HEURISTIC SEARCH ALGORITHMS TO DETECT COLLUSIVE OPPORTUNITIES IN DEREGULATED ELECTRICITY MARKETS

by

ELİF YILMAZ

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of Master of Science

Sabancı University
September 2020

HEURISTIC SEARCH ALGORITHMS TO DETECT COLLUSIVE OPPORTUNITIES IN DEREGULATED ELECTRICITY MARKETS

Approved by:

Prof. Güvenç Şahin ....
(Thesis Supervisor)

Assist. Prof. Emre Çelebi ....

Assist. Prof. Burak Kocuk ....

Date of Approval: September 2, 2020

ABSTRACT

HEURISTIC SEARCH ALGORITHMS TO DETECT COLLUSIVE OPPORTUNITIES IN DEREGULATED ELECTRICITY MARKETS

ELİF YILMAZ

INDUSTRIAL ENGINEERING M.Sc. THESIS, SEPTEMBER 2020

Thesis Supervisor: Prof. Güvenç Şahin

Keywords: tacit collusion, deregulated electricity markets, heuristic search algorithm, bi-level programming

In deregulated electricity markets, the main objective is to maintain a competitive trading environment and satisfy demand at lowest possible cost. However, sustaining a competitive environment is challenging and collusion among the Power Generation Companies (GenCos) might exist. Therefore, the Independent System Operator (ISO) controls the auction mechanism as a decision-maker for the electricity distribution. In order to guide the ISO to detect collusion, we develop a search algorithm and its variations by using the bi-level programming problem in Aliabadi, Kaya & Şahin (2016). We create 26 instances of 3 different problem sizes to test the performance of the algorithms. We compare the results of the algorithms to the total enumeration algorithm, which is an exact method to detect collusion but may not be executed in reasonable time, and among themselves. Moreover, we experiment with the algorithms using alternative single-level formulations of the bi-level programming problem.

ÖZET

SERBESTLEŞMİŞ ELEKTRİK PİYASALARINDA GİZLİ ANLAMAŞLARI TESPİT ETMEK İÇİN SEZGİSEL ARAMA ALGORİTMALARI

ELİF YILMAZ

ENDÜSTRİ MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, EYLÜL 2020

Tez Danışmanı: Prof. Dr. Güvenç Şahin

Anahtar Kelimeler: gizli anlaşma, serbestleşmiş elektrik piyasası, sezgisel arama algoritması, iki seviyeli programlama

Serbestleştirilmiş elektrik piyasalarının temel amacı rekabetçi ticaret ortamını sürdürmek ve talebi mümkün olan en düşük maliyetle karşılamaktır. Ancak, rekabetçi pazarın sürdürülmesi zordur ve elektrik üretim şirketleri (GenCos) arasında gizli anlaşma olabilir. Bu nedenle, bağımsız sistem operatörü (ISO) elektrik dağıtımında karar verici olarak ihale mekanizmasını kontrol eder. ISO'nun gizli anlaşmaları tespit edebilmesi için, Aliabadi et al. (2016)'da tanımlanan iki seviyeli programlama problemini kullanan bir arama algoritması geliştirilmiş, ve bu algoritmanın varyasyonları oluşturulmuştur. Algoritmaları uygulayabilmek için 3 farklı problem büyüklüğünde 26 örnek oluşturulmuştur. Algoritmaların sonuçları gizli anlaşmaları tespit etmek için pekin yöntem olan, ancak anlamlı bir sürede uygulanamayan tümden sayma algoritmasıyla ve kendi aralarında karşılaştırılmıştır. Ayrıca, algoritmalar iki seviyeli programlama probleminin farklı tek seviyeli formülasyonları ile de uygulanmıştır.

ACKNOWLEDGEMENTS

First of all, I want to thank my supervisor Prof. Güvenç Şahin for his support throughout my master's degree. I am grateful for his precious guidance and the opportunities that he has provided me. I would like to express my sincere gratitudes to TÜBİTAK for giving me the opportunity to work for the project 117M589 during my study. I would also like to thank Dr. Murat Elhüseyni for his contributions throughout this research.

I am appreciative of my thesis jury members, Assist. Prof. Emre Çelebi and Assist. Prof. Burak Kocuk, not only for their valuable time but also for the knowledge they have added to me.

I would particularly thank Erhun Kundakcıoğlu, who encouraged me to get a master's degree. I always feel grateful for his inspiration and invaluable support.

Finally, I wish to thank Semih Boz, Simge Güçlükol, Duygu Ay, Polen Arabacı and Maryam Toufani for their assistances, understanding and support. I have learned a lot from them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATONS

1. INTRODUCTION

In the 1990s, several countries started to liberalize the electricity markets. Today, most of the produced electricity is traded in deregulated markets. The main goal of the deregulated electricity markets is to encourage the competitive environment and satisfy consumer demand at a low price.

There are two types of players in the deregulated electricity market system: Power Generation Companies and Independent System Operator. Power Generation Companies (GenCos) generate electricity and sell it to consumers. In the day-ahead market, the GenCos declare their unit prices and production capacities for the electricity a day before the auction. The Independent System Operator (ISO) controls the auction mechanism and studies the day-ahead market periodically to designate the produced electricity based on the GenCos' bids and production capacities. The decision-making process of the ISO for the distribution is called the market-clearing process.

In the deregulated electricity markets, creating a competitive environment is challenging. Moreover, collusion might exist. Explicit collusion exists when GenCos make an agreement among themselves. Tacit collusion exists according to the GenCos strategic behaviors when there is no explicit collusion. Guan, Ho & Pepyne (2001), Sweeting (2007) and Fabra & Toro (2005) showed tacit collusion might exist based on strategic behaviors of the GenCos. However, it is a challenging task to identify a collusion.

