Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

A Dynamic Method to Solve the Fixed Charge Network Flow Problem ⋆

Zhibin Nie * **Shuning Wang** **

* *National Laboratory for Information Science and Technology, Department of Automation, Tsinghua University, Beijing 100084, China. (e-mail: nzb15@mails.tsinghua.edu.cn).*
** *National Laboratory for Information Science and Technology, Department of Automation, Tsinghua University, Beijing 100084, China. (e-mail: swang@mail.tsinghua.edu.cn)*

Abstract: This paper studies the widely applied fixed charge network flow problem (FCNFP), which is NP-hard. We approximate the FCNFP with a bilinear programming problem that is determined by a parameter ε. When ε is small enough, the optimal solution to the bilinear programming problem is the same as the optimal solution to the FCNFP. Therefore, solving the FCNFP can be transformed into solving a series of bilinear programming problems with decreasing ε. In this paper, these bilinear programming problems are solved by alternately solving two coupled linear programming problems. A dynamic method is proposed to update ε after solving one of the linear programming problems rather than solving the whole bilinear programming problem. Numerical experiments show the performance of the proposed method.

Keywords: Network flow problem, Fixed charge, Bilinear programming, Dynamic method.

1. INTRODUCTION

The fixed charge network flow problem (FCNFP) is widely applied in the field of network optimisation, in areas such as network design (Paraskevopoulosa et al. (2016)), production planning (Nasiri et al. (2014)), inventory management (Hovav and Tsadikovich (2015)) and transportation science (Moghaddam et al. (2019)). Theoretically, the FCNFP minimises the concave total cost function under linear constraints and is therefore NP-hard (Guisewite and Pardalos (1990)).

By adding a 0-1 variable for each arc to indicate whether there is some flow passing through, the FCNFP can be modelled as a mixed-integer linear programming (MILP) problem and then be solved exactly by branch-and-bound algorithms (F. Ortega (2003); Kowalski et al. (2014);Fontes et al. (2006);Palekar et al. (1990);G. Bernard (2014)). Another exact method is the vertex ranking algorithm (Murty (1968)), which is based on the property that the optimal solution to the concave minimisation problem on a convex polyhedron can be found at one of the vertices of the feasible region. However, these exact algorithms are only applicable to small-scale FCNFPs because their computational complexity increases exponentially with the problem scale. For large-scale FCNFPs, the calculations become unacceptably large.

A huge variety of heuristic algorithms have been presented to search for a suboptimal solution for large-scale FCNFPs. Walker (1976) proposed the heuristic adjacent extreme point algorithm, which escapes a local optimum by jumping over adjacent extreme points to resume iterating two or three extreme points away. Kim et al. (2006) and Kim and Pardalos (1999) proposed a dynamic slope scaling procedure (DSSP) to approximately solve the FCNFP. The DSSP transforms the FCNFP into a linear programming problem that is updated dynamically as the algorithm progresses. Nahapetyan and Pardalos (2008) transformed the FCNFP into a series of bilinear programming problems and then solved them by the adaptive dynamic cost updating procedure (ADCUP). Rebennack et al. (2009) combined the ADCUP with the branch-and-bound method. They used the ADCUP to find a suboptimal solution and then employed the suboptimal solution as a starting point to solve the FCNFP exactly by the branch-and-bound algorithm. Other classical heuristic methods (Lotfi and Moghaddamb (2013);Sherbiny and Alhamali (2013);Fontes and Goncalves (2007);Adlakha and Kowalski (2010);Hewitt et al. (2010)), such as the genetic algorithm and the particle swarm algorithm, were also applied to solve the FCNFP.

In these heuristic algorithms, the ADCUP has been proven to be very efficient. The objective function of the FCNFP is $f = \sum_{a \in A} f_a$, where A is the set of arcs and f_a is the cost function for arc a. In the ADCUP, the original cost function f_a with a fixed charge is underestimated by a piecewise linear function $\phi_a^{\varepsilon_a}$ with parameter ε_a. The smaller ε_a is, the smaller the difference between $\phi_a^{\varepsilon_a}$ and f_a. Let ε be the parameter vector consisting of all elements in set $\{\varepsilon_a | a \in A\}$. Then, the FCNFP can be approximated as a bilinear programming problem with parameter ε. Moreover, when ε is small enough, the FCNFP is equivalent to the bilinear programming

