Branch and Bound and Dynamic Programming Approaches for the Path Avoiding Forbidden Pairs Problem

Daniele Ferone, Paola Festa, and Matteo Salani

Abstract We propose a branch and bound (B&B) and a dynamic programming algorithm for the Path Avoiding Forbidden Pairs Problem (PAFPP). Given a network and a set of forbidden node pairs, the problem consists in finding the shortest path from a source node s to a target node t, avoiding to traverse both nodes of any of the forbidden pairs. The problem has been shown to be NP-complete. In this work, we describe the problem, its mathematical model and we propose two exact algorithms. We compare their performances against those of a commercial solver solving instances for two different graph topologies: fully random graphs and grid graphs.

Keywords Branch and bound · Dynamic programming · Constrained shortest paths · Forbidden pairs

1 Introduction

In this work, we study the *Path Avoiding Forbidden Pairs Problem* (PAFPP). The problem asks to find the shortest path between two nodes s and t in a given weighted directed graph $G = (V, A)$ or recognize that such path does not exist. The shortest path should not visit both nodes belonging to a set $F \subset (V \times V)$ of node pairs called

D. Ferone (✉)
Department of Mechanical, Energetic and Management Engineering, University of Calabria, Rende, Italy
e-mail: daniele.ferone@unical.it

P. Festa
Department of Mathematics and Applications, University of Napoli FEDERICO II, Naples, Italy
e-mail: paola.festa@unina.it

M. Salani
Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, Lugano, Switzerland
e-mail: matteo.salani@idsia.ch

R. Cerulli et al. (eds.), *Optimization and Decision Science*, AIRO Springer Series 7,
https://doi.org/10.1007/978-3-030-86841-3_19

forbidden pairs. Paths containing at most one vertex from each pair in F are called F-paths.

The PAFPP has been introduced in [12, 15] to design test cases for automatic software validation, where the nodes of the graph represent segments of code and edges represent the control flow. The goal is to cover the graph with $s - t$ paths corresponding to different test cases, introducing forbidden pairs which identify the mutually exclusive code segments.

A special case of PAFPP arises in bio-informatics, tackling the problem of peptide sequencing via tandem mass spectrometry. Chen et al. [3] model the peptide sequencing as a PAFPP on a directed acyclic graph.

PAFPP emerges in Ferone et al. [6] as a subproblem of the constrained shortest path problem named *Constrained Shortest Path Tour Problem (CSPTP)*. Authors reduce the CSPTP to the PAFPP and solve it with a branch and bound strategy.

Lastly, Ceselli et al. [2] solve a rich Vehicle Routing Problem using Branch and Price. Here the PAFPP structure emerges in the pricing problem modeling compatibility constraints between customers.

Gabow et al. [9] proves the NP-hardness of PAFPP, which is polynomially solvable under *skew symmetry* conditions [16]. Kolman and Pangrác [10] studies the complexity of PAFPP under different assumptions, showing that the problem remains NP-complete even if the graph is planar or presents an halving structure, but it becomes polynomial when the graph has a hierarchical structure. These results are extended in [11], proving that the PAFPP is NP-hard when the set of forbidden pairs has an overlapping structure or is sorted. Finally, Blanco et al. [1] presents a polyhedral study of the PAFPP.

The paper is organized as follows. In Sect. 2 we present the mathematical model of the problem. In Sects. 3 and 4 we present the solution approaches and the computational results, respectively. In Sect. 5 we conclude and discuss some future work.

2 Mathematical Formulation

Let $G = (V, A)$ be a weighted directed graph, where $V = \{1, \ldots, n\}$ is the set of nodes, and $A = \{(i, j) \in V \times V : i, j \in V \wedge i \neq j\}$ is the set of m arcs. Let $C : A \rightarrow \mathbb{R}_0^+$ a function that assigns a non-negative cost c_{ij} to each arc $(i, j) \in A$. For each node $i \in V$, let $FS(i) = \{j \in V : (i, j) \in A\}$ and $BS(i) = \{j \in V : (j, i) \in A\}$ be the forward star and backward star of node i, respectively. Given a source node $s \in V$ and a destination node $t \in V$, the PAFPP can be modeled with the following $0 - 1$ integer program:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1a}$$

s.t.

$$\sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = \begin{cases} 1, & i = s; \\ -1, & i = t; \\ 0, & \text{otherwise}; \end{cases} \tag{1b}$$

$$\sum_{j \in BS(a)} x_{ja} + \sum_{j \in BS(b)} x_{jb} \leq 1 \qquad \forall\, (a, b) \in F \tag{1c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in A. \tag{1d}$$

The objective function (1a) minimizes the path length. Constraints (1b) model the flow balance at each node. Constraints (1c) guarantee that two nodes belonging to a forbidden pair are never visited simultaneously.

