Yang, Lu; Yang, Zhouwang

Article

An advanced Successive Derivative Shortest Path algorithm for concave cost network flow problems

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Yang, Lu; Yang, Zhouwang (2025) : An advanced Successive Derivative Shortest Path algorithm for concave cost network flow problems, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 14, pp. 1-13, https://doi.org/10.1016/j.orp.2025.100331

This Version is available at:
https://hdl.handle.net/10419/325808

Operations Research Perspectives 14 (2025) 100331

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

An advanced Successive Derivative Shortest Path algorithm for concave cost network flow problems☆

Lu Yang, Zhouwang Yang *

University of Science and Technology of China, Hefei, PR China

ARTICLE INFO

Dataset link: https://github.com/lufizyang/Data-and-Code-for-CCNFP

Keywords:
Network flow
Non-convex problem
Approximation algorithm
Regional first-order information
Interval reduction

ABSTRACT

As production scales up, transportation networks increasingly involve nonlinear costs, leading to the concave cost network flow problem (CCNFP), which is notably challenging due to its nonlinearity. Existing nonlinear programming methods addressing the CCNFP often suffer from low efficiency and high computational cost, limiting their practical application. To overcome these limitations, this paper proposes the Successive Derivative Shortest Path (SDSP) algorithm, an efficient approach that combines a sequential linear approximation framework with regional first-order information of the objective function. By integrating regional first-order information and employing an interval reduction mechanism, the SDSP algorithm effectively avoids premature convergence to suboptimal solutions, thereby achieving higher-quality solutions. Numerical experiments, including parameter selection, validation, and comparative analysis, demonstrate that the SDSP algorithm outperforms existing methods in terms of both solution quality and convergence speed. This research offers a robust and efficient solution for the CCNFP, with potential applications in various fields, including logistics and supply chain networks, where concave cost network flow issues are common.

1. Introduction

As societal demands evolve and production processes grow in complexity, the need for optimizing resource allocation to maximize economic efficiency has become increasingly critical. The minimum-cost flow problem [1], a foundational model in optimization and resource allocation, has seen broad applications across diverse practical fields [2–4]. However, endogenous factors in real-world applications often introduce nonlinear cost structures, such as economies of scale [5] and risk-related costs [6], making these problems challenging for traditional minimum-cost flow algorithms [7–10] to address effectively. Consequently, concave cost network flow problems have emerged as a central focus within the optimization research community [11–13].

The Concave Cost Network Flow Problem (CCNFP) is a specialized variant of the minimum-cost flow problem[1] characterized by concave objective functions. Solving the CCNFP aims to identify feasible flows in a network that minimize the overall objective value. The inclusion of concave objectives enhances the CCNFP's modeling flexibility, making it applicable to a wide range of scenarios in transportation [14–16] and warehousing [17,18]. For instance, the most reliable path problem [19] is essential for determining optimal travel routes and departure times, with reliability quantified by the standard deviation of travel time—a nonlinear component in the objective function. In warehousing, the joint location-inventory problem [6] addresses demand fluctuation risks by minimizing the standard deviation of product demand to reduce reliability costs. While concave costs provide richer modeling capabilities, they also introduce significant computational challenges, especially given the increasing complexity and scale of network topologies. The CCNFP is known to be NP-hard [18], with complexity stemming from the fact that minimizing a concave cost over a convex feasible region does not guarantee finding a global optimum [20]. This NP-hard nature presents challenges in developing efficient algorithms, particularly when attempting to find optimal or near-optimal solutions within a reasonable time frame. Consequently, research into high-quality, efficient algorithms for solving the CCNFP is essential for both theoretical advancements and practical applications.

Existing research has yet to develop direct algorithms specifically for solving the CCNFP. Current approaches often simplify the network structure or treat it as a general nonlinear programming problem. The

☆ **Funding:** The work is supported by the NSF of China (Nos. 92270205, 12301659, 12171453), the National Key R&D Program of China (Nos. 2022YFA1005201, 2022YFA1005202, 2022YFA1005203), and the Major Project of Science and Technology Innovation Tackling Plan of Anhui Province (No. 202423e09050003).

* Corresponding author.
E-mail addresses: yl0501@mail.ustc.edu.cn (L. Yang), yangzw@ustc.edu.cn (Z. Yang).

[1] In this article, the term "minimum-cost flow problem" refers specifically to its form with linear objective functions.

https://doi.org/10.1016/j.orp.2025.100331
Received 23 November 2024; Received in revised form 19 January 2025; Accepted 11 February 2025
Available online 19 February 2025

Nomenclature

α	Lower bound of sampling interval
β	Upper bound of sampling interval
w	Average derivative value
κ	Unbalanced state of node
π	Potential function
x	Flow on edge

Acronyms

CCNFP	Concave cost network flow problem
AugLag	Augmented Lagrange function algorithm
Penalty	Penalty function algorithm
SDSP	Successive derivative shortest path
SGSP	Successive gradient shortest path
SLSQP	Sequential least square programming
SSP	Successive shortest path
ARE	Average relative error
MRE	Maximum relative error

Parameters

G	Graph/network structure
$\bar{G}$	Residual graph
V, S, D	Node set
$E, \bar{E}$	Edge set
v, l, k	Node
e	Edge
u	Finite capacity on edge
r	Residual capacity on $\bar{G}$
b	Supply on node
c	Cost function on edge
P	Path
a	Interval reduction coefficient
M	Maximum iteration count
n_s	Number of sampling points

concave cost transportation problem, for instance, is a special case of the CCNFP formulated on a bipartite graph [11]. Additionally, some studies have focused on variants of the CCNFP with single-source, uncapacitated edges [20,21]. However, these simplified graph structures overlook the complexities inherent in real-world networks, limiting the practical applicability of such algorithms. For example, bipartite graphs assume a structure consisting solely of sources and sinks, an assumption that rarely holds in realistic settings. Even with these simplifications, solutions to such cases typically rely on metaheuristic methods [21–24], which often require substantial time to find suboptimal solutions without guarantees of convergence, even for moderately sized problems. Furthermore, since the CCNFP is inherently a nonlinear programming problem, established nonlinear programming algorithms, such as augmented Lagrangian methods and sequential least squares programming [12,25], can be applied. However, these methods face challenges due to the lack of theoretical guarantees for finding exact solutions and their typically prolonged solving times, making them less suitable for large-scale or time-sensitive applications.

In this paper, we propose a novel sequential reduction algorithm, the Successive Derivative Shortest Path (SDSP) algorithm, which combines the Successive Shortest Path (SSP) algorithm [10] with first-order information of concave objective functions. While SSP is an exact algorithm for solving minimum-cost flow problems, it cannot be directly applied to the CCNFP. To address this, we approximate the concave cost function in the CCNFP by utilizing regional first-order information, thereby enabling an approximate solution to the problem. This approach allows us to establish a sequential process that generates a series of minimum-cost flow subproblems, each step incrementally optimizing the CCNFP solution. Additionally, by progressively reducing the interval size for calculating first-order information, the SDSP algorithm ensures convergence and mitigates the risk of premature convergence to suboptimal solutions. Theoretical analysis and validation experiments confirm that the sequential reduction process effectively directs the algorithm toward higher-quality solutions. Comparisons with other algorithms demonstrate the superior solution quality and convergence rate achieved by the SDSP algorithm.

The remainder of this paper is organized as follows. Section 2 formally defines the concave cost network flow problem (CCNFP). Section 3 presents the proposed algorithm in detail, followed by Section 4, which describes and discusses the simulation results. Finally, Section 5 concludes the paper with a summary of the findings and implications.

2. Formulation of problems

The Concave Cost Network Flow Problem (CCNFP) is a nonlinear programming problem formulated on a graph structure with a concave objective function and subject to linear constraints.

Let $G(V, E)$ be a directed graph with a set of n nodes $V = \{v_i\}_{i=1}^{n}$ and a set of m directed edges E. Each node v_i has an associated supply b_i, which is used to partition the node set V into the three subsets: the source node set $S = \{v_i \in V \mid b_i > 0\}$, the intermediate node set $V_{in} = \{v_i \in V \mid b_i = 0\}$, and the sink node set $D = \{v_i \in V \mid b_i < 0\}$. Each edge e_{ij} represents a directed connection from node v_i to node v_j. x_{ij} is a non-negative real number that denotes the amount of flow through the edge e_{ij}, typically subject to a finite capacity u_{ij}. In the CCNFP, each edge e_{ij} has an associated concave cost function $c_{ij}(x_{ij})$, which depends on the flow x_{ij}.

Similar to the minimum-cost flow problem, the Concave Cost Network Flow Problem (CCNFP) also enforces that the sum of the net flow and supply at each node equals zero, ensuring flow conservation across the network. This constraint can be written as

$$\sum_{j \in V_i^-} x_{ji} - \sum_{j \in V_i^+} x_{ij} + b_i = 0, \forall i \in V \tag{1}$$

where $V_i^- = \{j \in V : e_{ji} \in E\}$ and $V_i^+ = \{j \in V : e_{ij} \in E\}$. Additionally, the flow value on each edge is constrained within the range from zero to its capacity, i.e., $0 \leqslant x_{ij} \leqslant u_{ij}$. A solution that satisfies both of these constraints is a feasible solution to the CCNFP. Thus, the concave cost network flow problem can be formulated as

$$\begin{aligned} \min_{x} \quad & C(x) = \sum_{e_{ij} \in E} c_{ij}(x_{ij}) \\ \text{s.t.} \quad & \sum_{j \in V_i^-} x_{ji} - \sum_{j \in V_i^+} x_{ij} = b_i, \forall i \in V \\ & 0 \leqslant x_{ij} \leqslant u_{ij}, \forall e_{ij} \in E \end{aligned} \tag{2}$$

The formulation of CCNFP is a generalized definition of the network flow problems when Model (2) does not restrict its objective function type. The minimum-cost flow problem arises as a special case when the objective function simplifies to a linear function passing through the origin. In cases where the linear objective exhibits jump discontinuities at the origin, the problem transforms into a fixed-charge network flow problem [26]. The constraints are identical, establishing an equivalence in the feasible domains between the CCNFP and the minimum-cost flow problem on the same graph structure. This forms the basis for constructing a sequence of minimum-cost flow problems to approximate the CCNFP.