⋆ This project was supported by the National Natural Science Foundation of China (No. 61473165, U1813224) and the Science and Technology Innovation Committee of Shenzhen Municipality, China (JCYJ2017-0811-155131785).

problem. The ADCUP is an iterative algorithm. In each iteration, the ADCUP applies the dynamic cost updating procedure (DCUP) to find a locally optimal solution of the bilinear programming problem and then reduces the parameter ε based on the obtained solution until ε is small enough. This means that the ADCUP needs to implement a complete DCUP for each fixed ε, which is not efficient enough. Structurally, the DCUP consists of solving a series of coupled linear programming problems. To accelerate the update of ε, we propose the dynamic problem-updating procedure (DPUP), which updates ε after solving one linear programming problem instead of implementing the complete DCUP. In other words, the DPUP updates ε much more efficiently than the ADCUP. The quality of the solution obtained by the DPUP is also verified experimentally.

This paper is organised as follows: Section 2 formulates the FCNFP and approximates it as a bilinear programming problem. Section 3 presents the continuous bilinear algorithm and the DPUP. The validity and convergence of the DPUP are also analysed. Section 4 verifies the performance of the DPUP by numerical experiments. Section 5 concludes the paper.

2. PRELIMINARIES

2.1 The FCNFP

Let $G = (N, A)$ be a directed network with n nodes and m arcs, where N is the set of nodes and A is the set of arcs. Each arc $a \in A$ is associated with a flow x_a, a capacity u_a, and a cost function $f_a(x_a)$. In the FCNFP, $f_a(x_a)$ consists of two parts, the fixed cost s_a and the variable cost $c_a x_a$, where c_a is the unit cost. Therefore, $f_a(x_a)$ is discontinuous and can be expressed as

$$f_a(x_a) = \begin{cases} 0 & x_a = 0, \\ s_a + c_a x_a & x_a \in (0, u_a]. \end{cases} \tag{1}$$

Let x be the flow vector, B be the node-arc incidence matrix of G, and b be the node supply vector. The FCNFP can be formulated as the following problem:

$$\textbf{FCNFP:} \quad \begin{aligned} \min_x \ & f(x) = \sum_{a \in A} f_a(x_a) \\ \text{s.t.} \ & Bx = b \\ & 0 \le x \le u, \end{aligned} \tag{2}$$

where $x, u \in R^m$, $b \in R^n$, and $B \in R^{n \times m}$.

2.2 Approximation to the FCNFP

To address the discontinuity of $f_a(x_a)$, we approximate $f_a(x_a)$ as a continuous concave piecewise linear function

$$\phi_a^{\varepsilon_a}(x_a) = \begin{cases} c_a^{\varepsilon_a} x_a & x_a \in [0, \varepsilon_a) \ , \\ s_a + c_a x_a & x_a \in [\varepsilon_a, u_a], \end{cases} \tag{3}$$

where $c_a^{\varepsilon_a} = c_a + s_a/\varepsilon_a$.

Plots of $f_a(x_a)$ and $\phi_a^{\varepsilon_a}(x_a)$ are shown in Fig. 1. We can see that $\phi_a^{\varepsilon_a}(x_a)$ is an underestimate of $f_a(x_a)$. Specifically, we have $\phi_a^{\varepsilon_a}(x_a) < f_a(x_a)$ for $x_a \in (0, \varepsilon_a)$ and $\phi_a^{\varepsilon_a}(x_a) = f_a(x_a)$ for $x_a \in 0 \cup [\varepsilon_a, u_a]$.

For any ε with $\varepsilon_a \in (0, u_a], \forall a \in A$, a continuous piecewise linear network flow problem (CPLNFP) is defined as follows:

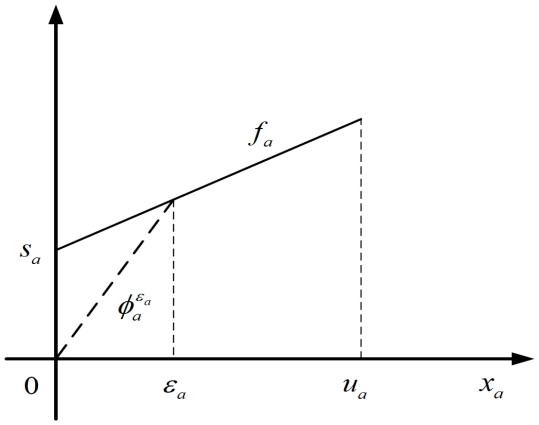

Fig. 1. Plots of $f_a(x_a)$ and $\phi_a^{\varepsilon_a}(x_a)$.