Aliabadi et al. (2016) proposed an exact method to detect collusion, however, it is not solvable in polynomial time. They also developed a multi-objective non-linear bi-level programming problem to detect collusion in a deregulated electricity market, which can be solved as an Linear Programming (LP). In this study, we propose a search algorithm by using the problem formulation defined by Aliabadi et al. (2016) in order to characterize the solution space. Furthermore, with the aim of improving the search algorithm, we develop its variants. We compare the algorithms with the exact method and among themselves.

The rest of the thesis is organized as follows: Chapter 2 covers the literature review related to collusion in deregulated electricity markets. In Chapter 3, we introduce the problem in detail. A search algorithm and its variations are proposed in Chapter 4. Chapter 5 presents our experiments and computational results. The thesis concludes by Chapter 6.

2. LITERATURE REVIEW

In the deregulated electricity markets, the aim is to clear the market in a competitive environment and maintain social welfare. Collusion between the GenCos will disrupt the competitive environment. Therefore, there are studies in order to detect collusion.

In the literature, different models for the strategic behaviors of the GenCos are presented such as Cournot, Bertrand, Stackelberg, Conjectural Variation, and Supply Function Equilibria (SFE). For example, Ruiz & Conejo (2008) apply the Cournot model and Ruiz, Conejo & Arcos (2010) apply Conjectural Variation to short-term electricity markets. Mainly, tacit collusion in the electricity markets is studied with two modeling approaches: Simulation-based models and optimization models within a game-theoretic framework. Guan et al. (2001), Blume & Heidhues (2008) and Bernheim & Whinston (1990) explore game-theoretic approaches for repeated games for tacit collusion. Since the complexity of computing Nash equilibrium for repeated games is high, often the physical limitations of transmission lines are not considered such as in Blume & Heidhues (2008) and Ruiz, Kazempour & Conejo (2012). Lee & Baldick (2003) propose an approach to find the Nash equilibrium in deregulated electricity markets, where the transmission constraints complicate the market clearing mechanism and cause the payoff functions to be non-differentiable and non-concave. However, there is a difficulty in finding the kernel of the solution because it searches a large number of candidate kernel sub-matrices, which means when the number of players increases, the difficulty increases.

There are two types of simulation models: Equilibrium models and agent-based models. Most of the studies work on agent-based simulation (ABS) models for the electricity markets and Bunn & Oliveira (2001) is one of the pioneers. Tellidou & Bakirtzis (2007) and Krause, Beck, Cherkaoui, Germond, Andersson & Ernst (2006) show that dynamic learning modeling is critical for the ABS models for tacit collusion, where the agents make decisions accordingly. Anderson & Cau (2011) explores the likelihood of tacit collusion under different market characterizations.

Shafie-khah, Moghaddam & Sheikh-El-Eslami (2013) propose a model using a heuristic dynamic game theory algorithm based on SFE and in the model the power market is simulated in a period. They consider the security constraints for the role of the ISO and the price uncertainty. For the learning model, the study assumes that an agent could observe some statistics related to previous actions of other agents. Furthermore, they propose a concept to distinguish between tacit and explicit collusion.

In Moiseeva, Hesamzadeh & Dimoulkas (2014), some elements of both game theory and agent-based simulations are combined and an ex-ante detection algorithm is introduced. It is assumed that the constraint sets of the agents are confidential and may vary in time. They use a distributed optimization concept and compare the outcome of tacit collusion with the outcome of Nash equilibrium over a given time horizon.

Harrington, Hobbs, Pang, Liu & Roch (2005) formulate an optimization problem to model tacit collusion, where the GenCos collectively maximize the Nash bargaining objective subject to a set of incentive compatibility constraints. Liu & Hobbs (2013) extend this study on a competitive pool-based electricity market operated by the ISO. They consider transmission congestion in the model in a repeated-game setting. The resulting model of each GenCo's profit maximization problem is a Mathematical Program with Equilibrium Constraints (MPEC) model and an Equilibrium Problem with Equilibrium Constraints (EPEC) model is obtained when all GenCos problems are combined. However, an equilibrium may not always exist and the EPEC models are hard to solve.

Aliabadi et al. (2016) introduce a mathematical model in the form of a multi-objective non-linear bi-level optimization problem, which is an alternative formulation to the EPEC model presented in Liu & Hobbs (2013). The introduced bi-level problem represents one iteration of the game and can be solved with linear programming (LP) under some assumptions. They also consider the transmission capacity constraints. Moreover, they present an algorithm to detect collusion when some sufficient conditions exist. However, the algorithm is computationally expensive since the computational complexity is $O(2^n)$, where n is the number of the GenCos.

In this study, in order to detect collusion, we generate search algorithms to characterize the solution space by using the multi-objective non-linear bi-level optimization problem introduced in Aliabadi et al. (2016).

3. PROBLEM DEFINITION

In the day-ahead operations of a deregulated electricity market, each GenCo offers a bid to the ISO for a certain period of the next-day. The collection of the submitted bids is defined as a state and denoted as $(b_1, b_2, \ldots, b_n)$, where b_i is the bid given by GenCo$_i$ and n is the number of the companies. For a given state, the ISO clears the market and determines the power assigned to each GenCo. This is how the market cleared repeatedly for all periods of a day over and over for each day. The market is affected by GenCos' strategic behaviors and as a result of the GenCos' strategic behaviors, collusion might exist. Aliabadi et al. (2016) characterize the existence of collusion by identifying the profitability of a collusive state over all Nash equilibria. In order to clear the market and then calculate the payoffs of the GenCos, the well-known Direct Current Optimal Power Flow (DC-OPF) problem can be solved.

The DC-OPF problem that also considers the transmission network capacity is formulated using a network. In the network representation, the nodes stand for the GenCos and the arcs between the nodes are the transmission lines. The GenCos may not produce power in each auction. Therefore, the set of all nodes is represented as I, and the set of nodes for which generates power is denoted as IG. Only the nodes in IG can submit a bid; power will not be allocated to the nodes which are not in IG.