The concepts outlined in the definition of CCNFP have direct correspondences with real-world scenarios. In logistics and transportation, the storage points of goods correspond to the source nodes in the graph, the demand points for goods correspond to the sink nodes, and the

transit points correspond to the intermediate nodes. The transportation routes naturally establish connections between these nodes. Goods transportation involves moving goods from storage points to demand points, often characterized by multi-sourcing and multi-tier transportation tasks. Additionally, large-scale or long-term transportation tasks, due to decreasing marginal costs [27], result in objective functions exhibiting concave characteristics. Therefore, CCNFP is particularly well-suited for application in such scenarios.

3. Sequential reduction algorithm

In this section, we detail the SDSP algorithm, covering the foundational principles of the SSP algorithm and the sequential reduction mechanism that distinguishes SDSP.

3.1. Successive shortest path

The SSP algorithm finds the optimal solution for the minimum-cost flow problem by iteratively searching for the shortest path that balances supply and demand nodes between source and sink nodes in the residual graph. The residual graph $\bar{G}(V,\bar{E})$ is constructed based on $G(V,E)$ by adding the reverse edges corresponding to each original edge in E. The set of reverse edges is denoted as $E'=\{e'_{ji}:e_{ij}\in E\}$, where the edges e'_{ji} and e_{ji} are not the same. Therefore, the edge set $\bar{E}$ can be represented as $\bar{E}=E\cup E'$. During the solving process, flows on reverse edges $\{x_e\}_{e\in E'}$ represent the algorithm's ability to backtrack previously allocated flows. The unit cost of the reverse edge is the opposite number of the unit cost of the original edge, and the capacity of the reverse edge is the allocated flow on the original edge. Thus, the residual capacity in $\bar{G}(V,\bar{E})$ can be written as

$$\begin{cases} r_e = u_e - x_e & , \quad e\in E \\ r_{e'} = x_e & , \quad e'\in E' \end{cases} \tag{3}$$

where the residual capacity of reverse edges e' is the allocated flow x_e on the original edge. SSP identifies the shortest path between a pair of source and sink nodes within the current residual graph, assigns the maximum feasible flow in each iteration, and repeats the above operation until all nodes satisfy the conservation condition. The unbalanced state $\kappa(i)$, which characterizes the change of node supply in iterations, is defined as

$$\kappa(i)=b(i)+\sum_{e\in E_i^-}x_e-\sum_{e\in E_i^+}x_e,\ \text{for all } i\in V \tag{4}$$

where $E_i^-=\{e_{ji}\in E:v_j\in V\}\cup\{e'_{ji}\in E':v_j\in V\}$, $E_i^+=\{e_{ij}\in E:v_j\in V\}\cup\{e'_{ij}\in E':v_j\in V\}$. SSP introduces the potential function $\pi(\cdot)$ to eliminate the impact of negative costs in solving SSP and establishes a transformation from the edge cost c_e to the equivalent cost c_e^π as

$$c^\pi_{e_{ij}}=c_{e_{ij}}-\pi(i)+\pi(j) \tag{5}$$

where $c_{e_{ij}}$ is the unit cost of edge $e_{ij}\in\bar{E}$.

In the SSP algorithm, the initial conditions are set as $x_e=0,\forall e\in\bar{E}$ and $\pi(i)=0,\forall v_i\in V$. Each node's imbalance $\kappa(i)$ is initialized to $b(i)$ for all $v_i\in V$, forming the unbalanced node sets $S=v_i:\kappa(i)>0$ and $D=v_i:\kappa(i)<0$. SSP then selects a source k from set S and a sink l from set D, identifies all shortest paths from k to other nodes $v_j\in V$, and calculates the path cost P_{kj} as $d(j)=\sum_{e\in P_{kj}}c_e$. Each node's potential function $\pi(i)$ is updated based on the value $d(i)$ as

$$\pi(i)=\pi(i)-d(i),\forall v_i\in V \tag{6}$$

Using the maximum feasible flow δ on the shortest path P_{kl}, the flow values for all edges on P_{kl} are updated by

$$x_e=\begin{cases} x_e+\delta & e\in P_{kl} \\ x_e & e\in\bar{E}\backslash P_{kl}\end{cases} \tag{7}$$

where $\delta=\min[\kappa(k),-\kappa(l),\min\{r_e:e\in P_{kl}\}]$. The residual capacity r_e, source set S, sink set D, and equivalent costs c_e^π are then updated accordingly. Throughout the solving process, the positive value of δ ensures that SSP terminates after a finite number of steps, achieving a balanced state for all nodes. At the end, SSP yields the optimal solution to the minimum-cost flow problem. The full SSP process is outlined in Algorithm 1.

Algorithm 1 Successive Shortest Path Algorithm

Input: Residual graph $\bar{G}(V,\bar{E})$, vector $\mathbf{b}$
Output: Optimal solution $\mathbf{x}^*$

Initialization: $x_e=0,\forall e\in\bar{E}$; $\pi(i)=0,\kappa(i)=b(i),\forall v_i\in V$.
$S=\{v_i:\kappa(i)>0\}$, $D=\{v_i:\kappa(i)<0\}$
while $S\neq\emptyset$ **do**
  Select nodes $k\in S$ and $l\in D$
  Calculate $d(j)=\min_{P_{kj}}\left\{\sum_{e\in P_{kj}}c_e:P_{kj}\subset\bar{E}\right\},\forall v_j\in V$
  Update $\pi(i)=\pi(i)-d(i),\forall v_i\in V$
  Compute $\delta=\min[\kappa(k),-\kappa(l),\min\{r_e:e\in P_{kl}\}]$
  Update flows: $x_e=x_e+\delta,\forall e\in P_{kl}$
  Update r_e: $r_e=\begin{cases} r_e-\delta & \forall e\in P_{kl}\\ r_e+\delta & \forall e\in P'_{kl}\end{cases}$ ▷ P'_{lk} is the reverse path of P_{kl}.
  Update $\kappa(k)=\kappa(k)-\delta$, $\kappa(l)=\kappa(l)-\delta$
  if $\kappa(k)=0$ **then**
    Remove node k from S
  end if
  if $\kappa(l)=0$ **then**
    Remove node l from D
  end if
  Update c_e^π: $c^\pi_{e_{ij}}=c_{e_{ij}}-\pi(i)+\pi(j),\forall e_{ij}\in\bar{E}$
end while
return $\mathbf{x}^*=\{x_e-x_{e'}\}_{e\in E}$

3.2. Successive derivative shortest path

The SSP algorithm is unsuitable for solving CCNFP because the flow allocation in each iteration disrupts the order of edge costs, leading to inconsistencies in the shortest paths before and after allocation. Recognizing these limitations of SSP for CCNFP, this subsection describes the SDSP algorithm, which approximates the solution to CCNFP by iteratively solving a series of minimum-cost flow models.

SDSP constructs an approximate minimum-cost flow model for CCNFP in each iteration, using the first-order information of the concave objective function to create a linear approximation cost. Proposition 1 provides theoretical support, showing that the optimal solution of the approximate model is consistently superior in objective value to the previous feasible solution when the first-order information is set to the gradients at the previous solution. However, the linear approximation based on the gradients leads the algorithm to converge to a suboptimal solution, with the objective value highly dependent on the initial feasible solution chosen. To address this, SDSP constructs regional first-order information by averaging derivative values of sample points within a given interval and gradually reduces the sampling interval to approach the gradient, allowing SDSP to converge to an approximate solution closer to the optimal solution of the original problem.

Proposition 1. *Given a feasible solution x^0 of the original problem (2), the approximate cost is generated by the gradients of the original objective function at x^0. The gradients are denoted as $\nabla C(x^0)=(\dots,c'_e(x_e^0),\dots)$. The objective of the approximate minimum-cost flow model is $\tilde{C}(x)=\sum_{e\in E}c'_e(x_e^0)\cdot x_e$. Then, the optimal solution of the approximate model x^1 satisfies that*

$$C(x^1)\leqslant C(x^0)$$

In other words, solution x^1 improves upon x^0 in objective value when $x^1\neq x^0$.