$$\textbf{CPLNFP}(\varepsilon): \quad \begin{aligned} \min_x \ & \phi^{\varepsilon}(x) = \sum_{a \in A} \phi_a^{\varepsilon_a}(x_a) \\ \text{s.t.} \ & Bx = b \\ & 0 \le x \le u. \end{aligned} \tag{4}$$

CPLNFP(ε) is an approximation to the FCNFP. Moreover, Nahapetyan and Pardalos (2008) has proven that, for sufficiently small ε, CPLNFP(ε) is equivalent to the FCNFP.

Let V be the set of vertices in the feasible region of the FCNFP. We define δ as $\delta = \min\{x_a | x \in V, x_a > 0, a \in A\}$. Let x^{ε} (x^*) be the optimal solution to CPLNFP(ε) (the FCNFP). The following theorem shows the relationship between CPLNFP(ε) and the FCNFP.

Theorem 1. (Nahapetyan and Pardalos (2008)). For any ε with $\varepsilon_a \in (0, \delta], \forall a \in A$, we have $\phi^{\varepsilon}(x^{\varepsilon}) = f(x^*)$.

Proof. See the proof of Theorem 2 in Nahapetyan and Pardalos (2008).

2.3 Relaxation of CPLNFP(ε)

Note that $\phi_a^{\varepsilon_a}(x_a)$ is a one-dimensional piecewise linear function with two segments. A binary variable y_a can be used to indicate which segment x_a is located in.

$$y_a = \begin{cases} 0 & x_a \in [0, \varepsilon_a), \\ 1 & x_a \in [\varepsilon_a, u_a]. \end{cases} \tag{5}$$

By replacing the binary constraint (5) with $0 \le y_a \le 1$, CPLNFP(ε) can be relaxed as the following continuous bilinear network flow problem (CBLNFP).

$$\textbf{CBLNFP}(\varepsilon): \quad \begin{aligned} \min_{x,y} \ & \varphi^{\varepsilon}(x) = \sum_{a \in A} (c_a^{\varepsilon_a} x_a + (s_a - \frac{\varepsilon_a}{s_a} x_a) y_a) \\ \text{s.t.} \ & Bx = b \\ & 0 \le x \le u \\ & 0 \le y \le 1. \end{aligned} \tag{6}$$

Fortunately, Rebennack et al. (2009) has proven that (x^*, y^*) is the optimal solution to CBLNFP(ε) if and only if x^* is the optimal solution to CPLNFP(ε). When ε is small enough, x^* is also the optimal solution to the FCNFP.

3. ALGORITHM

3.1 Continuous bilinear algorithm

Based on sections 2.2 and 2.3, we can solve the FCNFP by solving CBLNFP(ε) with a sufficiently small ε. However, it is difficult to obtain the applicable ε by definition. The following theorem provides a feasible method to obtain an applicable ε.

Theorem 2. (Rebennack et al. (2009)) For a specified ε, let (x^*, y^*) be the optimal solution to CBLNFP(ε). If

$$x_a^* \in 0 \cup [\varepsilon_a, u_a], \forall a \in A, \tag{7}$$

then x^* is also the optimal solution to the FCNFP.

Proof. See the proof of Corollary 3.2 in Rebennack et al. (2009).

We can start from a large parameter ε and iterate to reduce ε until the optimal solution to CBLNFP(ε) satisfies condition (7). The solution obtained in the last iteration is used to update the parameter ε in the next iteration. This process can be shown as the following continuous bilinear algorithm (CBA):

Algorithm 1 Continuous bilinear algorithm.