In order to solve the DC-OPF problem, we assume that all the parameters are known and the generators can only bid from a given bid set. In the problem, P_i^{max} denotes the maximum production capacity for node $i \in IG$ (GenCo$_i$). D_i is the power demand at node $i \in I$. γ_{ij} represents the admittance of the line connecting node $i \in I$ to node $j \in I$ and F_{ij}^{max} the maximum flow allowed in the transmission line connecting node $i \in I$ to node $j \in I$. b_i is the bid given by node $i \in IG$. θ_i is a free variable for the voltage angle at node $i \in I$. The power allocated by node $i \in I$ is a non-negative decision variable, denoted as P_i. The DC-OPF problem is represented as follows:

$$
\begin{array}{llll}
(3.1a) & \underset{\{P_i,\theta_i\}}{\text{Minimize}} \sum_i b_i P_i & & \\
(3.1b) & \text{subject to} \quad P_i - D_i = \sum_{ij \in BR} \gamma_{ij}(\theta_i - \theta_j) & \forall i & [LMP_i] \\
(3.1c) & P_i \leq P_i^{max} & \forall i & [\phi_i] \\
(3.1d) & |\gamma_{ij}(\theta_i - \theta_j)| \leq F_{ij}^{max} & \forall ij \in BR & [\psi_{ij}^+, \psi_{ij}^-] \\
(3.1e) & -\pi \leq \theta_i \leq \pi & \forall i & \\
(3.1f) & P_i \geq 0 & \forall i &
\end{array}
$$

The DC-OPF problem is an LP. The objective function (3.1a) is to minimize the total cost of the produced electricity. In the flow balance constraint (3.1b), after the demand is satisfied extra injected power flows to transmission lines. The production capacity constraint (3.1c) limits the injected power by the maximum power of each GenCo. The flow capacity constraint (3.1d) restricts the flow of each line in the transmission grid, where BR is the set of all available distinct transmission lines. Constraint (3.1e) defines upper and lower bounds to θ_i for each node $i \in I$. Between (3.1b) and (3.1d), the associated dual variables to these constraints are given in the square brackets. It is known that $-\pi \leq \theta_i \leq \pi$ would be satisfied by the structure of the problem even without the constraint (3.1e). Therefore, no dual variable is assigned to the constraint (3.1e). Once the market is cleared through the solution of a DC-OPF problem for a given state, the payoff of GenCo$_i$ is calculated as $r_i = P_i(LMP_i - C_i)$, where C_i is the unit power production cost and the dual variable LMP_i of constraint (3.1b) in the DC-OPF problem represents the locational marginal price at node i.

Aliabadi et al. (2016) studies the detection of collusion in deregulated electricity markets. For this purpose, they first define how a collusive state is identified. In a collusive state, each GenCo profits more than it would profit from all Nash equilibria. In other words, $r_i > r_i^* \geq 0$, $\forall i$, where r_i is the payoff of GenCo$_i$ in a collusive state and r_i^* is the maximum payoff of GenCo$_i$ can attain from all Nash states and a state is called as a Nash state when a Nash equilibrium exists for given bids. In the following, we first discuss alternative ways of detecting collusion analytically.

3.1 Total Enumeration

As there is a finite number of states, for any repetition of the market clearance the DC-OPF problem can be solved for each state to attain the earnings of the GenCos and the payoffs can be compared with the corresponding r_i^*'s. This can be called as total enumeration. Aliabadi et al. (2016) proved that this method always detects all collusive states.

Once all the possible states are accounted for and the DC-OPF problem is solved for each state, Nash states can be identified by simple comparison among the payoffs of neighboring states. When all Nash states are identified, for all $i \in IG$, the maximum payoff of GenCo_i can be obtained from the Nash states, i.e. r_i^*, is determined. Then, each state, which is not a Nash state, is scanned to find out whether it is collusive or not. When each GenCo_i, $i \in IG$, of a state has a greater payoff than its corresponding r_i^*, then that state is collusive.

When the number of nodes increases, the computational time of generating all the states increases exponentially and the computational complexity of enumerating all possible states is $O(2^n)$. Hence, total enumeration is computationally expensive. Therefore, in the next section we discuss a mathematical formulation to detect a collusive state, as developed in Aliabadi et al. (2016). It is in the form of a bi-level programming problem.

3.2 Bi-level Problem

Using a mathematical modelling approach, Aliabadi et al. (2016) develop a multi-objective non-linear bi-level programming problem to detect collusion in a deregulated electricity market. The bi-level problem integrates two groups of decision makers: GenCos and the ISO. In the upper level, GenCos choose their bids with the aim of maximizing their payoffs. In the lower level, the follower level, the ISO makes decisions for the power allocation and the nodal price. The optimal solution of the bi-level problem indicates a collusive state, where it is also an optimal solution that minimizes the cost of produced electricity of the state.