3 Solution Approaches

In this section, we describe two exact approaches to solve the PAFPP. In particular, we present a branch and bound algorithm (B&B) in Sect. 3.1, and a dynamic programming algorithm in Sect. 3.2.

3.1 Branch and Bound Approach

Observing the mathematical model (1a)–(1d), it is evident that relaxing the constraints (1c) we obtain the model of a Shortest Path Problem (SPP), which is polynomially solvable (for example, with the Dijkstra's algorithm [5]).

Therefore, we devise a B&B algorithm using a polynomial algorithm for the SPP to compute a combinatorial bound and performing branching operations when the optimal solution of the relaxation results infeasible for the PAFPP.

Let G^t be the graph associated to a generic iteration t of the B&B, let P^t be the optimal solution of the relaxed problem PAFPP ^t_R on G^t and let UB be the value of an incumbent feasible solution. If the value of P^t is not less than UB, then the graph G^t does not contain any improving solution and P^t can be disregarded. Instead, if P^t does not contain any forbidden pair, then it is also feasible for PAFPP defined on G and it improves the incumbent solution. On the contrary, if P^t contains both nodes of any forbidden pair, two sub-problems are generated.

In particular, let (v, w) one of the forbidden pairs violated by P^t, two graphs G^{t^1} and G^{t^2} are generated and associated to the branching nodes t^1 and t^2, respectively. The graphs G^{t^1} and G^{t^2} are obtained removing from G^t the nodes v and w, respectively. More formally, $A^{t^1} = A^t \setminus \{v\}$ and $A^{t^2} = A^t \setminus \{w\}$.

Obviously, as in classic B&B framework, if the sub-problem of iteration t is not feasible – i.e., it does not exist an $s - t$ path due to the removed nodes—the node is

not further branched. The incumbent best solution found is optimal and returned as final solution.

3.2 Dynamic Programming for PAFPP

Dynamic programming have been extensively and successfully applied to constrained shortest path problems [4, 7]. Therefore, we solve PAFPP to optimality by a bi-directional dynamic programming algorithm [13] implementing the Decremental State Space Relaxation (DSSR) strategy [14].

A state associated with vertex $i \in N$ represents a partial path from the source node s to the node i. Different states can be associated with the same node and they correspond to different partial paths.

The dynamic programming algorithm iteratively extends states until no further extensions are possible. Among all feasible states reaching the destination node d the one with minimal cost represent the optimal solution to PAFPP.

Each state is encoded in a label, in bi-directional dynamic programming called *forward* and *backward* labels. A forward label associated with node $i \in N$ is a tuple:

$$l_i^f = (i, c_i, S, B), \tag{2}$$

where i is the last node visited in the partial path, c_i is the accumulated cost, S is a binary vector that keep tracks of the visited nodes in the partial path and B is a binary vector with size $|B| = |F|$. In vector B, $b_j \in B = 1$ indicates that one of the nodes belonging to the jth forbidden pair is visited along the partial path. Note that, S does not keep any information about the order in which the vertices are visited. Similarly, a backward label associated with node $i \in N$, corresponding to paths from node i to destination node d, is a tuple:

$$l_i^b = (i, c_i, S, B), \tag{3}$$

where tuple's elements have the same meaning as those of forward labels.

The dynamic programming algorithm extends all feasible forward and backward labels to generate new forward and backward labels. The extension of a forward label corresponds to appending an additional arc (i, j) to a path from s to i, obtaining a path from s to j, while the extension of a backward label corresponds to appending an additional arc (j, i) to a path from i to d, obtaining a path from j to d.