Require: Matrix B, vector b, u, parameter $\alpha \in (0, 1)$;
Ensure: Parameter ε, solution x^ε;
1: **Step 1:** Initialise: $\varepsilon = u$;
2: **Step 2:** Solve CBLNFP(ε) and obtain the optimal solution x^ε.
3: **Step 3:**
4: **if** $\exists x_a^\varepsilon \in (0, \varepsilon_a), a \in A$ **then**
- $A^\varepsilon = \{a | a \in A, x_a^\varepsilon \in (0, \varepsilon_a)\}$;
- $\varepsilon_a = \alpha \cdot \varepsilon_a, \forall a \in A^\varepsilon$;
- Go to **Step 2**;

5: **end if**
6: **return** ε, x^ε;

3.2 The dynamic method to update ε

CBLNFP(ε) is still concave and thus is computationally expensive to solve. From Step 2 of **Algorithm 1**, we can see that x^ε is only used to update ε before the last iteration. If we can use much less time to obtain a suboptimal solution to CBLNFP(ε), which can also update ε, the efficiency of the CBA can be greatly improved.

An efficient method to obtain a suboptimal solution to CBLNFP(ε) is the variable rotation method, which alternately fixes x and y and then solves the resulting linear programming problem. The variable rotation method divides CBLNFP(ε) into the following two coupled linear programming problems, which we refer to as $LP^\varepsilon(y)$ and $LP^\varepsilon(x)$.

$$\begin{aligned} &LP^\varepsilon(y): \\ &\min_x \ \varphi^\varepsilon(x) = \sum_{a \in A}((c_a^{\varepsilon_a} - \frac{s_a}{\varepsilon_a} y_a)x_a + s_a y_a) \\ &\text{s.t.} \ \ Bx = b \\ &\qquad 0 \le x \le u, \end{aligned} \tag{8}$$

$$\begin{aligned} &LP^\varepsilon(x): \\ &\min_y \ \varphi^\varepsilon(y) = \sum_{a \in A}((s_a - \frac{\varepsilon_a}{s_a} x_a)y_a + c_a^{\varepsilon_a} x_a) \\ &\text{s.t.} \ \ 0 \le y \le 1. \end{aligned} \tag{9}$$

The dynamic cost updating procedure (DCUP) (Nahapetyan and Pardalos (2007)) is a practical variable rotation algorithm to find a local optimum to CBLNFP(ε) by alternately solving $LP^\varepsilon(y)$ and $LP^\varepsilon(x)$.

In fact, it is not necessary to wait until the local optimum of CBLNFP(ε) is obtained before updating ε. When we solve CBLNFP(ε) by alternately solving $LP^\varepsilon(y)$ and $LP^\varepsilon(x)$, we can update ε once $LP^\varepsilon(y)$ is solved and then solve $LP^\varepsilon(x)$ with the new ε. The problem to be solved is updated synchronously with the parameter ε. Based on this update process, we propose the dynamic problem-updating procedure (DPUP). For a specified ε, the DPUP only needs to solve one linear programming problem. The outline of the DPUP can be seen in **Algorithm 2**.

Algorithm 2 Dynamic problem-updating procedure

Require: Matrix B, vector b, u, y^0, parameter $\alpha \in (0, 1)$;
Ensure: Parameter ε, solution x^ε;
1: **Step 1:** Initialise:
- $\varepsilon = u$;
- $y^\varepsilon = y^0$;

2: **Step 2:** Solve $LP^\varepsilon(y^\varepsilon)$ and obtain the optimal solution x^ε.
3: **Step 3:**
4: **if** $\exists x_a^\varepsilon \in (0, \varepsilon_a], a \in A$ **then**
- $A^\varepsilon = \{a | a \in A, x_a^\varepsilon \in (0, \varepsilon_a)\}$;
- $\varepsilon_a = \alpha \cdot \varepsilon_a, \forall a \in A^\varepsilon$;
- Solve $LP^\varepsilon(x^\varepsilon)$ and obtain the optimal solution y^ε;
- Go to **Step 2**;

5: **end if**
6: **return** ε, x^ε;

3.3 Analysis of the convergence

The following corollary shows the convergence of the DPUP for solving the FCNFP.

Corollary 1. The DPUP solves the FCNFP in a finite number of iterations.