The lower level of the bi-level programming problem is the DC-OPF problem (3.2e)

- (3.2j), as explained in Chapter 3. The upper level of the bi-level programming problem is between (3.2a) and (3.2d). As defined in Chapter 3, LMP_i generates the price of one unit of electricity at a node, C_i is the cost of production for a unit of power and P_i represents the power assigned to a generator node. $r_i = P_i(LMP_i - C_i)$ represents the payoff for a GenCo. b_i is a decision variable representing the bid value of a GenCo, which, can take a value only from a predetermined set of values, denoted as Bid_i. λ is an auxiliary variable used to approximate the original objective function $\underset{\{b_i\}}{\text{Maximize}}\ (r_1, r_2, \ldots, r_n)$ in Aliabadi et al. (2016) in order to maximize the payoffs for all GenCos. The bi-level programming model is represented as follows:

$$\underset{\{b_i,\lambda\}}{\text{Maximize}} \quad \lambda \tag{3.2a}$$

$$\text{subject to} \quad \lambda \leq P_i(LMP_i - C_i) \qquad \forall i \tag{3.2b}$$

$$P_i(LMP_i - C_i) \geq r_i^* \qquad \forall i \tag{3.2c}$$

$$b_i \in Bid_i \qquad \forall i \tag{3.2d}$$

$$\underset{\{P_i,\theta_i\}}{\text{Minimize}} \sum_i b_i P_i \tag{3.2e}$$

$$\text{subject to} \quad P_i - D_i = \sum_{ij \in BR} \gamma_{ij}(\theta_i - \theta_j) \qquad \forall i \qquad [LMP_i] \tag{3.2f}$$

$$P_i \leq P_i^{max} \qquad \forall i \qquad [\phi_i] \tag{3.2g}$$

$$|\gamma_{ij}(\theta_i - \theta_j)| \leq F_{ij}^{max} \qquad \forall ij \in BR \qquad [\psi_{ij}^+, \psi_{ij}^-] \tag{3.2h}$$

$$-\pi \leq \theta_i \leq \pi \qquad \forall i \tag{3.2i}$$

$$P_i \geq 0 \qquad \forall i \tag{3.2j}$$

From the ISO's perspective, the objective function (3.2e) is to minimizes the total cost of produced electricity. In order to approximate the original objective function, constraint (3.2b) is used to define λ as equal to the minimum of all payoffs. Constraint (3.2c) ensures that a collusive state is found, where each GenCo has a payoff greater than or equal to r_i^*.

3.2.1 Reformulation as a Single-Level Problem

In order to solve a bi-level programming problem with today's solvers, we have to reformulate it as a single level problem. One way to convert a bi-level problem into a single-level problem is to use the Strong Duality theorem defined in Vanderbei (2014). According to the theorem, if a primal problem has an optimal solution, then the objective values of the primal and the dual problems will be equal to each other. Since all the decision variables in the DC-OPF problem has an upper and a lower bound, the solution space can only be bounded or infeasible. Therefore, a solution state cannot be obtained from the bi-level programming problem if the DC-OPF problem has no feasible solution. Hence, while converting the bi-level problem to a single level problem, we add the dual constraints of the lower level problem, and a constraint to enforce the equality of the primal and dual objective values. For this conversion, the dual of the lower level (DC-OPF) problem can be written as follows:

(3.3a) $$\text{Maximize} \sum_i D_i LMP_i - \sum_i P_i^{max}\phi_i - \sum_{ij} F_{ij}^{max}(\psi_{ij}^+ + \psi_{ij}^-)$$

(3.3b) $$\text{subject to} \quad LMP_i - \phi_i \leq b_i \qquad \forall i$$

(3.3c) $$\sum_{ij\in BR}\gamma_{ij}(LMP_j - LMP_i) + \sum_{ij\in BR}\gamma_{ij}(\psi_{ij}^- - \psi_{ij}^+) + \sum_{ji\in BR}\gamma_{ji}(\psi_{ji}^+ - \psi_{ji}^-) = 0 \quad \forall i$$

(3.3d) $$\phi_i \geq 0 \qquad \forall i$$

(3.3e) $$\psi_{ij}^+, \psi_{ij}^- \geq 0 \qquad \forall ij \in BR$$

Based on the dual of the lower level problem, the bi-level problem is reformulated as a regular (single-level) problem as follows:

(3.4a) $$\underset{\{b_i,\lambda,P_i,\theta_i,LMP_i,\phi_i,\psi_{ij}^+,\psi_{ij}^-\}}{\text{Maximize}} \quad \lambda$$

(3.4b) $$\text{s.t.} \quad \lambda \leq P_i(LMP_i - C_i) \qquad \forall i$$

(3.4c) $$P_i(LMP_i - C_i) \geq r_i^* \qquad \forall i$$

(3.4d) $$b_i - LMP_i + \phi_i \geq 0 \qquad \forall i$$

(3.4e) $$\sum_{ij \in BR} \gamma_{ij}(LMP_j - LMP_i) + \sum_{ij \in BR} \gamma_{ij}(\psi_{ij}^- - \psi_{ij}^+) + \sum_{ji \in BR} \gamma_{ji}(\psi_{ji}^+ - \psi_{ji}^-) = 0 \quad \forall i$$

(3.4f) $$P_i - D_i = \sum_{ij \in BR} \gamma_{ij}(\theta_i - \theta_j) \qquad \forall i$$

(3.4g) $$P_i \leq P_i^{max} \qquad \forall i$$

(3.4h) $$\gamma_{ij}(\theta_i - \theta_j) \leq F_{ij}^{max} \qquad \forall ij \in BR$$

(3.4i) $$\gamma_{ij}(\theta_i - \theta_j) \geq -F_{ij}^{max} \qquad \forall ij \in BR$$

(3.4j) $$\sum_i b_i P_i = \sum_i D_i LMP_i - \sum_i P_i^{max} \phi_i - \sum_{ij} F_{ij}^{max}(\psi_{ij}^+ + \psi_{ij}^-)$$

(3.4k) $$-\pi \leq \theta_i \leq \pi \qquad \forall i$$

(3.4l) $$P_i, \phi_i \geq 0 \qquad \forall i$$

(3.4m) $$\psi_{ij}^+, \psi_{ij}^- \geq 0 \qquad \forall ij \in BR$$

(3.4n) $$b_i \in Bid_i \qquad \forall i$$

The objective function of the single level problem is the same with the objective function of the upper level of the bi-level problem (3.4a). All the constraints of both the upper level and lower level problems are the constraints of the single level problem as (3.4b)-(3.4c) and (3.4f)-(3.4g). As a result of the Strong Duality Theorem; the constraints of the dual problem (3.3b)-(3.3c) and the equality of the objective functions of the primal-dual problems of the lower level problem, are introduced as (3.4d)-(3.4e) and (3.4j), respectively. Constraints (3.4h) and (3.4i) correspond to linearization of (3.2h).