Proof. Since the feasible domains of CCNFP and the minimum-cost flow problem are the same, x^1 is the feasible solution of CCNFP. By the first-order condition of the concave function, the original objective $C(x)$ satisfies that

$$C(y) \leqslant C(x) + \nabla C(x)^T (y - x)$$

Thus, the difference between objectives at solutions x^1 and x^0 satisfies

$$\begin{aligned} C(x^1) - C(x^0) \leqslant & \nabla C(x^0)^T (x^1 - x^0) \\ = & \tilde{C}(x^1) - \tilde{C}(x^0) \\ \leqslant & 0 \end{aligned}$$

Equality holds if and only if x^0 is the optimal solution of the approximate model, that is, $x^1 = x^0$. □

In the graph $G(V, E)$, let $[\alpha_e, \beta_e]$ represent the sampling interval for edge e, and let n_s denote the number of sampling points. The following formula yields the average derivative value w_e under equidistant sampling:

$$w_e = \frac{1}{n_s} \sum_{i=1}^{n_s} c_e'(\alpha_e + (i-1) \cdot \Delta_e), \text{where } \Delta_e = \frac{\beta_e - \alpha_e}{n_s - 1} \tag{8}$$

Considering w_e as the unit cost on edge e, the constructed approximation model can be solved using SSP. In each iteration of SDSP, the algorithm solves the approximate model to obtain a feasible solution $x_e^{(k)}$ and then updates the lower and upper bounds of the sampling intervals for the next iteration. The updates for the interval's bounds are as follows:

$$\begin{aligned} \alpha_e^{(k+1)} &= \max\left\{ x_e^{(k)} - \frac{1}{2} \cdot a \cdot \left(\beta_e^{(k)} - \alpha_e^{(k)}\right), 0 \right\}, \\ \beta_e^{(k+1)} &= \min\left\{ x_e^{(k)} + \frac{1}{2} \cdot a \cdot \left(\beta_e^{(k)} - \alpha_e^{(k)}\right), u_e \right\}, \end{aligned} \tag{9}$$

where $a \in (0, 1)$ is the interval reduction coefficient, controlling the extent of reduction in each iteration. The algorithm iteratively performs this process until the optimal solution of the approximate model remains unchanged from the previous iteration. The details of SDSP are presented in Algorithm 2.

Algorithm 2 Successive Derivative Shortest Path

Input: Graph $G(V, E)$; Objective and derivative functions $c_e(x_e), c_e'(x_e)$; Number of sampling points n_s
Output: Flows $\{x_e\}_{e \in E}$

```
Initialize {w_e^(1)}_{e∈E} and let [α_e^(1), β_e^(1)] = [0, u_e]
for k = 1 to M do
    x_e^(k) = SSP(G, {w_e^(k)})
    if |c_e(x_e^(k)) − c_e(x_e^(k−1))| < ε, ∀e ∈ E then
        break
    end if
    α_e^(k+1) = max{ x_e^(k) − a · (β_e^(k) − α_e^(k)), 0 }
    β_e^(k+1) = min{ x_e^(k) + a · (β_e^(k) − α_e^(k)), u_e }
    Δ_e^(k+1) = (β_e^(k+1) − α_e^(k+1)) / (n_s − 1)
    w_e^(k+1) = (1/n_s) Σ_{i=1}^{n_s} c_e'(α_e^(k+1) + (i − 1) · Δ_e^(k+1))
end for
return {x_e^*}_{e∈E}
```

In SDSP, the initial values of regional first-order information w_e and the parameters n_s, a cannot be directly determined through theoretical analysis, nor can they be automatically optimized by the algorithm. To address this, we design a series of numerical experiments to evaluate the impact of various initializations and parameter settings on the algorithm's performance. These experiments serve as a reference for selecting initial values for regional first-order information and parameter settings, facilitating the practical application of SDSP.

3.3. Time and space complexity analysis

This section analyzes the time and space complexity of the SSP and SDSP. The analysis of the SSP algorithm is based on the conclusions in the book *Network Flows*. The analysis of the SDSP is derived and analyzed based on the proposed algorithm in this paper.

3.3.1. Time complexity analysis

In the SSP, each iteration solves a shortest path problem with nonnegative weights and strictly decreases the unbalanced state of some node. Consequently, if n is the number of nodes in Graph, m is the number of edges in Graph, and U is an upper bound on the largest supply of any node, the SSP terminates in at most nU iterations. Let $S(n, m, C)$ denote the time taken to solve a shortest path problem with nonnegative weights, the time complexity of the SSP is $O(nU\,S(n, m, nC))$. In this paper, we apply Dijkstra's algorithm to solve the shortest path problem, using a special data structure, the *heap*, to accelerate the Dijkstra's algorithm. Lemma 1 provides the time complexity of Dijkstra's algorithm with binary heap implementation.

Lemma 1. *A binary heap data structure requires $O(\log n)$ time to perform insert, decrease-key, and delete-min, and it requires $O(1)$ time for the other heap operations. Consequently, the binary heap version of Dijkstra's algorithm runs in $O(m \log n)$ time.*

Proof. The details of this lemma are provided in Section 4.7 of the book *Network Flows*. □

Lemma 1 states that $S(n, m, C) = O(m \log n)$, therefore establishing the time complexity of the SSP algorithm as presented in Theorem 1.

Theorem 1. *When the time complexity taken to solve a shortest path problem with nonnegative weights is $O(m \log n)$, the time complexity of the SSP algorithm is $O(nmU \log n)$.*

In the SDSP algorithm, Proposition 1 proves that the algorithm's mechanism ensures the descent of the objective function, but the theoretical property of finite-step convergence remains unclear. To address this, a maximum iteration count M is introduced to guarantee the termination of the SDSP algorithm. Building on the time complexity of the SSP algorithm, Corollary 1 provides the time complexity of the SDSP algorithm.

Corollary 1. *Let M be the maximum iteration count of the SDSP algorithm, the time complexity is $O(MnmU \log n)$.*

Although the finite-step convergence of the interval reduction mechanism in the SDSP algorithm lacks theoretical proof, numerical experiments show that the algorithm effectively achieves finite-step convergence in practice, requiring only a few iterations.

3.3.2. Space complexity analysis

In the analysis of space complexity, it is necessary to consider the storage of the graph structure as well as the storage of variables involved in the solution process of the related algorithms. To determine the space complexity of the SDSP algorithm, we first calculate the space complexities of the Dijkstra and SSP algorithms as a foundation.

In Dijkstra's algorithm, the space complexity includes graph storage, priority queue, distance array, visited array, and auxiliary structures. The SSP algorithm computes the shortest path problem by invoking Dijkstra's algorithm. Although the worst-case iteration count for the SSP algorithm is $V \times U$, since the space occupied by the variables in Dijkstra's algorithm can be released after each invocation. Therefore, the SSP algorithm does not increase the space complexity order. Similarly, when the SDSP algorithm iterates over the SSP algorithm, it stores only the numerical results from the previous iteration. As a result, the overall space complexity remains unchanged in terms of order, with a slight increase in the constant factor coefficient. Table 1 provides the detailed space complexity calculations for the three algorithms.

Table 1
Space complexity of variables in different algorithms.

Variable	Dijkstra	SSP	SDSP
Graph storage	$O(V+E)$	$O(V+E)$	$O(V+E)$
Priority queue (Binary heap)	$O(V)$		
Distance & Visited array	$O(V)$		
Residual graph		$O(V+E)$	
S, D, π, d, κ		$O(V)$	
Flows x		$O(E)$	$O(E)$
$\alpha, \beta, \Delta, \omega$			$O(E)$
Auxiliary structures	$O(1)$	$O(1)$	$O(1)$
Total	$O(V+E)$	$O(V+E)$	$O(V+E)$

4. Numerical experiments

This section demonstrates the effectiveness and advantages of SDSP in solving CCNFP through a series of numerical experiments. Section 4.1 introduces the problem instances and their construction methods. Section 4.2 presents experiments that identify the most effective initialization method for regional first-order information and optimal parameter settings. Section 4.3 examines the impact of the sequential process and interval reduction mechanisms in SDSP. Finally, Section 4.4 compares the performance of SDSP with that of nonlinear programming algorithms and a sequential linear approximation algorithm, highlighting its superiority.

The algorithms used in the numerical experiments are compiled and executed in a C++17 environment, except for the Sequential Least Squares Programming, which is implemented in Fortran and invoked through a Python interface. All program codes presented in this paper are executed on the same desktop computer, equipped with an Intel Core i7-11700 2.50 GHz processor and 32 GB of RAM.

4.1. Instruction of instances

Since the related work does not provide available instances, we construct the graph structure of CCNFP based on the instance scales outlined in the literature. Additionally, we apply various categories of concave functions as the objective of CCNFP.

The instance scale typically refers to the number of nodes and edges in the graph structure. In this paper, we construct seven groups of instances with different scales: (10, 40), (20, 100), (40, 300), (60, 400), (100, 1000), (150, 2500), and (250, 7500).[2] We randomly generate a specified number of nodes within a two-dimensional rectangular region using a uniform distribution. The connections between these nodes are determined by comparing their Euclidean distance to a threshold. A flexible threshold controls the number of connected node pairs, ensuring that the number of edges matches the specified scale for the graph. Next, we randomly select 10%, 20%, and 40% of the nodes to serve as the source and sink nodes in each instance, respectively. The node supply for each source follows the uniform distributions $\mathrm{U}(3, 30)$, while the node supply for each sink follows $\mathrm{U}(-30, -3)$. Additionally, we constructed instances with various sampling distributions to assess their impacts the algorithm's performance. Adjustments are applied to ensure the total supply equals zero. Excluding source and sink nodes, all other nodes are designated as intermediate nodes with a supply of zero. We assume that the cost function of each edge is a concave function passing the point $(0, 0)$. These cost functions are divided into three categories: Logarithmic, Power, and Sigmoid, and are defined as follows:

$$C_{Log}(x) = \log_\theta(x+1), \quad C_{Pow}(x) = (x+1)^{\frac{1}{\theta}} - 1, \quad C_{Sig}(x) = \frac{1}{1+\theta^{-x}} - \frac{1}{2} \tag{10}$$

[2] The first number represents the number of nodes, while the second represents the number of edges.

Table 2
Information about instances.