Proof. In the worst case, each iteration of the DPUP updates the parameter ε_a for only one arc $a \in A$. Hence, for each arc $a \in A$, the maximum number of iterations needed is given by $I_a = I + 1$, where I is the smallest integer satisfying $\alpha^I u_a \le \delta$ and 1 is the step needed to check the stopping criterion. In actual implementation, all the arcs in A need only one step in total to check the stopping criterion. Therefore, the upper bound on the total number of iterations of the DPUP is given by

$$N_u = 1 + \sum_{a \in A} \max\{\lceil log_\alpha^{\frac{\delta}{u_a}} \rceil, 0\}. \tag{10}$$

4. NUMERICAL EXPERIMENTS

Numerical experiments are conducted in MATLAB 2014a on a Windows 10 platform with an Intel Core i7 3.2 GHz

processor and 16.0 GB of RAM. We compare the DPUP with the ADCUP and the CPLEX MIP Solver.

The ADCUP uses the complete DCUP to obtain a local optimum for CBLNFP(ε) and then updates the parameter ε. The CPLEX MIP Solver solves the following 0-1 mixed-integer programming problem, which is equivalent to the FCNFP.

$$\begin{aligned} \textbf{MIP-FCNFP}: \min_{x,y} \quad & f(x) = \sum_{a \in A} c_a x_a + s_a y_a \\ \text{s.t.} \quad & Bx = b \\ & 0 \le x_a \le u_a y_a, \forall a \in A \\ & y_a \in \{0,1\}, \forall a \in A. \end{aligned} \tag{11}$$

For small-scale problems, the CPLEX MIP Solver can obtain the exact solution of the FCNFP. However, for large-scale problems, the CPLEX MIP Solver cannot find the optimal solution in an acceptable time. This paper uses the CPLEX MIP Solver with version 12.6 and sets the acceptable time to 200 seconds. If the CUP running time exceeds 200 seconds, we say the MIP-FCNFP cannot be solved by the CPLEX MIP Solver.

The parameter α also affects the convergence speed of the DPUP and ADPUP. For fairness in the comparison, α is set to 0.5 for both the DPUP and ADPUP.

4.1 Test problems

The test problems are divided into 12 problem sets according to the network scale (the number of nodes (n) and the number of arcs (m)) shown in Table 1. Each problem set consists of 10 test problems with the same network scale, where networks are randomly generated by the benchmark network generator NETGEN [16]. In each test problem, the fixed charge s_a and the unit cost c_a for any arc $a \in A$ are randomly generated in $U[50, 100]$ and $[5, 15]$, respectively. The number of supply (demand) nodes is generated uniformly between 20% and 30% of the total nodes. The total supply flow is generated uniformly between 40 and 50 times the number of nodes and then randomly assigned to the supply nodes.

Table 1. Test problem sets.

Set	n	m	**Set**	n	m
1	20	100	**7**	200	4000
2	60	400	**8**	220	5000
3	120	1500	**9**	240	6000
4	140	2000	**10**	260	7000
5	160	2500	**11**	280	8000
6	180	3000	**12**	300	9000

According to whether they can be solved by the CPLEX MIP Solver, the test problems are classified into small-scale problems and large-scale problems. In our experiments, sets **1-2** consist of small-scale problems, and sets **3-12** consist of large-scale problems.

For ease of expression, a solver set S is defined as

$$S = \{D, A, C\},$$

where "D" denotes the DPUP, "A" denotes the ADCUP and "C" denotes the CPLEX MIP Solver. When a test problem is solved by $\zeta \in S$, we use f_ζ and T_ζ to denote the obtained objective function value and the CPU running time, respectively.

4.2 Computation results for small-scale problems

The small-scale problems in sets **1-2** are solved by the DPUP, ADCUP and CPLEX MIP Solver. The CPU running time, measured in seconds, is used to evaluate the solving efficiency. The objective function values are used to evaluate the solving accuracy. The smaller the objective function value is, the better the corresponding solution is. Since the solutions obtained by the DPUP and ADCUP are generally not global optimal solutions, we use the relative error (RE) to evaluate the accuracy.

$$\begin{aligned} RE_D(\%) &= \frac{f_D - f_C}{f_C} \times 100\%, \\ RE_A(\%) &= \frac{f_A - f_C}{f_C} \times 100\%. \end{aligned} \tag{12}$$

The computation results are shown in Table 2, where the bold item in each row represents the minimum CPU running time, the minimum objective function value or the smallest relative error for the corresponding test problem. It can be seen that the CPLEX MIP Solver can always obtain the best solution, but it consumes much more computation time than the DPUP and ADCUP. For all the test problems, we have

$$T_D < T_A < T_C. \tag{13}$$

Moreover, as the problem scale increases, the gap between T_D, T_A and T_C becomes wider. For the relative error, we can see that there are 13 (65%) test problems with $RE_D < RE_A$ and 7 (35%) test problems with $RE_D > RE_A$. That is, compared with the ADCUP, the DPUP has a higher probability of finding a better solution.