3.2.2 Solving the Reformulation

As mentioned before, r_i^* is the maximum payoff that GenCo$_i$ obtains from the Nash states. Since the Nash states are not known, in advance r_i^*'s cannot be calculated without using an algorithm similar to the total enumeration. Therefore, the constraint (3.4c) is removed from the model. However, a feasible solution of the problem may not necessarily be collusive anymore. As a matter of fact, a solution can be suspected to be collusive as the payoffs are still maximized for each GenCo. Yet, there is no guarantee. Therefore, we name the obtained state as a suspicious state since it is suspicious of being collusive.

The single-level formulation is an MINLP (with assumption b_is are integer). It is clear that working with an MILP would be much easier. The constraints (3.4b) and (3.4j), have nonlinear expressions P_iLMP_i and P_ib_i. A multiplication of two continuous decision variables can be linearized by an approximation.

According to the complementary slackness conditions of the DC-OPF problem,

$$P_i(b_i - LMP_i + \phi_i) = 0 \tag{3.5}$$

$$(P_i - P_i^{max})\phi_i = 0 \tag{3.6}$$

Therefore,

$$P_i > 0 \Rightarrow b_i = LMP_i - \phi_i \tag{3.7}$$

$$P_i \neq P_i^{max} \Rightarrow \phi = 0 \text{ and } LMP_i = b_i \tag{3.8}$$

Then; b_i is a lower bound for LMP_i or $LMP_i = b_i$. Therefore, the constraint (3.4b) is replaced with $\lambda \leq P_i(b_i - C_i)$. Now, only P_ib_i creates the non-linearity in the model.

We consider the linearization of the constraint by using similar linearization method as in Pozo, Sauma & Contreras (2013):

- The representation of the bid sets is changed: the kth element of a bid set is denoted as Bid_{ik}, where k is a number which is less then or equal to the number of bid values that GenCo$_i$ can take.
- B_{ik} is defined a binary variable to represent whether associated bid from the bid set is chosen. When GenCo$_i$ bids the kth element of the bid set Bid_{ik}, the value of B_{ik} is equal to 1.
- V_{ik} is defined as an auxiliary variable, which equals to P_iB_{ik}.

Accordingly, (3.4b) and (3.4j) are replaced by

$$\lambda \leq \sum_{k \in K} \mathrm{Bid}_{ik} V_{ik} - P_iC_i \qquad \forall i \tag{3.9a}$$

$$b_i = \sum_{k \in K} \mathrm{Bid}_{ik} B_{ik} \qquad \forall i \tag{3.9b}$$

$$\sum_{k \in K} B_{ik} = 1 \qquad \forall i \tag{3.9c}$$

$$V_{ik} \le P_i^{max} B_{ik} \qquad \forall i \text{ and } k \tag{3.9d}$$

$$V_{ik} \le P_i \qquad \forall i \text{ and } k \tag{3.9e}$$

$$V_{ik} \ge P_i - P_i^{max}[1 - B_{ik}] \qquad \forall i \text{ and } k \tag{3.9f}$$

$$\sum_i \sum_{k \in K} \mathrm{Bid}_{ik} V_{ik} = \sum_i D_i LMP_i - \sum_i P_i^{max} \phi_i - \sum_{ij} F_{ij}^{max}(\psi_{ij}^{+} + \psi_{ij}^{-}) \tag{3.9g}$$

$$V_{ik} \ge 0 \qquad \forall i \text{ and } k \tag{3.9h}$$

$$B_{ik} \in \{0, 1\} \qquad \forall i \text{ and } k \tag{3.9i}$$

Finally, we have a MILP problem to solve. Since r_i^*'s are not known, the original formulation is modified and the constraint that guarantees the formulation to find a collusive state is removed. A solution to the MILP problem can be suspected to be a collusive state. This state is only one of the states in the solution space. It is known that there can be multiple collusive states. Therefore, we call this problem SCSF to stand for "suspiciouslly collusive state finder", and also use SCSFP to refer to the problem formulation. In Chapter 4, with the aim of detecting all of the collusive states; we generate search algorithms, which solves the SCSF problem in each iteration to find another state.

4. ALGORITHMS

The original multi-objective problem formulation of the bi-level problem might have multiple optimal solutions each of which corresponds to a state in the feasible solution space. However, changing the objective function to a singleton might disallow this. One may overcome this obstruction by solving the problem iteratively while prohibiting the already found solutions in each iteration. The interactive scheme may continue until no feasible solution is attained. Furthermore, since the constraint (3.4c) is removed from the problem formulation assuming that r_i^*'s are unknown, a state obtained from the problem is not ensured to be collusive. However, the states are suspicious of being collusive. In this respect, we develop a search algorithm in order to designate suspicious states by iteratively solving the SCSF problem and approximating r_i^*'s.

4.1 Elementary Search

It is known that each payoff of a collusive state (r_i) is strictly greater than the corresponding payoffs of all Nash states, i.e. $r_i > r_i^* \geq 0,\ \forall i \in IG$. Therefore, it is guaranteed that in a collusive state, all GenCo$_i$'s, $i \in IG$, can have a non-zero payoff. In this respect, in each iteration of the Elementary Search algorithm, a state with no zero payoffs is found; it is declared as suspicious for being collusive. The iterations continue until a zero payoff is obtained for at least one of the GenCos. From one iteration to the next, we modify the mathematical formulation not to find the most recently found solution as well as all previous solutions.

