以下三项理论对比指标，并给出其形式化定义与计算方式：

1. 总运行时间：设 T_{init} 为算法初始化阶段(包括第一次 Dijkstra 搜索与距离数组计算)的时间开销，T_{pred} 为预测模型的推理时间，T_{search} 为第二条路径的搜索时间。算法的总运行时间 T_{total} 定义为：

$$T_{\text{total}} = T_{\text{init}} + T_{\text{pred}} + T_{\text{search}},$$

其中，T_{search} 依赖于预测质量：当预测准确时，剪枝效果显著，T_{search} 可大幅降低；预测失效时，T_{search} 退化为经典 Suurballe 算法的搜索时间 $T_{\text{Suurballe}}$。

2. 扩展节点数：令 V_{exp} 表示在第二条路径搜索过程中从优先队列PQ中弹出并进行松弛操作的节点集合。扩展节点数 N_{exp} 定义为：

$$N_{\text{exp}} = \left|V_{\text{exp}}\right|,$$

在预测引导的剪枝机制下，N_{exp} 满足：

$$N_{\text{exp}} \le N_{\text{Suurballe}},$$

其中 $N_{\text{Suurballe}}$ 为经典 Suurballe 算法在相同实例上的扩展节点数。

3. 加速比：加速比 S 用于量化本算法相对于基准方法的性能提升，定义为经典 Suurballe 算法总运行时间 $T_{\text{Suurballe}}$ 与本算法总运行时间 T_{total} 之比：

$$S = \frac{T_{\text{Suurballe}}}{T_{\text{total}}}.$$

当 $S > 1$ 时，表明本算法实现了加速；$S = 1$ 时，性能与基准持平；$S < 1$ 时，性能下降(仅发生于预测严重误导且额外开销显著的情形)。在预测准确时，$T_{\text{search}} \ll T_{\text{Suurballe}}$，且 T_{pred} 通常可忽略，此时 $S \gg 1$，体现出预测机制的有效性。在最坏情况下，算法退化为 Suurballe 过程，此时 $S \approx 1$，保证了算法的鲁棒性。这些指标共同构成了评估算法效率与预测引导效果的理论基础，并为后续实验验证提供了明确的量化方向。

5. 结论

本文是将新兴的“带预测的算法”范式首次应用于多路径优化问题，相较于传统的启发式方法，我们创新性地将机器学习预测、双向搜索与经典组合优化算法进行了深度融合与协同。这种多策略协作的框架为解决其他复杂优化问题提供了借鉴模型，展现了不同技术间互补增强的潜力。然而，本研究目前

仍存在一定局限性，首先，算法初始化阶段需两次 Dijkstra 搜索，未来的工作可探索更轻量化的距离估计方法，或针对特定图结构(如层次化网络)设计专用初始化策略。其次所提出的方法尚未通过实际仿真实验验证其性能与鲁棒性，缺乏在不同规模与拓扑结构网络中的实证评估。在未来的研究中，应当引入大规模仿真与真实网络数据实验，理论对比指标(总运行时间、扩展节点数、加速比)为实验评估提供了明确方向。总结而言，本文提出的算法不仅为提升 Suurballe 算法的计算效率提供了理论思路，也为应对复杂的网络优化挑战提供了新的方法论视角与技术路径。我们相信，这种融合机器学习与经典算法的研究方向，未来将在网络科学、运筹学等领域产生广泛影响。

基金项目

山西省基础研究计划项目(202103021224058)。

参考文献

[1] Schrijver, A. (2003) Combinatorial Optimization: Polyhedra and Efficiency. Springer.

[2] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B. (1993) Network Flows: Theory, Algorithms, and Applications. Prentice Hall.

[3] Medard, M., Finn, S.G., Barry, R.A. and Gallager, R.G. (1999) Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs. *IEEE/ACM Transactions on Networking*, **7**, 641-652. https://doi.org/10.1109/90.803380

[4] Ramasubramanian, S., Krishnamoorthy, H. and Krunz, M. (2007) Disjoint Multipath Routing Using Colored Trees. *Computer Networks*, **51**, 2163-2180. https://doi.org/10.1016/j.comnet.2006.09.019

[5] Watel, D. and Weisser, M. (2016) A Practical Greedy Approximation for the Directed Steiner Tree Problem. *Journal of Combinatorial Optimization*, **32**, 1327-1370. https://doi.org/10.1007/s10878-016-0074-0

[6] Suurballe, J.W. (1974) Disjoint Paths in a Network. *Networks*, **4**, 125-145. https://doi.org/10.1002/net.3230040204

[7] Suurballe, J.W. and Tarjan, R.E. (1984) A Quick Method for Finding Shortest Pairs of Disjoint Paths. *Networks*, **14**, 325-336. https://doi.org/10.1002/net.3230140209

[8] Fredman, M.L. and Tarjan, R.E. (1987) Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the ACM*, **34**, 596-615. https://doi.org/10.1145/28869.28874

[9] Goldberg, A.V. and Harrelson, C. (2005) Computing the Shortest Path: A* Search Meets Graph Theory. *Proceedings of the* 16*th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), Vancouver, 23-25 January 2005, 156-165.

[10] Sanders, P. and Schultes, D. (2007) Engineering Highway Hierarchies. *Journal of Experimental Algorithmics*, **12**, 1.6:1-1.6:29.

[11] Madkour, A., Aref, W.G., Rehman, F.U., Rahman, M.A. and Basalamah, S. (2022) A Survey of Shortest-Path Algorithms. *ACM Computing Surveys*, **55**, 129:1-129:36.

[12] Bast, H., Mehlhorn, K., Schäfer, G. and Tamaki, H. (2003) A Heuristic for Dijkstra's Algorithm with Many Targets and Its Use in Weighted Matching Algorithms. *Algorithmica*, **36**, 75-88. https://doi.org/10.1007/s00453-002-1008-z

[13] Mitzenmacher, M. and Vassilvitskii, S. (2022) Algorithms with Predictions. *Communications of the ACM*, **65**, 33-35. https://doi.org/10.1145/3528087

[14] Bengio, Y., Lodi, A. and Prouvost, A. (2021) Machine Learning for Combinatorial Optimization: A Methodological Tour D'horizon. *European Journal of Operational Research*, **290**, 405-421. https://doi.org/10.1016/j.ejor.2020.07.063

[15] Chen, J., Silwal, S., Vakilian, A. and Zhang, F. (2022) Faster Fundamental Graph Algorithms via Learned Predictions. *Proceedings of the* 39*th International Conference on Machine Learning* (*ICML*), Vol. 162, 3583-3602.

[16] Feijen, W. and Schäfer, G. (2024) Dijkstra's Algorithm with Predictions to Solve the Single-Source Many-Targets Shortest-Path Problem. *SIAM Journal on Computing*, **53**, 463-492.

[17] Dinitz, M., Im, S., Lavastida, T., Moseley, B. and Vassilvitskii, S. (2021) Faster Matchings via Learned Duals. *Advances in Neural Information Processing Systems* 34 (*NeurIPS* 2021), 6-14 December 2021, 10393-10406.

[18] Thorup, M. (1999) Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time. *Journal of the ACM*, **46**, 362-394. https://doi.org/10.1145/316542.316548

[19] Gilbert, E.N. (1959) Random Graphs. *The Annals of Mathematical Statistics*, **30**, 1141-1144. https://doi.org/10.1214/aoms/1177706098