Instance characteristics	Parameter values
Node number	10, 20, 40, 60, 100, 150, 250
Edge number	40, 100, 300, 400, 1000, 2500, 7500
Source node radio	10%, 20%, 40%
Sink node radio	10%, 20%, 40%
Samp. Dist. for Source	$\mathrm{U}(3, 30), \mathrm{U}(1, 5)$ to $\mathrm{U}(26, 30)$
Samp. Dist. for Sink	$\mathrm{U}(-30, -3), \mathrm{U}(-30, -26)$ to $\mathrm{U}(-5, -1)$
Objective function	$C_{Log}(x), C_{Pow}(x), C_{Sig}(x), C_{Mix}(x)$

Table 3
The formula for initializing w_e at specific points.

Formula	$x_e = 0$	$x_e = u_e/2$	$x_e = u_e$
Derivative value	$c'_e(0)$	$c'_e(u_e/2)$	$c'_e(u_e)$
Interpolation slope	–	$\frac{2}{u_e} c_e(u_e/2)$	$\frac{1}{u_e} c_e(u_e)$

where $\theta \sim \mathrm{U}(2, 12)$. Additionally, we define a mixed-cost function that combines the three concave functions. By partitioning the edge set E into three disjoint subsets E_1, E_2, E_3, we define the mixed objective function as follows:

$$C_{Mix}(x) = \begin{cases} \log_\theta(x+1) & \text{for } e \in E_1 \\ (x+1)^{\frac{1}{\theta}} - 1 & \text{for } e \in E_2 \\ \frac{1}{1+\theta^{-x}} - \frac{1}{2} & \text{for } e \in E_3 \end{cases} \tag{11}$$

The all information for generating instances are listed in Table 2, where Samp. Dist. stands for the sampling distribution.

4.2. Initialization and parameter selection

In this subsection, we provide guidance on initialization methods, the number of sampling points, and the interval reduction coefficients based on numerical experiments.

The initialization of regional first-order information significantly affects the solution. Numerical experiments help identify the most suitable initialization method by comparing the algorithm's performance across different strategies. These strategies can be categorized into two groups. One group computes the regional first-order information using the update formula for $w_e^{(k)}$ over the initial sampling interval $[0, u_e]$, denoted as $\bar{c}'_e$. The other group uses the derivative values or linear interpolation slopes at specific points as the regional first-order information. In the experiments, we calculate the derivative value and the linear interpolation slope at the points $x_e = 0, x_e = u_e/2$, and $x_e = u_e$, as summarized in Table 3.

Table 4 presents the objective values obtained by SDSP under six different initialization methods for the constructed instances. In the experiment, 30 sampling points are used, and the interval reduction coefficient is set to 0.5. The *Min* column displays the minimum objective value for each instance. The relative error between the objective values for different initializations and the minimal value is then calculated for direct comparison. Two indicators are used to compare the results for different methods: ARE and MRE. ARE stands for Average Relative Error, which serves as an overall metric for evaluating algorithm performance. MRE stands to Maximum Relative Error, which is used to assess the algorithm's performance in the worst-case scenario. Both ARE and MRE are provided in the table. The results indicate that SDSP performs best with the following initialization methods: $w_e = \frac{2}{u_e} c_e(\frac{u_e}{2})$, $\frac{c_e(u_e)}{u_e}$, and $\bar{c}'_e$. For these three initializations, the ARE remains consistently below 2.5%. In contrast, the ARE for the other methods exceeds 10%. However, the difference between these three initialization methods is insufficient to determine a clear winner. Next, we combine the results from Table 4 with experiments on the number of samples and the interval reduction coefficient to identify the optimal initialization method, sampling number, and coefficient.

Table 4
The objective values of SDSP under six types of initialization.

Objective	Scale	Objective values ($n_s = 30, a = 0.5$)						Min
		$c'_e(0)$	$c'_e(\frac{u_e}{2})$	$\frac{2}{u_e}c_e(\frac{u_e}{2})$	$c'_e(u_e)$	$\frac{c_e(u_e)}{u_e}$	$\bar{c}'_e(x_e)$	
$C_{Log}(x)$	(20, 100)	16.110	15.966	**15.928**	15.966	15.966	**15.928**	15.928
	(60, 400)	14.535	**14.316**	14.339	14.764	14.339	14.339	14.316
	(150, 2500)	**48.754**	49.478	51.536	49.497	49.617	49.617	48.754
	(250, 7500)	119.695	107.613	108.444	107.960	107.814	**106.965**	106.965
$C_{Pow}(x)$	(20, 100)	**4.582**	4.617	4.617	4.617	4.617	4.617	4.582
	(60, 400)	1.040	1.083	1.050	1.110	1.083	**1.029**	1.029
	(150, 2500)	**13.362**	14.107	13.791	13.620	13.838	13.529	13.362
	(250, 7500)	33.160	32.196	31.351	31.898	**30.995**	31.351	30.995
$C_{Sig}(x)$	(20, 100)	6.495	6.753	**5.749**	6.753	**5.749**	6.249	5.749
	(60, 400)	14.990	18.499	**14.467**	17.980	**14.467**	**14.467**	14.467
	(150, 2500)	28.140	32.946	**21.737**	32.187	**21.737**	23.618	21.737
	(250, 7500)	50.981	80.461	36.201	80.050	36.201	**35.444**	35.444
$C_{Mix}(x)$	(20, 100)	7.694	7.584	**6.793**	7.133	7.021	**6.793**	6.793
	(60, 400)	7.946	12.666	8.037	12.666	**7.917**	8.725	7.917
	(150, 2500)	28.148	37.207	**18.596**	38.605	18.799	19.149	18.596
	(250, 7500)	55.973	96.011	39.558	96.758	39.787	**38.972**	38.972
ARE (%)		13.82	34.99	1.22	34.74	1.33	2.23	
MRE (%)		51.37	146.36	5.71	148.28	5.25	10.21	

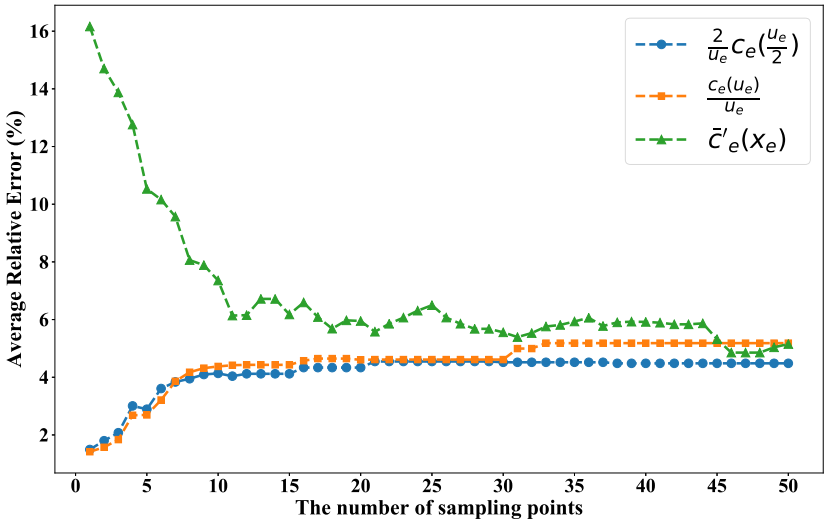


Fig. 1. The trend of average relative error corresponds to the number of sampling points. The interval reduction coefficient is set to 0.5.

Two experiments are designed to compare the performance of SDSP with varying sampling numbers and interval reduction coefficients under three selected initialization methods. Fig. 1 illustrates the trend of the relative error in objective values as a function of the sampling number, while Fig. 2 shows the trend of relative error with respect to the interval reduction coefficient. The results demonstrate that the linear interpolation slope initialization methods outperform the derivative mean method. However, no significant difference is observed between the two linear interpolation slope methods. Based on these findings, we adopt $\frac{c_e(u_e)}{u_e}$ as the initial regional first-order information, set the number of sampling points to one, and choose an interval reduction coefficient of 0.65.

These two figures show more information about the effect of the regional first-order information and the interval reduction mechanism. In Fig. 1, fewer sampling points implies that the averaged derivative computed within the interval effectively approximates the gradient at a point closer to the upper part of the region. As the number of sampling points increases, the averaged derivative shifts to reflect the gradient of a point closer to the interval's midpoint. When the initialization method uses the linear interpolation slope, fewer sampling points demonstrate a distinct advantage. From an optimization perspective, this phenomenon arises because the gradient of a concave function in the upper region of the interval is inherently steeper than at the midpoint. This steepness results in a lower effective cost for the reverse edge in the residual graph during the algorithm's iterations, increasing the likelihood of its selection in subsequent iterations. This mechanism introduces natural reversibility to the flow assignments, allowing the algorithm to explore the solution space more effectively and avoid premature convergence. As a result, the algorithm can overcome the limitations of the successive shortest path algorithm, which struggles with concave cost structures, ultimately achieving higher-quality solutions and better objective values. This behavior highlights the practical utility of leveraging regional first-order information in the SDSP framework.