The Elementary Search algorithm starts solving the SCSF problem. From the solution of the problem, a state, which is denoted as s, and λ are obtained. Instead of checking each payoff for having a non-zero value, basically $\underset{i}{min}\{r_i\}$, i.e. λ, is checked for a non-zero value. When $\lambda > 0$, to avoid obtaining the same suspicious state in the next iteration, a constraint is added to the problem formulation. This iteration is repeated until a solution with $\lambda = 0$ is found. At the end of each iteration, a new constraint is added to the formulation to avoid finding the most recently found solution. This algorithm is shown in Figure 4.1.

The new constraints added at the end of each iteration are called suspicious cuts. Since every bid has an associated binary variable, the new constraint should prohibit that the associated binary variables are equal to 1 at the same time, as

$$\sum_{i} B_{ik} \leq n-1 \qquad \forall\ b_i \in (b_1, b_2, \ldots, b_n),\ b_i = Bid_{ik}$$

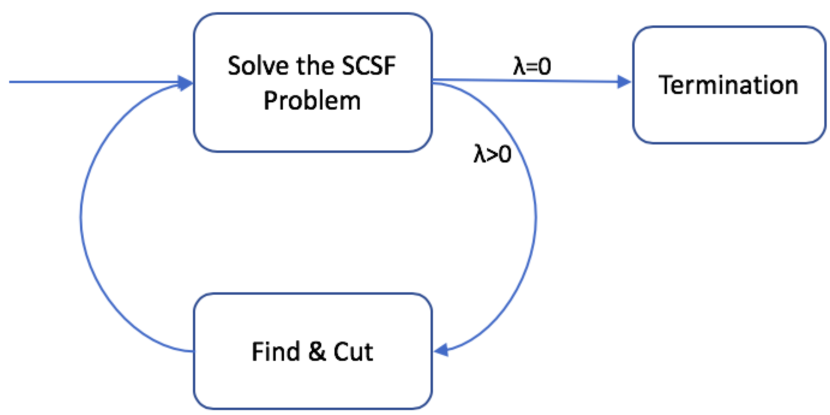
for a given state s, where n is the number of the nodes. The Find & Cut module, in Algorithm 1, summarizes this operation for a given state s. On line 2 in Algorithm 1, the state s is added to $SuspS$, where $SuspS$ is a set for the suspicious states and at the beginning of the algorithm it is an empty set. Then, on line 3 in Algorithm 1, the suspicious cut is added to the SCSF problem.

Algorithm 1 Find & Cut

1: *Input: SuspS, s, model*
2: $SuspS \leftarrow SuspS \cup \{s\}$
3: Add suspicious cut to the SCSF problem
4: *Output: SuspS, model*

The Elementary Search algorithm iteratively solves a modified SCSF problem until a zero payoff, i.e. $\lambda = 0$, is obtained or the problem becomes infeasible as demonstrated in the flow chart in Figure 4.1.

Figure 4.1 Elementary Search Algorithm



At the end of the Elementary Search algorithm, it is not exactly known which suspicious states are actually collusive since they are obtained without knowing r_i^*'s. However, as mentioned before, in a collusive state all GenCo's have a non-zero payoff, it is guaranteed to have the all collusive states in the set $SuspS$. The algorithm can be considered also as an efficient frontier search with additional constraints; the likelihood of a suspicious state being actually collusive can be low, but, the algorithm is not expected to be as costly as the total enumeration, from a computational point of view. Instead, the likelihood of a suspicious state being actually collusive can be increased with some information on Nash states, which we exploit further next.

4.2 A Search Algorithm with Partial Nash Information

When the Elementary Search algorithm terminates, non-collusive states can be eliminated from the suspicious states and all the collusive states can be designated, if r_i^*'s are known. Since computing r_i^*'s based on the total enumeration or similar approaches are expensive, one way to do such elimination is to approximate r_i^*'s with the aim of increasing the likelihood of a suspicious state being actually collusive. It is clear that $r_i^* \geq r_i^{\mathbb{N}}$, where $r_i^{\mathbb{N}}$ denotes a payoff of a Nash state. Therefore, information regarding only one Nash state is used in order to approximate r_i^*'s. When the payoffs of a Nash state is obtained, according to these payoffs, some states are excluded from being suspicious. Hence, the algorithm searches suspicious states by using partial Nash information and we name it as the SwpN to stand for "Search with partial Nash".

The Nash Finder, in Algorithm 2, outlines the steps of scanning states until a Nash state is found. PS denotes the number of possible states and at most PS times a state can be generated. First generated state is the combination of the initial elements of the bid sets, then each state is generated by changing one bid of the previously generated state. For a generated state, on line 4, the DC-OPF problem is solved and on line 5 the payoffs (r_i^s) are calculated. ν_s^i denotes a state, which is a neighbor to the state s based on the bid of GenCo$_i$. The neighborhood of a state includes all of the neighbor states based on the bid of GenCo$_i$ for all $i \in IG$. On line 8 in Algorithm 2, the DC-OPF problem is solved for each neighbor ν_s^i in the neighborhood; the payoffs ($r_i^{\nu_s^i}$) are computed for each $i \in IG$ on line 9 . When the DC-OPF problem is solved and $r_i^{\nu_s^i}$'s are calculated for each ν_s^i, $i \in IG$; on line 12, the payoffs of the state and the payoff of the neighbors are compared. For each

$i \in IG$; if r_i^s is greater than or equal to each $r_i^{\nu_s^i}$, that state is a Nash state. Then, state s is added to the set $Nash$. The payoff of the found Nash state is denoted as $r_i^{\mathbb{N}}$. If the state is not a Nash state, then the same process is repeated with another generated state until there is no possible state.