The binary vector B is used to avoid visiting pair of nodes belonging to a forbidden pair while vector S is used to avoid cycles. Anyway, as the cost matrix is non-negative and triangular inequality holds, all partial paths with cycles are suboptimal and can be safely ignored. When a label $l_i = (i, c_i, S, B)$ is extended to a vertex j, a new label $l_j = (j, c_j, S', B')$ is generated. The update rules of the

vectors S and B are as follows:

$$S'_k = \begin{cases} S_k + 1, & k = j; \\ S_k, & k \neq j; \end{cases} \tag{4}$$

$$B'_k = \begin{cases} B_k + 1, & (a,b)_k \in F,\ a_k = j \vee b_k = j; \\ B_k, & \text{otherwise.} \end{cases} \tag{5}$$

A label $l_i = (i, c_i, S, B)$ is feasible if $S_k \leq 1$ and $B_f \leq 1$ for all $k \in N$ and all $f \in F$, respectively.

The effectiveness of dynamic programming depends on the number of generated labels. In order to control the number of labels, dominance tests are performed. Let $l' = (i, c'_i, S', B')$ and $l'' = (i, c''_i, S'', B'')$ be two labels associated with node i. l' dominates l'' and label l'' can be safely discarded only if

$$c'_i \leq c''_i; \tag{6}$$

$$B'_f \leq B''_f, \quad \forall f \in F \tag{7}$$

and at least one inequality is strict. Please note that only vector B participates in the domination criterion.

In bi-directional dynamic programming forward and backward labels are joined to produce complete paths from node s to node d. Let $l_i^f = (i, c_i^f, S^f, B^f)$ a forward label and $l_i^b = (i, c_i^b, S^b, B^b)$. The join is feasible if

$$S_k^f + S_k^b \leq 1, \quad \forall k \in N; \tag{8}$$

$$B_f^f + S_f^b \leq 1, \quad \forall f \in F. \tag{9}$$

The join condition ensures that the final path does not contain cycles nor visit both nodes of a forbidden pairs. Even if the vector S is not included in domination conditions, it is guaranteed that none of the optimal paths is eliminated. Indeed, suppose that a join operation is prohibited because a node k is visited in both forward and backward labels. As the cost matrix is positive and triangular inequality holds, there must be non dominated forward and backward labels where node k is not visited and the join is feasible and more profitable.

We reduce the number of labels by selecting a monotone resource and extend labels for which the resource consumption is less than half of a given threshold T. In PAFPP, the only available monotone resource is the accumulated cost. In order to apply the bi-directional dynamic programming algorithm we compute an upper bound to the optimal cost $\bar{c^*}$ as the threshold T.

Decremental state space relaxation (DSSR) introduced by Righini and Salani [14] aims at reducing the number of states to be explored by dynamic programming. For PAFPP, the basic idea is that not all forbidden pairs are tracked in the vector B

and are therefore not imposed in the domination criterion. If the optimal solution visits both nodes of a forbidden pair, the corresponding pair is added to the vector B and the process is iterated. We remark that at each iteration a lower bound to the optimal solution is computed.

4 Computational Results

We present some preliminary results to compare the performances of DSSR and B&B against those of a commercial solver (IBM CPLEX) directly solving the model (1a)–(1d). A time limit of 10 min has been used for each solution method.

The instances were randomly generated through an adaption of the generator presented in [8] and can be divided in two classes: fully random and grid graphs.

The number m of edges in the random graphs has been selected to be in $\{5 \cdot n, 10 \cdot n, 15 \cdot n\}$, where $n = |V|$. The total number of forbidden pairs for each instance is in $\{25 \cdot n, 30 \cdot n, 35 \cdot n\}$. Since the grid graphs are more sparse, the number of forbidden pairs in grid graphs ranges in $\{\lfloor 6.25 \cdot n \rfloor, \lfloor 12.50 \cdot n \rfloor, \lfloor 18.75 \cdot n \rfloor, \lfloor 25 \cdot n \rfloor, \lfloor 30 \cdot n \rfloor\}$.

Each combination of graph size characterizes a collection of similar instances, denoted as $\{R1, \ldots, R9, G1, \ldots, G3\}$. Each random (grid) collection contains 30 (50) different instances of the same type, 10 for each different number of forbidden pairs. The characteristics of the data-set are summarized in Table 1.

The computational results obtained by CPLEX, B&B, and the dynamic programming approach (DSSR) are reported in Table 2. For each instance type, we report the time spent by the algorithms in solving instances of that type (avg. time), and the number of instances of that type for which a proved optimal (O) solution has been found.