In Fig. 2, the average relative error exhibits a distinct "U-shaped" relationship with the interval reduction coefficient. When the coefficient approaches zero, the algorithm relies only on the gradient value at the solution from the previous iteration to approximate the concave cost function. Conversely, when the coefficient approaches one, the algorithm uses a fixed-length interval for sampling to construct the linear objective. The experimental results show that both extreme approaches are suboptimal for solving the problem. An iterative framework based solely on single-point gradient values often converges to suboptimal solutions due to the challenge of revisiting previously assigned flow values. Conversely, using fixed-length interval sampling throughout the

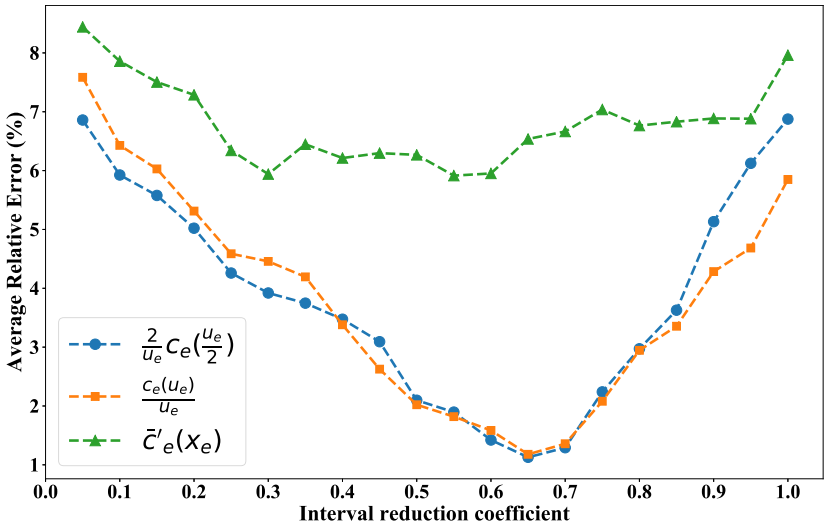


Fig. 2. The trend of average relative error corresponds to the interval reduction coefficient. The number of sampling points is set to 30.

Table 5
Validation on sequential process and regional first-order information.

Objective	Scale	Objective values				
		LPo	LPu	SGSP	SDSP($a = 1$)	SDSP
$C_{Log}(x)$	(20, 100)	16.248	15.966	15.966	15.844	**15.760**
	(60, 400)	14.689	15.233	15.233	14.556	**14.222**
	(150, 2500)	53.033	55.037	55.037	45.689	**44.772**
	(250, 7500)	89.473	80.368	78.816	75.795	**71.763**
$C_{Pow}(x)$	(20, 100)	4.820	**4.617**	**4.617**	**4.617**	**4.617**
	(60, 400)	1.069	1.116	1.116	1.015	**1.006**
	(150, 2500)	13.734	14.053	13.891	13.198	**12.830**
	(250, 7500)	23.483	21.327	20.553	20.444	**19.793**
$C_{Sig}(x)$	(20, 100)	6.495	**5.749**	**5.749**	**5.749**	**5.749**
	(60, 400)	15.999	14.500	14.000	14.500	**12.956**
	(150, 2500)	34.614	22.834	22.834	22.834	**19.244**
	(250, 7500)	57.630	38.026	38.206	38.026	**37.525**
$C_{Mix}(x)$	(20, 100)	8.914	7.442	7.442	7.076	**7.021**
	(60, 400)	9.033	8.578	7.912	7.728	**7.604**
	(150, 2500)	29.050	22.317	20.887	19.900	**17.788**
	(250, 7500)	33.166	30.333	26.717	27.913	**25.166**
ARE (%)		24.79	10.52	7.90	4.67	–

process risks assigning reverse flow to edges that should receive flow, hindering the algorithm's ability to converge to high-quality solutions. The interval reduction mechanism effectively navigates the trade-offs between these opposing factors, enabling the algorithm to find a Pareto-optimal balance. This mechanism is analogous to step-size adjustment in optimization algorithms: larger step sizes help escape local minima, while reductions ensure convergence to a local minimum in a more promising neighborhood.

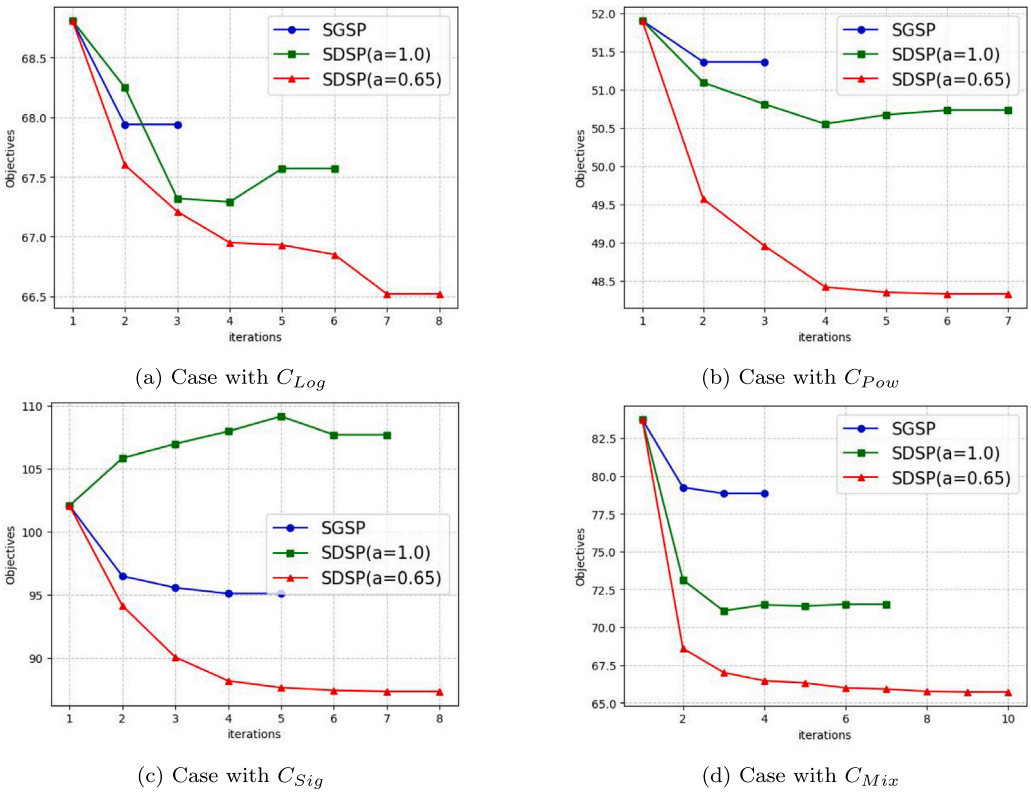
4.3. Effectiveness

This section aims to validate the effectiveness of SDSP by showcasing the improved performance achieved through both the sequential process and the incorporation of regional first-order information.

In comparison, we construct two minimum-cost flow models to approximate the CCNFP in a single step. The linear unit cost on edges is defined as $\tilde{c}_e(x) = c'_e(0)\cdot x$ or $\tilde{c}_e(x) = \frac{c_e(u_e)}{u_e}\cdot x$, with these two approximate models denoted as LPo and LPu respectively. Additionally, we simplify the SDSP by updating $w_e^{(k+1)}$ based solely on the gradients at point $x_e^{(k)}$, that is, $w_e^{(k+1)} = c'_e(x_e^{(k)})$ which is denoted as SGSP. In the experiments, we ensure that the initialization and parameter settings of SGSP are consistent with those of SDSP.

Table 5 presents the objective values of these four approaches across 16 instances. The objective values obtained by SDSP are consistently minimal compared to the others for all instances. We calculate the average relative error between SDSP's objective values and those of the other methods. The results show that the objective values achieved by SGSP are always less than or equal to those obtained by LPu, confirming the descent property outlined in Proposition 1. However, when updating $w_e^{(k+1)}$ using gradients, the sequential process only results in a modest 2.62% decrease in the objective value. Introducing regional first-order information into the sequential process leads to a substantial improvement, with the average relative error between SGSP and SDSP reaching 7.9%. The descending trend in ARE from LPu to SDSP highlights the crucial role that the sequential process and the integration of regional first-order information play in optimizing the objective values of CCNFP. Also, the results of SDSP with the parameter $a = 1$ have been listed in the table for comparison. The SDSP algorithm without interval reduction explores a wider solution space than SGSP, resulting in a generally superior solution. However, the algorithm still shows a 4.67% potential for improvement when the interval reduction mechanism is incorporated.

Fig. 3 shows the decrease in the objective function across four cases with distinct target functions, comparing the performance of three iterative strategies. Both SDSP($a = 0.65$) and SGSP show a



(a) Case with C_{Log} (b) Case with C_{Pow}

(c) Case with C_{Sig} (d) Case with C_{Mix}

Fig. 3. The decreasing process of SGSP and SDSP.

steady decrease in the objective value over iterations, but SGSP often converges prematurely to suboptimal local minima. Conversely, SDSP without interval reduction (when $a = 1$) achieves better results than SGSP in several cases, but it lacks guaranteed monotonic decrease and quickly stagnates after a few iterations. The proposed interval reduction mechanism in SDSP effectively integrates the advantages of the other two strategies, enabling it to identify higher-quality solutions. Furthermore, even without a predefined maximum iteration limit, the interval reduction mechanism exhibits finite-step convergence in numerical experiments, reliably meeting the stopping criteria within a limited number of iterations. These findings highlight how SDSP balances exploration and exploitation, leading to robust and efficient optimization performance.

4.4. Comparison

To evaluate the performance of SDSP, we conducted comparison experiments with existing algorithms for solving the concave cost network flow problem on different graph structure instances.

We compare SDSP with two categories of algorithms. The first category includes three nonlinear programming algorithms:

- Sequential Least Square Programming (SLSQP) [28]
- Augmented Lagrange Function algorithm (AugLag) [29–33]
- Penalty Function algorithm (Penalty) [29,30,34,35]

SLSQP is a gradient-based optimization method that solves constrained nonlinear optimization problems by iteratively approximating the objective function and constraints with quadratic models, efficiently handling both equality and inequality constraints. The AugLag algorithm combines Lagrangian relaxation of constraints with an augmented penalty term, improving convergence and stability in solving constrained optimization problems. The Penalty method incorporates a penalty term into the objective function to penalize constraint violations, solving the problem as an unconstrained optimization while progressively tightening the penalty. The second category includes the sequential linear approximate algorithm, Dynamic Slope Scaling Procedure (DSSP) [36], which is also applicable to CCNFP. All baseline algorithm parameter settings are available in the shared GitHub repository, ensuring consistency by using the same set of parameters for all experimental cases. First, we conduct experiments on instances with graph structures that include seven different scales and three source node ratios, resulting in a total of 84 instances. The source node supply follows a sampling distribution of U(3,30). Next, we explore the performance of five algorithms on instances with different supply sampling distributions.