Algorithm 2 Nash Finder

1: *Input: Nash, PS*
2: **for** $k \in range(PS)$ **do**
3:    Generate a state s
4:    Solve the DC-OPF problem for s
5:    Compute r_i^s's
6:    **for** each $i \in IG$ **do**
7:       **for** each ν_s^i of s **do**
8:          Solve the DC-OPF problem for ν_s^i
9:          Compute $r_j^{\nu_s^i}$'s, $\forall j \in IG$
10:       **end for**
11:    **end for**
12:    **if** $r_i^s \geq r_i^{\nu_s^i}$, $\forall i, \nu_s^i$ **then**
13:       $Nash \leftarrow Nash \cup \{s\}$
14:       Go to line 16
15:    **end if**
16: **end for**
17: *Output: Nash*

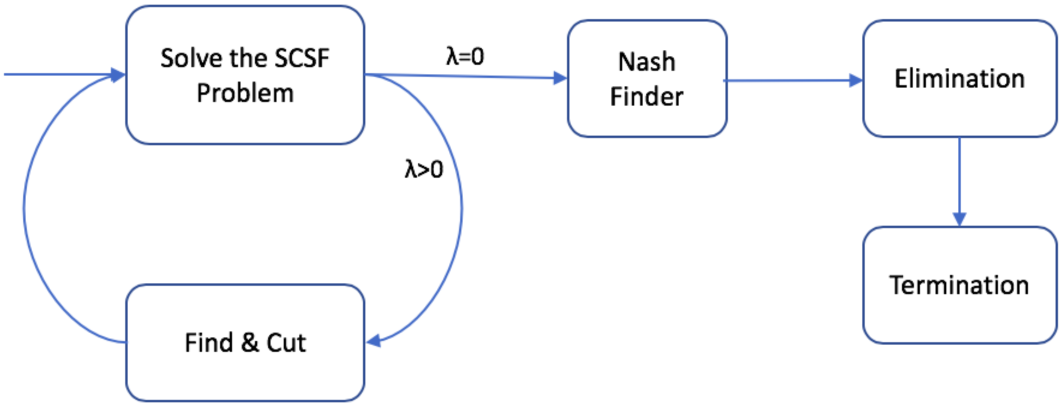
After a Nash state is found, the algorithm proceeds with eliminating some of the suspicious states. The Elimination module, in Algorithm 3, summarizes the steps of the elimination procedure with respect to $r_i^{\mathbb{N}}$. On line 3, for each state s in the set *SuspS*, each payoff of the state is compared to its corresponding payoff of the Nash state. If $r_i^s > r_i^{\mathbb{N}}$ for all $i \in IG$ then, state s is added to the set *Post-Elimination SuspS*. *Post-Elimination SuspS* includes states which are still suspicious of being collusive after the elimination. After all states $s \in SuspS$ are checked for the elimination criteria, the Elimination module terminates.

Algorithm 3 Elimination

1: *Input: SuspS, Post-Elimination SuspS,* $r_i^{\mathbb{N}}$*'s*
2: **for** each $s \in SuspS$ **do**
3:    **if** $r_i^s > r_i^{\mathbb{N}}$, $\forall i$ **then**
4:       *Post-Elimination SuspS* $\leftarrow$ *Post-Elimination SuspS* $\cup \{s\}$
5:    **end if**
6: **end for**
7: *Output: Post-Elimination SuspS*

The SwpN algorithm starts with the same procedure as the Elementary Search algorithm; right before the termination, it executes an elimination procedure which decreases the number of suspicious states without eliminating any actually collusive state. The flow of the algorithm is illustrated in Figure 4.2.

Figure 4.2 SwpN Algorithm



Since only guaranteed to be non-collusive states are excluded from being suspicious, at the end of the algorithm, the number of non-collusive states declared as suspicious is smaller, in comparison to the Elementary Search algorithm.

4.3 Variations of the Search Algorithm with Partial Nash Information

We propose some possible improvements and alternative versions of the SpwN algorithm. We discuss whether finding a Nash state before solving the SCSF problem can be beneficial from a computational point of view and how the algorithms can reach to the actual collusive states faster.

4.3.1 The Search with Partial Nash Information-Neighborhood Search Algorithm

It is known that solving the DC-OPF problem is faster than solving a SCSF problem. Hence, to detect suspicious states faster, the DC-OPF problem is solved for the neighbors of suspicious states which are obtained as solutions of the SCSFP. While scanning the neighbors of a suspicious state, the neighbor state is added to set *SuspS* and a suspicious cut is added to the SCSF problem, when minimum payoff of a neighbor is non-zero.

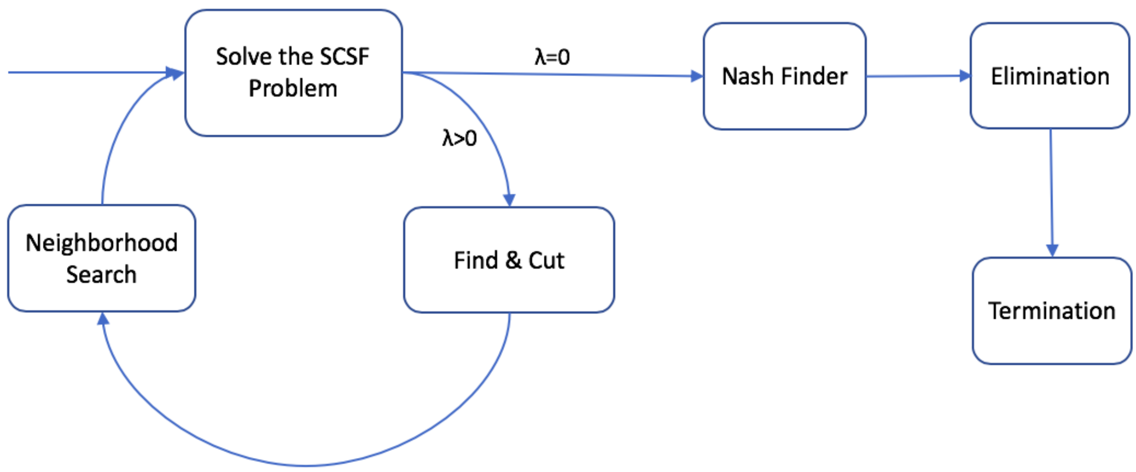
The Neighborhood Search module, in Algorithm 4, outlines the procedure for searching the neighborhood of a state. In Algorithm 4, for a neighbor of a given state s, the DC-OPF problem is solved and the payoffs are calculated on lines 4-5. On lines 6-8, the neighbor is checked for the conditions of being suspicious. When a neighbor is suspicious, the Find & Cut module (in Algorithm 1) is called. After scanning each neighbor ν_s^i, $\forall i \in IG$, of the state s, the module terminates.