The results highlight that in spite of their size the random graphs are much easier to solve respect to the grid networks. This was an expected result, since random graphs are denser respect to the grid graphs, it is therefore much easier to find an

Table 1 Instance parameters

Fully random graphs			Grid graphs	
Problem	Nodes	Arcs	Problem	size
R1	1500	7500	G1	200 × 200
R2	1500	15,000	G2	200 × 400
R3	1500	22,500	G3	300 × 300
R4	2000	10,000		
R5	2000	20,000		
R6	2000	30,000		
R7	2500	12,500		
R8	2500	25,000		
R9	2500	37,500		

Table 2 Experimental results

	CPLEX		B&B		DSSR	
	O	Avg. time	O	Avg. time	O	Avg. time
G1	27	363.28	18	387.05	10	481.58
G2	28	382.40	30	314.45	14	452.67
G3	23	362.71	27	283.32	15	434.85
Average	*26*	*369.46*	*25*	*328.27*	*13*	*456.37*
R1	30	1.35	30	0.00	30	0.00
R2	30	2.89	30	0.00	30	0.01
R3	30	4.63	30	0.00	30	0.01
R4	30	1.97	30	0.00	30	0.01
R5	30	3.94	30	0.00	30	0.01
R6	30	6.46	30	0.00	30	0.01
R7	30	2.48	30	0.00	30	0.01
R8	30	5.32	30	0.01	30	0.01
R9	30	8.59	30	0.01	30	0.02
Average	*30*	*4.18*	*30*	*0.00*	*30*	*0.01*

alternative path that does not violates any forbidden pair. For random graphs, all the algorithms are very fast (less than 5 s in average). Both B&B and DSSR get the optimal solution for all the instances. It is worthy to note that all the instances were generated to be non-trivial, i.e. the simple shortest path $s - t$ contains at least one forbidden pair.

On the other hand, the grid graphs are more challenging. In this case, CPLEX misses the optimum in 54 out of 150 cases, B&B is not able to find the optimal solution in 48 cases, and DSSR fails in 111 cases. This is caused by the sparsity of the grid graphs that induces a lower number of feasible $s - t$ paths.

Figure 1 illustrates the number of optimal solutions found with respect to the ratio of forbidden pairs over the number of nodes in the network. The B&B and CPLEX approaches show a decreasing trend: for an increasing number of forbidden pairs, the instances become more challenging and, generally, the methods find less optimal solutions. Instead, the DSSR does not seem to be strongly influenced by the number of forbidden pairs as no clear trend is visible.

These are preliminary results, but they give valuable information. When instances are not extremely challenging (Random), both B&B and DSSR perform well presenting an high converge speed to the optimum. Meanwhile, on sparse graphs B&B has similar performances with respect to CPLEX, but solving a higher number of instances. DSSR is the worst approach on these problems. Nevertheless, we believe that DSSR can be strongly improved with the use of an upper bound that permits to prune many feasible labels.

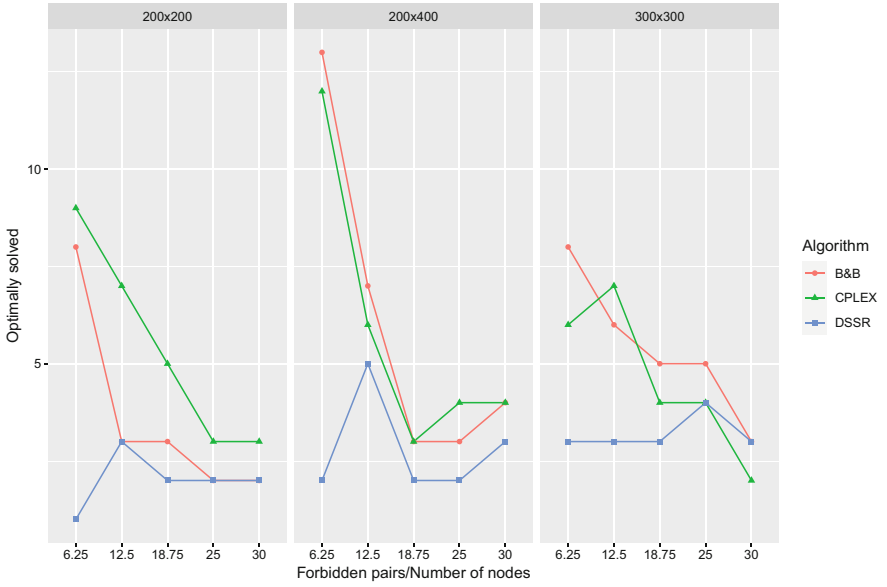

Fig. 1 Optimal solutions found on grid graphs with respect to the relative number of forbidden pairs