Tables 6 and 7 present the average and maximum relative errors between the objective values obtained by the five algorithms and the minimum across 84 instances. The minimum for each instance is the smallest objective value among the five algorithms. The details of the results for all instances are listed in Appendix. Table 6 shows the performance differences of the algorithms across instances with varying scales. Table 7 shows the performance of the algorithms under different source node ratios. In Table 6, we observe that the average relative error of the proposed algorithm decreases as the instance scale increases: the larger the scale, the smaller the relative error. This suggests that as the problem scale increases, the SDSP algorithm can find the optimal solution in most cases. In contrast, both the SLSQP and DSSP algorithms show an increase in average relative error as the problem scale grows, while the average relative error of the AugLag algorithm remains around 10%. This further highlights the significant advantage of the SDSP algorithm in solving large-scale concave cost network flow problems. With an average relative error of 0.71% and a maximum relative error of 8.69%, the SDSP algorithm demonstrates both effectiveness and stability in solving this problem. Even in the worst-case scenario, the algorithm's relative error remains within 10%, a level unattainable by the other compared algorithms.

In Table 7, the performance of all five algorithms generally shows that the average relative error decreases as the ratio of source nodes

Table 6
The performance of SDSP and comparison algorithms.

Scale	SLSQP		AugLag		Penalty		DSSP		SDSP	
	ARE	MRE	ARE	MRE	ARE	MRE	ARE	MRE	ARE	MRE
(10, 40)	170.57	417.58	14.58	66.26	51.37	169.61	16.86	141.8	2.37	8.69
(20, 100)	152.07	276.91	5.49	15.98	36.18	143.30	3.92	15.68	0.99	4.50
(40, 300)	252.92	421.02	11.02	48.77	65.14	192.75	6.98	16.14	0.64	4.05
(60, 400)	267.55	533.55	17.81	72.42	142.67	485.69	7.46	27.29	0.48	2.60
(100, 1000)	274.19	395.98	7.76	27.86	72.75	184.67	8.12	17.74	0.49	5.85
(150, 2500)	489.89	1160.9	16.65	45.24	119.83	223.76	22.24	79.88	0.00	0.05
(250, 7500)	952.04	2105.8	14.19	39.94	171.06	375.67	45.80	96.54	0.00	0.00
Mean (%)	365.60		12.50		94.14		15.91		0.71	
Std.	280.968		4.589		50.692		14.662		0.812	
Max. (%)		2105.8		72.42		485.69		141.8		8.69

Table 7
The performance of SDSP and comparison algorithms.

Source radio	SLSQP		AugLag		Penalty		DSSP		SDSP	
	ARE	MRE	ARE	MRE	ARE	MRE	ARE	MRE	ARE	MRE
10%	497.63	2105.79	15.61	72.42	120.34	485.69	24.65	141.75	0.72	5.85
20%	312.46	1471.19	12.64	65.92	81.07	213.28	13.78	66.85	0.93	8.67
40%	286.72	1115.52	9.00	45.24	81.20	223.76	9.30	44.49	0.48	7.26

Table 8
The memory usage of SDSP and comparison algorithms.

Scale	I/O	Average memory usage (KB)				
		SLSQP	AugLag	Penalty	DSSP	SDSP
(10, 40)	3477	451	104	473	100	10
(20, 100)	3489	1 140	260	1 021	146	14
(40, 300)	3518	1 450	615	3 015	294	43
(60, 400)	3549	2 432	682	4 417	351	60
(100, 1000)	3640	5 916	13 032	10 672	1129	58
(150, 2500)	3877	18 873	14 757	9 770	1143	335
(250, 7500)	4645	75 373	680 440	42 575	2001	995

increases. This is because, as the ratio of source and sink nodes increases, the proportion of intermediate nodes decreases, causing the graph structure to gradually approach a bipartite graph. Simple graph structures reduce the performance gap between the algorithms, resulting in a decrease in the average relative error for all five algorithms. A notable difference from the baseline is that the SDSP algorithm shows improvement even when only 10% of the nodes are sources. This is because the problem tends to resemble a single-source network flow problem, where the SDSP algorithm, which is based on minimum-cost flow optimization, has advantages over other nonlinear programming algorithms when handling such problems.

Table 8 presents the memory consumption of the five algorithms for solving various instances, as well as the memory used for I/O operations by Visual Studio. Since the space complexity of most algorithms is related to the problem scale, Table 8 reports the average memory usage for each algorithm across different problem scales. The memory consumption of Visual Studio for I/O operations is relatively constant, ranging from 3000 KB to 5000 KB, with a slow increase as the data size grows. Comparing the memory usage of the five algorithms, it is evident that the SDSP algorithm has a significant advantage in terms of memory consumption compared to the baselines. Even for the largest case, its average memory usage remains below 1 MB. Clearly, in terms of memory usage, the SDSP algorithm has an advantage that is difficult for other algorithms to match, making it highly feasible for large-scale problem-solving or scenarios involving integrated calls.

Efficiency is a key criterion for evaluating the practicality of an algorithm. Table 9 presents the solving time of each algorithm across the instances. SDSP demonstrates significantly superior solving efficiency compared to nonlinear programming algorithms. This advantage enables SDSP to tackle large-scale problems that nonlinear programming algorithms struggle with, making it more suitable for practical applications. In comparison to DSSP, SDSP also shows a slight efficiency advantage. This discrepancy can be attributed to the lack of a monotonic descent guarantee in the iterative framework of DSSP, and it is expected to become more pronounced as problem scales increase.

Table 9
The solving time of SDSP and comparison algorithms.

Scale	Average solving time (s)					Iter.[a]
	SLSQP	AugLag	Penalty	DSSP	SDSP	
(10, 40)	0.013	0.129	0.163	0.001	<0.001	2.5
(20, 100)	0.054	1.746	0.634	0.006	0.002	3.2
(40, 300)	0.675	8.164	3.051	0.074	0.012	4.1
(60, 400)	1.937	21.946	6.336	0.188	0.051	4.6
(100, 1000)	27.543	510.061	56.963	1.343	0.233	5.7
(150, 2500)	748.825	4362.6	454.676	4.243	0.681	5.3
(250, 7500)	9.37 h	37.45 h	3.65 h	20.283	4.269	6.9

[a] Iter. stands for the average iteration count of SDSP.

To explore whether the sampling distribution of supply values at source nodes impacts the algorithm's performance, we divided the interval $[1, 30]$ into six subintervals and randomly sampled supply values from a uniform distribution within each subinterval to construct the instances. Table A.13 presents the results of the five algorithms on instances with five different graph structures, under these six sampling distributions. From the data, it is evident that the different sampling distributions do not significantly impact on the performance and efficiency of the SDSP algorithm.

In Table A.13, we can observe that across all the distributions (from $U(1, 5)$ to $U(26, 30)$), the SDSP algorithm consistently performs well, with minimal fluctuations in the objective values and solution times. For example, in most cases, the objective values for SDSP are either equal to or very close to the best possible values, demonstrating stability across all supply distributions. Additionally, the algorithm's

Table A.10
The performance of SDSP and comparison algorithms (source radio = 10%).

Objective	Scale	S.N.[a]	Objective value					Min
			SLSQP	AugLag	Penalty	DSSP	SDSP	
$C_{Log}(x)$	(10, 40)	1	9.424	**4.953**	5.940	**4.953**	**4.953**	4.953
	(20, 100)	2	33.386	**15.319**	17.514	16.248	15.760	15.319
	(40, 300)	4	53.306	**20.549**	23.558	23.718	21.098	20.549
	(60, 400)	7	40.743	**13.968**	16.608	15.440	14.222	13.968
	(100, 1000)	10	156.394	52.453	52.414	52.837	**46.970**	46.970
	(150, 2500)	11	272.358	45.824	53.787	52.592	**44.772**	44.772
	(250, 7500)	25	665.361	78.236	102.791	108.983	**71.763**	71.763
$C_{Pow}(x)$	(10, 40)	1	3.536	1.673	1.698	**1.659**	1.728	1.659
	(20, 100)	2	12.342	4.582	5.222	**4.527**	4.617	4.527
	(40, 300)	4	24.454	7.064	9.192	7.073	**6.832**	6.832
	(60, 400)	7	4.057	1.081	5.892	1.051	**1.006**	1.006
	(100, 1000)	10	52.419	14.629	21.567	14.532	**13.440**	13.440
	(150, 2500)	11	60.879	13.410	26.640	14.399	**12.830**	12.830
	(250, 7500)	25	242.382	21.624	78.630	26.159	**19.792**	19.792
$C_{Sig}(x)$	(10, 40)	1	8.364	1.985	3.476	1.985	**1.968**	1.968
	(20, 100)	2	21.665	6.500	13.985	6.649	**5.749**	5.749
	(40, 300)	4	39.269	8.977	14.966	8.448	**7.537**	7.537
	(60, 400)	7	82.076	22.001	75.424	16.490	**12.956**	12.956
	(100, 1000)	10	99.034	**22.635**	51.045	26.650	23.960	22.635
	(150, 2500)	11	242.649	25.311	59.777	34.616	**19.244**	19.244
	(250, 7500)	25	625.530	39.964	178.491	73.751	**37.524**	37.524
$C_{Mix}(x)$	(10, 40)	1	7.686	2.469	2.485	3.590	**1.485**	1.485
	(20, 100)	2	23.400	7.297	8.690	**7.012**	7.021	7.012
	(40, 300)	4	40.870	9.920	10.856	**8.939**	8.991	8.939
	(60, 400)	7	43.242	13.111	26.926	8.011	**7.604**	7.064
	(100, 1000)	10	109.278	22.823	30.708	26.256	**22.655**	22.655
	(150, 2500)	11	121.382	25.475	37.259	25.434	**17.788**	17.788
	(250, 7500)	25	555.087	35.217	75.282	40.215	**25.165**	25.165
ARE (%)			497.63	15.61	120.34	24.65	0.72	–
MRE (%)			2105.79	72.42	485.69	141.75	5.85	–

[a] S.N. stands for the number of sources.

computational time shows no significant variation as the supply distribution changes, further supporting the claim that the sampling distribution of supply values does not affect the algorithm's efficiency or effectiveness. This suggests that the SDSP algorithm is robust to variations in the supply distribution, which is a desirable property when solving real-world problems where supply distributions may not be known in advance.