4.3 Computation results for large-scale problems

For the large-scale test problems in sets **3-12**, the CPLEX MIP Solver cannot obtain the exact solutions in an acceptable CPU running time (200 seconds). Therefore, the test problems in these 10 sets are solved by the DPUP and ADCUP. For each test problem, the time ratio (TR) is defined as

$$TR = T_A / T_D. \tag{14}$$

The average, minimum and maximum time ratios for test problems in each problem set are used to evaluate the solving efficiency.

For each test problem, since the exact solution cannot be obtained, we define the pseudo-error (PE) as

$$\begin{aligned} PE_D(\%) &= \frac{f_D - \min\{f_D, f_A\}}{\min\{f_D, f_A\}} \times 100\%, \\ PE_A(\%) &= \frac{f_A - \min\{f_D, f_A\}}{\min\{f_D, f_A\}} \times 100\%. \end{aligned} \tag{15}$$

Similarly, the average and maximum pseudo-error for the test problems in each problem set are used to evaluate the solving accuracy.

Moreover, for each problem set, we define the percentage of better values (PV) to compare the overall accuracy of the solutions obtained by the DPUP and ADCUP.

$$\begin{aligned} PV_D(\%) &= \frac{N_{f_D < f_A}}{N_p} \times 100\%, \\ PV_A(\%) &= \frac{N_{f_A < f_D}}{N_p} \times 100\%. \end{aligned} \tag{16}$$

Table 2. Performance of the DPUP, ADCUP and CPLEX MIP Solver for small-scale problems.

Set	Problem	Size		CPU Running Time (seconds)			Objective Function Value			RE(%)	
No.	No.	n	m	T_D	T_A	T_C	f_D	f_A	f_C	RE_D	RE_A
I	1	20	100	**0.0032**	0.0554	0.1389	2329	2342	**2075**	**12.24**	12.87
	2			**0.0013**	0.0313	0.1196	1639	1821	**1554**	**5.47**	17.18
	3			**0.0034**	0.0199	0.0721	2000	1831	**1664**	20.19	**10.04**
	4			**0.0025**	0.0301	0.1756	1647	1794	**1529**	**7.71**	17.33
	5			**0.0021**	0.0185	0.2357	1645	1633	**1488**	10.55	**9.75**
	6			**0.0016**	0.0259	0.0962	1649	1551	**1437**	14.75	**7.93**
	7			**0.0015**	0.0234	0.0858	1932	2075	**1867**	**3.48**	11.14
	8			**0.0024**	0.0416	0.1239	1636	1660	**1541**	**6.16**	7.72
	9			**0.0019**	0.0433	0.2382	1483	1580	**1381**	**7.39**	14.41
	10			**0.0012**	0.0421	0.2203	1655	1469	**1405**	17.79	**4.56**
I	1	60	400	**0.0063**	0.2314	11.975	4135	4340	**3831**	**7.94**	13.29
	2			**0.0061**	0.1583	26.907	4338	4455	**3933**	**10.30**	13.27
	3			**0.0043**	0.1766	24.819	4488	4765	**4141**	**8.38**	15.07
	4			**0.0062**	0.2697	129.15	5413	5802	**4764**	**13.62**	21.79
	5			**0.0051**	0.1617	5.8716	4710	4487	**3979**	19.27	**13.62**
	6			**0.0053**	0.0710	31.713	4504	4987	**4018**	**12.10**	24.12
	7			**0.0045**	0.1354	24.988	4766	4489	**4224**	12.83	**6.27**
	8			**0.0050**	0.2329	11.022	4724	4669	**4356**	8.45	**7.19**
	9			**0.0043**	0.1550	114.79	4701	5352	**4505**	**4.35**	18.80
	10			**0.0050**	0.1605	30.721	4836	4951	**4169**	**16.00**	18.76

Table 3. Performance of the DPUP and ADCUP for large-scale problems.