5 Conclusions and Future Work

This paper presents the Path Avoiding Forbidden Pairs Problem (PAFPP). We propose two exact algorithms to solve the problem to proven optimality: a branch and bound (B&B) algorithm and a dynamic programming (DSSR) algorithm. Some preliminary results are compare the performance of the methods against those of a commercial solver. The results evidence that on Random instances the two approaches are very performing. On the other hand, on sparse instances B&B seems to be equivalent to CPLEX while DSSR needs to be improved.

As future research perspectives, we are sure to be able to obtain a better upper bound to improve the performance of DSSR and we plan to better investigate the instances' properties that have an impact to the performance of the algorithms.

References

1. Blanco, M., Borndörfer, R., Brückner, M., Hoàng, N.D., Schlechte, T.: On the path avoiding forbidden pairs polytope. Electron. Notes Discrete Math. **50**, 343–348 (2015). https://doi.org/10.1016/j.endm.2015.07.057
2. Ceselli, A., Righini, G., Salani, M.: A column generation algorithm for a vehicle routing problem with economies of scale and additional constraints. Transp. Sci. **43**(1), 56–69 (2009). https://doi.org/10.1287/trsc.1080.0256

3. Chen, T., Kao, M.Y., Tepel, M., Rush, J., Church, G.M.: A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. J. Comput. Biol. **8**(3), 325–337 (2001). PMID: 11535179. https://doi.org/10.1089/10665270152530872
4. Di Puglia Pugliese, L., Ferone, D., Festa, P., Guerriero, F.: Shortest path tour with time windows. Eur. J. Oper. Res. **282**(1), 334–344 (2020). https://doi.org/10.1016/j.ejor.2019.08.052
5. Dijkstra, E.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959). https://doi.org/10.1007/BF01386390
6. Ferone, D., Festa, P., Guerriero, F., Laganà, D.: The constrained shortest path tour problem. Comput. Oper. Res. **74**, 64–77 (2016). JCR. https://doi.org/10.1016/j.cor.2016.04.002
7. Ferone, D., Festa, P., Fugaro, S., Pastore, T.: A dynamic programming algorithm for solving the k-color shortest path problem. Optim. Lett. (2020). https://doi.org/10.1007/s11590-020-01659-z
8. Festa, P., Pallottino, S.: A pseudo-random networks generator. Tech. rep., Department of Mathematics and Applications ''R. Caccioppoli'', University of Napoli FEDERICO II (2003)
9. Gabow, H.N., Maheshwari, S.N., Osterweil, L.J.: On two problems in the generation of program test paths. IEEE Trans. Softw. Eng. **SE-2**(3), 227–231 (1976). https://doi.org/10.1109/TSE.1976.233819
10. Kolman, P., Pangrác, O.: On the complexity of paths avoiding forbidden pairs. Discrete Appl. Math. **157**(13), 2871–2876 (2009). https://doi.org/10.1016/j.dam.2009.03.018
11. Kováč, J.: Complexity of the path avoiding forbidden pairs problem revisited. Discrete Appl. Math. **161**(10–11), 1506–1512 (2013). https://doi.org/10.1016/j.dam.2012.12.022
12. Krause, K., Goodwin, M., Smith, R.: Optimal software test planning through automated network analysis. TRW Systems Group (1973)
13. Righini, G., Salani, M.: Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optim. **3**(3), 255–273 (2006). https://doi.org/10.1016/J.DISOPT.2006.05.007
14. Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks Int. J. **51**(3), 155–170 (2008). https://doi.org/10.1002/net.20212
15. Srimani, P.K., Sinha, B.P.: Impossible pair constrained test path generation in a program. Inf. Sci. **28**(2), 87–103 (1982). https://doi.org/10.1016/0020-0255(82)90019-6
16. Yinnone, H.: On paths avoiding forbidden pairs of vertices in a graph. Discrete Appl. Math. **74**(1), 85–92 (1997). https://doi.org/10.1016/S0166-218X(96)00017-0