The superior objective values, less memory usage, and extremely short solving times demonstrate that SDSP can effectively and rapidly solve the CCNFP. By integrating the SSP with regional first-order information, SDSP emerges as a versatile and efficient algorithm for solving the CCNFP.

5. Conclusion

This paper presents a universal algorithm for the concave cost network flow problem (CCNFP), combining the successive shortest path method with regional first-order information to enhance efficiency and solution quality. Experiments on initialization and parameter settings provide practical guidance for real-world applications. While the successive derivative shortest path (SDSP) algorithm contributes significantly to CCNFP research and industrial applications, future research could focus on improving scalability for large-scale networks, integrating machine learning for dynamic parameter tuning, extending the algorithm to stochastic or dynamic network problems, and conducting practical case studies to validate its utility. Additionally, future research could explore adaptive interval reduction strategies, alternative mechanisms for leveraging first-order information, and exploring more rigorous approximation bounds to further strengthen the algorithm's robustness and theoretical foundation.

CRediT authorship contribution statement

Lu Yang: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Conceptualization. **Zhouwang Yang:** Writing – review & editing, Supervision.

Ethical approval

This study does not involve human or animal subjects and, therefore, does not require ethical approval.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work is supported by the NSF of China (Nos. 92270205, 12301659, 12171453), the National Key R&D Program of China (Nos. 2022YFA1005201, 2022YFA1005202, 2022YFA1005203), and the Major Project of Science and Technology Innovation Tackling Plan of Anhui Province (No. 202423e09050003).

Appendix. The detail of results in comparison

Tables A.10, A.11, and A.12 show the objective values of SDSP and baselines on all instances with sampling distribution $U(3, 30)$. Table A.13 presents the results of five algorithms on instances using six different supply sampling distributions.

Table A.11
The performance of SDSP and comparison algorithms (source radio = 20%).

Objective	Scale	S.N.	Objective value					Min
			SLSQP	AugLag	Penalty	DSSP	SDSP	
$C_{Log}(x)$	(10, 40)	2	12.233	6.852	7.137	**6.359**	6.817	6.359
	(20, 100)	4	31.043	**14.429**	15.213	14.648	14.641	14.429
	(40, 300)	8	79.667	37.438	40.974	37.364	**33.862**	33.862
	(60, 400)	12	125.229	**55.867**	60.465	56.722	57.318	55.867
	(100, 1000)	20	180.192	66.711	81.398	68.125	**66.520**	66.520
	(150, 2500)	30	246.597	71.621	81.842	78.679	**70.261**	70.261
	(250, 7500)	50	628.204	126.212	156.388	156.816	**120.735**	120.735
$C_{Pow}(x)$	(10, 40)	2	4.067	**1.927**	2.218	2.187	2.094	1.927
	(20, 100)	4	10.695	**4.625**	5.462	4.809	4.833	4.625
	(40, 300)	8	31.517	**10.948**	17.320	11.828	10.991	10.948
	(60, 400)	12	52.718	17.899	24.176	17.549	**17.475**	17.475
	(100, 1000)	20	71.233	19.472	32.521	19.819	**19.298**	19.298
	(150, 2500)	30	89.989	21.037	49.901	22.031	**19.375**	19.375
	(250, 7500)	50	194.859	35.513	97.613	41.199	**33.426**	33.426
$C_{Sig}(x)$	(10, 40)	2	8.892	3.997	6.495	2.881	**2.409**	2.409
	(20, 100)	4	18.572	7.930	12.467	7.376	**7.075**	7.075
	(40, 300)	8	48.366	17.687	34.805	13.808	**11.889**	11.889
	(60, 400)	12	75.675	25.767	50.759	**22.225**	22.517	22.225
	(100, 1000)	20	138.305	35.654	70.943	32.124	**27.885**	27.885
	(150, 2500)	30	160.998	35.874	103.043	39.036	**32.892**	32.892
	(250, 7500)	50	550.960	-[a]	156.605	92.784	**55.608**	55.608
$C_{Mix}(x)$	(10, 40)	2	9.434	4.364	4.477	3.987	**3.597**	3.597
	(20, 100)	4	21.409	9.340	11.229	9.239	**9.175**	9.175
	(40, 300)	8	49.441	18.894	24.626	16.831	**15.974**	15.974
	(60, 400)	12	94.826	30.305	37.676	30.801	**27.011**	27.011
	(100, 1000)	20	134.053	36.667	55.024	35.456	**32.269**	32.269
	(150, 2500)	30	167.05	34.419	62.201	33.089	**28.157**	28.157
	(250, 7500)	50	688.008	56.490	113.206	71.632	**43.789**	43.789
ARE (%)			312.46	12.64	81.07	13.78	0.93	–
MRE (%)			1471.19	65.92	213.28	66.85	8.67	–

[a] The case was excluded due to a runtime exceeding 100 h.

Table A.12
The performance of SDSP and comparison algorithms (source radio = 40%).

Objective	Scale	S.N.	Objective value					Min
			SLSQP	AugLag	Penalty	DSSP	SDSP	
$C_{Log}(x)$	(10, 40)	4	21.899	**12.098**	**12.098**	13.040	**12.098**	12.098
	(20, 100)	8	50.016	**28.848**	31.456	29.820	29.136	28.848
	(40, 300)	16	90.048	32.743	33.051	31.578	**31.536**	31.536
	(60, 400)	24	134.844	54.683	60.982	55.515	**53.662**	53.662
	(100, 1000)	40	265.986	112.562	121.834	112.51	**109.64**	109.64
	(150, 2500)	60	571.204	137.229	158.675	145.319	**136.279**	136.279
	(250, 7500)	100	667.077	191.432	239.798	225.117	**183.717**	183.717
$C_{Pow}(x)$	(10, 40)	4	9.335	3.639	5.070	**3.639**	3.682	3.639
	(20, 100)	8	23.617	11.430	13.059	11.566	**11.245**	11.245
	(40, 300)	16	33.850	8.744	13.058	8.842	**8.689**	8.689
	(60, 400)	24	51.909	17.352	24.385	18.141	**17.029**	17.029
	(100, 1000)	40	103.238	34.698	54.264	34.129	**33.847**	33.847
	(150, 2500)	60	166.401	39.115	78.062	**37.626**	37.646	37.626
	(250, 7500)	100	351.553	51.094	135.612	55.435	**48.326**	48.326
$C_{Sig}(x)$	(10, 40)	4	9.818	4.990	11.951	4.753	**4.457**	4.457
	(20, 100)	8	21.385	10.254	15.882	9.608	**8.843**	8.843
	(40, 300)	16	55.959	17.881	44.38	**15.740**	16.378	15.740
	(60, 400)	24	76.183	28.943	56.894	23.667	**22.866**	22.866
	(100, 1000)	40	182.580	51.267	127.699	48.013	**44.858**	44.858
	(150, 2500)	60	387.752	85.658	190.943	73.743	**58.977**	58.977
	(250, 7500)	100	548.775	-[a]	259.373	126.161	**87.312**	87.312
$C_{Mix}(x)$	(10, 40)	4	14.576	**6.999**	8.495	7.090	7.507	6.999
	(20, 100)	8	35.952	16.278	15.490	**14.035**	**14.035**	14.035
	(40, 300)	16	67.424	18.818	27.261	20.249	**18.255**	18.255
	(60, 400)	24	92.275	23.376	34.273	25.509	**22.690**	22.690
	(100, 1000)	40	166.795	53.625	82.058	50.107	**48.864**	48.864
	(150, 2500)	60	323.589	64.378	119.289	61.524	**50.927**	50.927
	(250, 7500)	100	798.803	83.781	175.731	94.553	**65.717**	65.717
ARE (%)			286.72	9.00	81.02	9.30	0.48	–
MRE (%)			1115.52	45.24	223.76	44.49	7.26	–

[a] The case was excluded due to a runtime exceeding 100 h.

Table A.13
The performance of algorithms on different distributions.