Set	Size		CPU Running Time (seconds)		Time Ratio	PE (%)		PV (%)	
No.	n	m	T_D	T_A	TR	PE_D	PE_A	PV_D	PV_A
			Aver (min,max)	Aver (min,max)	Aver (min,max)	Aver max	Aver max		
3	120	1500	0.0197 (0.0142,0.0335)	0.1029 (0.5782,1.0270)	41 (25,75)	0.39 2.55	4.39 8.91	70	30
4	140	2000	0.0266 (0.0206,0.0453)	1.6749 (1.0278,2.4983)	62 (31,97)	0.26 2.56	5.77 9.49	80	10
5	160	2500	0.0314 (0.0193,0.0494)	2.7123 (1.8815,3.8172)	86 (46,184)	0.59 4.32	5.33 12.1	90	10
6	180	3000	0.0894 (0.0546,0.1456)	10.136 (7.5152,15.067)	113 (58,205)	0.50 5.04	7.46 17.8	90	10
7	200	4000	0.1071 (0.0599,0.2120)	14.4167 (9.8830,19.415)	134 (92,287)	0 0	5.15 11.0	100	0
8	220	5000	0.1104 (0.0728,0.1363)	19.563 (13.099,28.850)	177 (112,287)	0 0	7.55 13.3	100	0
9	240	6000	0.1291 (0.0941,0.1735)	27.101 (13.968,37.794)	209 (122,339)	0 0	9.47 14.6	100	0
10	260	7000	0.1498 (0.1197,0.2509)	34.967 (29.189,43.484)	233 (169,289)	0 0	7.53 13.2	100	0
11	280	8000	0.1901 (0.1440,0.2600)	48.376 (38.765,68.336)	254 (130,339)	0 0	6.34 10.7	100	0
12	300	9000	0.2185 (0.2076,0.3098)	67.826 (48.038,80.180)	280 (174,372)	0 0	8.46 15.8	100	0

In (16), $N_{f_D<f_A}$ is the number of test problems where the objective function values obtained by the DPUP are smaller than those obtained by the ADCUP, $N_{f_A<f_D}$ is the number of test problems where the objective function values obtained by the ADCUP are smaller than those obtained by the DPUP, and N_p is the number of test problems in each problem set. In this paper, $N_p = 10$.

Computation results are shown in Table 3. We can see that for any problem set, the average (maximum or minimum) of T_D is always much less than the average (maximum or minimum) of T_A. The average, maximum and minimum of the TR are all much greater than 1. To further demonstrate the trend of TR with the problem scale, the computation results for TR are shown in Fig. 2, where the thick line represents the average of TR and each thin vertical line represents the range between the minimum and maximum of TR. As the problem scale increases, the average of TR increases approximately linearly.

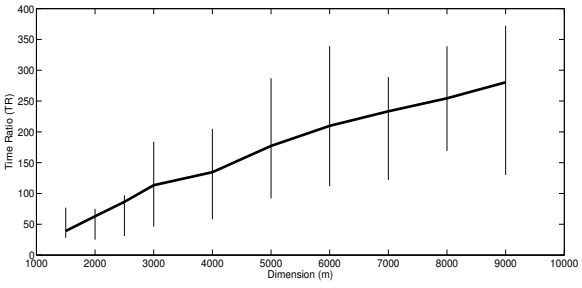

Fig. 2. Plots of the average, minimum and maximum of TR.

For each problem set, PE_D is clearly smaller than PE_A, and PV_D is clearly larger than PV_A. Moreover, for problem sets **7-12**, PE_D is 0 and PV_D is 100%, which means that the DPUP can obtain better solutions than the ADCUP for all test problems.

4.4 Summary

Considering the solving efficiency, the DPUP always shows obvious superiority over the ADCUP in problems of different scales.

In regard to optimising capacity, neither the DPUP nor the ADCUP can obtain the global optimal solution in most cases. For small-scale problems, the accuracy of the solution obtained by the DPUP and ADCUP is related to the test problem itself. However, the DPUP statistically obtains better solutions in more test problems. For very large-scale problems, the DPUP can always obtain better solutions.

In general, compared with the ADCUP, the DPUP possesses an obvious superiority in both solving efficiency and optimising capability.

5. CONCLUSION

The motivation for our study comes from the wide application of the FCNFP. We transform the task of solving the FCNFP into solving a series of bilinear programming problems. The major contribution of this paper is the DPUP, which is used to update the bilinear programming problem dynamically. The superiority of the DPUP is that it only needs to solve a linear programming problem instead of solving the complete bilinear programming problem before updating the parameter ε. In numerical experiments, the performance of the DPUP is evaluated by comparison with the ADCUP and CPLEX MIP Solver in randomly generated test problems.

REFERENCES

Adlakha, V. and Kowalski, K. (2010). A heuristic algorithm for the fixed charge problem. *Opsearch*, 47, 166–175.

F. Ortega, L.A.W. (2003). A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, 41, 143–158.

Fontes, D. and Goncalves, J. (2007). Heuristic solutions for general concave minimum cost network flow problems. *Networks*, 50, 67–76.

Fontes, D., Hadjiconstantinou, E., and Christofides, N. (2006). A branch-and-bound algorithm for concave network flow problems. *Journal of Global Optimization*, 34, 127–155.

G. Bernard, L.M. (2014). Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 2, 55–75.

Guisewite, G.M. and Pardalos, P.M. (1990). Minimum concave-cost network flow problems: applications, complexity, and algorithms. *Annals of Operations Research*, 25, 75–99.

Hewitt, M., Nemhauser, G., and Savelsbergh, M. (2010). Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *Informs Journal on Computing*, 22, 314–325.

Hovav, S. and Tsadikovich, D. (2015). A network flow model for inventory management and distribution of influenza vaccines through a healthcare supply chain. *Operations Research for Health Care*, 5, 49–62.

Kim, D., Pan, X., and Pardalos, P. (2006). An enhanced dynamic slope scaling procedure with tabu scheme for fixed charge network flow problems. *Computational Economics*, 27, 273–293.

Kim, D. and Pardalos, P.M. (1999). A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters*, 24, 195–203.

Kowalski, K., Levb, B., Shen, W., and Tu, Y. (2014). A fast and simple branching algorithm for solving small scale fixed-charge transportation problem. *Operations Research Perspectives*, 1, 1–5.

Lotfi, M. and Moghaddamb, R. (2013). A genetic algorithm using priority-based encoding with new operators for fixed charge transportation problems. *Applied Soft Computing*, 13, 2711–2726.

Moghaddam, S., Keshteli, M., and Mahmoodjanloo, M. (2019). New approaches in metaheuristics to solve the fixed charge transportation problem in a fuzzy environment. *Neural Computing and Applications*, 31.

Murty, K. (1968). Solving the fixed charge problem by ranking the extreme points. *Operations Research*, 16, 268–279.

Nahapetyan, A. and Pardalos, P. (2008). Adaptive dynamic cost updating procedure for solving fixed charge network flow problems. *Computational Optimization and Applications*, 39, 37–50.

Nahapetyan, A. and Pardalos, P. (2007). A bilinear relaxation based algorithm for concave piecewise linear network flow problems. *Journal of Industrial and Management Optimization*, 3, 71–85.

Nasiri, G., Zolfaghari, R., and Davoudpour, H. (2014). An integrated supply chain production–distribution planning with stochastic demands. *Computers & Industrial Engineering*, 77, 35–45.

Palekar, U., Karwan, M., and Zionts, S. (1990). A branch-and-bound method for the fixed charge transportation problem. *Management Science*, 36, 1092–1105.

Paraskevopoulosa, D., Bektasb, T., Gabriel, T., Chris, C., and N.Pottsd (2016). A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem. *European Journal of Operational Research*, 253, 265–279.

Rebennack, R., Nahapetyan, A., and Pardalos, P. (2009). Bilinear modeling solution approach for fixed charge network flow problems. *Optimization Letters*, 3, 347–355.

Sherbiny, M. and Alhamali, R. (2013). A hybrid particle swarm algorithm with artificial immune learning for solving the fixed charge transportation problem. *Computers & Industrial Engineering*, 64, 610–620.

Walker, W. (1976). A heuristic adjacent extreme point algorithm for the fixed charge problem. *Management Science*, 22, 587–596.