Dist.	Scale	SLSQP		AugLag		Penalty		DSSP		SDSP	
		Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)
U(1, 5)	(10, 40)	1.233	0.027	**0.490**	0.012	**0.490**	0.007	2.7439	0	**0.490**	0
	(20, 100)	5.782	0.064	**0.544**	0.520	**0.544**	0.842	**0.544**	0	**0.544**	0
	(40, 300)	19.600	0.454	3.037	6.393	2.875	3.835	2.912	0.022	**2.746**	0.003
	(60, 400)	9.505	2.597	4.721	38.861	**4.159**	8.993	4.637	0.028	4.581	0.004
	(100, 1000)	11.728	34.397	**6.463**	168.171	6.828	40.463	6.791	0.178	7.093	0.016
U(6, 10)	(10, 40)	5.686	0.011	**1.378**	0.048	**1.378**	0.021	**1.378**	0.001	**1.378**	0
	(20, 100)	7.295	0.065	**2.994**	0.094	**2.994**	0.065	**2.994**	0.001	**2.994**	0.001
	(40, 300)	16.900	1.531	6.289	8.870	5.864	2.732	6.929	0.026	**5.295**	0.002
	(60, 400)	30.033	2.475	**9.386**	26.222	10.97	5.505	10.273	0.053	9.741	0.015
	(100, 1000)	92.566	52.944	12.354	191.858	14.077	40.431	11.015	0.211	**10.575**	0.038
U(11, 15)	(10, 40)	7.736	0.029	**2.999**	0.376	3.089	0.212	**2.999**	0	**2.999**	0
	(20, 100)	15.710	0.044	**2.666**	0.557	5.678	0.280	3.165	0.003	**2.666**	0
	(40, 300)	30.678	0.732	5.088	13.900	7.895	3.608	4.842	0.018	**4.658**	0.003
	(60, 400)	49.007	2.491	10.822	24.504	13.184	4.167	11.040	0.030	**9.428**	0.004
	(100, 1000)	105.015	10.885	**13.696**	195.967	20.725	58.975	16.006	0.303	13.802	0.048
U(16, 20)	(10, 40)	8.919	0.015	**1.546**	0.093	2.879	0.380	2.008	0.001	**1.546**	0
	(20, 100)	16.480	0.010	2.000	0.098	2.666	0.210	**0.790**	0.001	**0.790**	0
	(40, 300)	28.831	1.600	6.588	4.903	10.729	2.095	7.166	0.025	**5.419**	0.003
	(60, 400)	58.931	1.173	15.686	62.042	17.553	12.306	12.002	0.051	**10.893**	0.011
	(100, 1000)	108.513	24.762	19.412	279.809	31.521	53.620	18.924	0.301	**14.794**	0.026
U(21, 25)	(10, 40)	11.294	0.020	**3.031**	0.232	5.517	0.551	4.297	0.001	**3.031**	0.001
	(20, 100)	17.965	0.177	7.257	1.027	7.705	0.900	7.198	0.003	**5.178**	0
	(40, 300)	49.990	0.506	8.796	7.756	23.956	5.889	10.873	0.029	**7.747**	0.004
	(60, 400)	47.212	2.968	11.723	17.649	20.677	8.747	13.822	0.085	**10.786**	0.008
	(100, 1000)	120.061	35.225	18.720	435.540	38.801	68.195	23.184	0.623	**17.877**	0.049
U(26, 30)	(10, 40)	13.085	0.008	**4.166**	0.192	4.996	0.182	4.363	0.001	**4.166**	0.001
	(20, 100)	23.087	0.119	**9.071**	0.423	12.675	1.635	9.413	0.004	9.416	0.001
	(40, 300)	66.460	0.849	16.811	10.953	27.407	4.917	19.164	0.067	**14.655**	0.013
	(60, 400)	77.165	1.411	21.391	26.999	32.316	7.841	**20.406**	0.097	20.854	0.011
	(100, 1000)	99.401	28.575	20.825	196.434	39.568	57.889	22.338	0.934	**15.598**	0.100

Data availability

The data generated and analyzed during this study are publicly available at the following link: https://github.com/lufizyang/Data-and-Code-for-CCNFP.

References

[1] Klein M. A primal method for minimal cost flows with applications to the assignment and transportation problems. Manag Sci 1967;14:205–20. http://dx.doi.org/10.1287/mnsc.14.3.205.

[2] Bassetti F, Gualandi S, Veneroni M. On the computation of kantorovich–wasserstein distances between two-dimensional histograms by uncapacitated minimum cost flows. SIAM J Optim 2020;30:2441–69. http://dx.doi.org/10.1137/19M1261195.

[3] Sarkar S, Chakrabarti A, Prasad Mukherjee D. Generation of ball possession statistics in soccer using minimum-cost flow network. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR) workshops. 2019.

[4] Jenne M. Minimum-cost flow algorithms for the wind farm cabling problem (Bachelor thesis), Karlsruhe Institute of Technology; 2020.

[5] Farsi M, Fetz A, Filippini M. Economies of scale and scope in local public transportation. J Transp Econ Policy (JTEP) 2007;41:345–61.

[6] Shen Z-JM, Coullard C, Daskin MS. A joint location inventory model. Transp Sci 2003;37:40–55. http://dx.doi.org/10.1287/trsc.37.1.40.12823.

[7] Orlin J. A faster strongly polynomial minimum cost flow algorithm. In: Proceedings of the twentieth annual ACM symposium on theory of computing. 1988, p. 377–87.

[8] Sifaleras A. Minimum cost network flows: Problems, algorithms, and software. Yugosl J Oper Res 2016;23(1).

[9] Kovács P. Minimum-cost flow algorithms: an experimental evaluation. Optim Methods Softw 2015;30(1):94–127.

[10] Ahuja RK, Magnanti TL, Orlin JB. Network flows. 1988.

[11] Wu Z, Karimi HR, Dang C. A deterministic annealing neural network algorithm for the minimum concave cost transportation problem. IEEE Trans Neural Netw Learn Syst 2019;31:4354–66. http://dx.doi.org/10.1109/TNNLS.2019.2955137.

[12] Lin Y, Schrage L. The global solver in the lindo api. Optim Methods Softw 2009;24:657–68. http://dx.doi.org/10.1080/10556780902753221.

[13] Balakrishnan A, Graves SC. A composite algorithm for a concave-cost network flow problem. Networks 1989;19:175–202. http://dx.doi.org/10.1002/net.3230190202.

[14] Konno H, Egawa T. Computational studies on large scale concave cost transportation problems. Pac J Optim 2006;2:327–40.

[15] Altiparmak F, Karaoglan I. An adaptive tabu-simulated annealing for concave cost transportation problems. J Oper Res Soc 2008;59(3):331–41.

[16] Pegon P, Piazzoli D, Santambrogio F. Full characterization of optimal transport plans for concave costs. 2013, arXiv preprint arXiv:1311.3406.

[17] Kelly DL, Khumawala BM. Capacitated warehouse location with concave costs. J Oper Res Soc 1982;33(9):817–26.

[18] Guisewite GM, Pardalos PM. Minimum concave-cost network flow problems: Applications, complexity, and algorithms. Ann Oper Res 1990;25:75–99. http://dx.doi.org/10.1007/BF02283688.

[19] Xing T, Zhou X. Finding the most reliable path with and without link travel time correlation: A lagrangian substitution based approach. Transp Res B: Methodol 2011;45:1660–79. http://dx.doi.org/10.1016/j.trb.2011.06.004.

[20] Monteiro MS, Fontes DB, Fontes FA. Concave minimum cost network flow problems solved with a colony of ants. J Heuristics 2013;19:1–33. http://dx.doi.org/10.1007/s10732-012-9214-6.

[21] Ghasemishabankareh B, Ozlen M, Li X, Deb K. A genetic algorithm with local search for solving single-source single-sink nonlinear non-convex minimum cost flow problems. Soft Comput 2020;24:1153–69. http://dx.doi.org/10.1007/s00500-019-03951-2.

[22] Monteiro MS, Fontes DB, Fontes FA. An ant colony optimization algorithm to solve the minimum cost network flow problem with concave cost functions. In: Proceedings of the 13th annual conference on genetic and evolutionary computation. 2011, p. 139–46. http://dx.doi.org/10.1145/2001576.2001596.

[23] Yan S, Shih Y, Wang C. An ant colony system-based hybrid algorithm for square root concave cost transhipment problems. Eng Optim 2010;42:983–1001. http://dx.doi.org/10.1080/03052150903563751.

[24] Fontes DB, Gonçalves JF. Heuristic solutions for general concave minimum cost network flow problems. Netw: Int J 2007;50:67–76. http://dx.doi.org/10.1002/net.20167.

[25] Larsson T, Migdalas A, Rönnqvist M. A lagrangean heuristic for the capacitated concave minimum cost network flow problem. European J Oper Res 1994;78:116–29. http://dx.doi.org/10.1016/0377-2217(94)90126-0.

[26] Yang L, Yang Z. A sequential reduction algorithm for the large-scale fixed-charge network flow problems. Optim Lett 2023;1–19. http://dx.doi.org/10.1007/s11590-023-02040-6.

[27] Button K. Transport economics. Aldershot: Edward Elgar Publishing; 2010.

[28] Kraft D. A software package for sequential quadratic programming. Tech. rep. DFVLR-FB 88-28, Koln, Germany: DLR German Aerospace Center — Institute for Flight Mechanics; 1988.
[29] Powell MJD. A method for nonlinear constraints in minimization problems. Optimization 1969;283–98.
[30] Hestenes MR. Multiplier and gradient methods. J Optim Theory Appl 1969;4(5):303–20.
[31] Rockafellar RT. A dual approach to solving nonlinear programming problems by unconstrained optimization. Math Program 1973;5(1):354–73.
[32] Nocedal J, Wright SJ. Numerical optimization. Springer; 1999.
[33] Bertsekas DP. Constrained optimization and Lagrange multiplier methods. Academic Press; 2014.
[34] Fletcher R. A class of methods for nonlinear programming with termination and convergence properties. J Inst Math Appl 1970;6(1):76–90.
[35] Zangwill WI. Nonlinear programming via penalty functions. Manag Sci 1967;13(5):344–58.
[36] Kim D, Pardalos PM. A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. Oper Res Lett 1999;24:195–203. http://dx.doi.org/10.1016/S0167-6377(99)00004-8.