Algorithm 4 Neighborhood Search

1: *Input: SuspS, s*
2: **for** each $i \in IG$ **do**
3:     **for** each ν_s^i of s **do**
4:         Solve the DC-OPF problem
5:         Compute $r_j^{\nu_s^i}$'s , $\forall j \in IG$
6:         **if** $\min_j \{r_j^{\nu_s^i}\} > 0$ **then**
7:             Find & Cut
8:         **end if**
9:     **end for**
10: **end for**
11: *Output: SuspS*

After the neighborhood of a suspicious state is scanned, a modified SCSF problem is solved until $\lambda = 0$ or it becomes infeasible. Later, as in the SwpN algorithm, a Nash state is found by calling the Nash Finder, and some non-collusive states are eliminated in the Elimination module. The flow chart of the new version algorithm is shown in Figure 4.3. We call this version SwpN-NS, where NS stands for "Neighborhood Search".

Figure 4.3 SwpN-NS Algorithm



Since the SwpN-NS algorithm searches the neighborhood of the suspicious states, it is expected to be faster than the SwpN algorithm.

4.3.2 The Preprocessing-Search with Partial Nash Information Algorithm

As stated before, the DC-OPF problem is solved to find a Nash state, and solving the DC-OPF problem is faster than solving the SCSF problem. Once the payoffs of a state are calculated from the result of a DC-OPF problem, then the state can be checked for being suspicious. When a state is suspicious, the SCSF problem can be modified to avoid obtaining the same suspicious state unnecessarily. In these respects, a new version of the SwpN algorithm, where a Nash state is found before solving the SCSF problem is developed.

Aliabadi et al. (2016), prove that no Nash equilibrium exists in the neighborhood of a collusive state. This implies that there cannot be any collusive state in the neighborhood of a Nash state. Therefore, constraints for each neighbor of the Nash state are added to the SCSF problem to prohibit obtaining these states. These operations are outlined in the Nash Finder with Find & Cut module, in Algorithm 5.

Initially, the Nash Finder with Find & Cut module proceeds exactly as the Nash Finder module. The only difference is that the state is checked for being suspicious on lines 6-8, whenever the DC-OPF problem is solved. When a state is suspicious, it is added to *SuspS* and a suspicious cut is added to the SCSF problem, on lines

13-15. When a Nash state is found, a suspicious cut is added to the SCSFP for each neighbor of the Nash state on lines between 23 and 29.

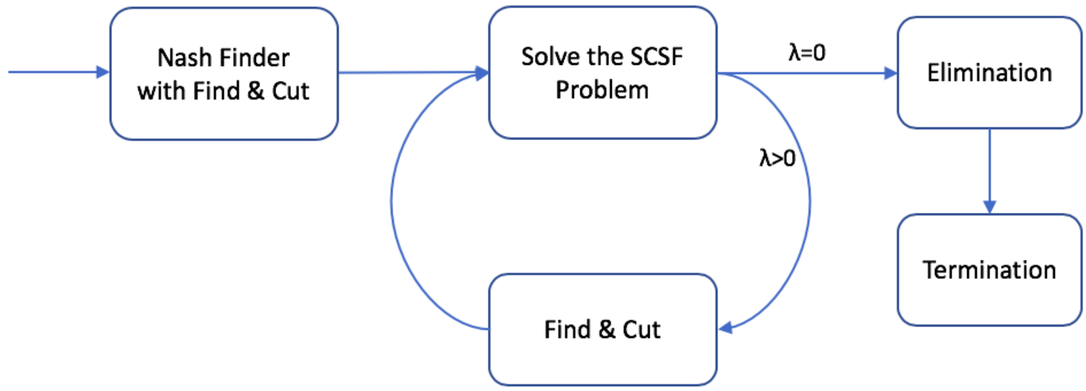
We name this algorithm a pp-SwpN since the Nash Finder is used at a preprocessing, hence pp, stage.

Algorithm 5 Nash Finder with Find & Cut

1: *Input: Nash, SuspS, PS*
2: **for** $k \in range(PS)$ **do**
3: Generate a state s
4: Solve the DC-OPF problem for s
5: Compute r_j^s's, $\forall j \in IG$
6: **if** $\min_j \{r_j^s\} > 0$ **then**
7: Find & Cut
8: **end if**
9: **for** each $i \in IG$ **do**
10: **for** each ν_s^i of s **do**
11: Solve the DC-OPF problem for ν_s^i
12: Compute $r_j^{\nu_s^i}$'s , $\forall j \in IG$
13: **if** $\min_j \{r_j^{\nu_s^i}\} > 0$ **then**
14: Find & Cut
15: **end if**
16: **end for**
17: **end for**
18: **if** $r_i^s \geq r_i^{\nu_s^i}$, $\forall i, \nu_s^i$ **then**
19: $Nash \leftarrow Nash \cup \{s\}$
20: Go to line 23
21: **end if**
22: **end for**
23: **for** $s \in Nash$ **do**
24: **for** each $i \in IG$ **do**
25: **for** each ν_s^i of s **do**
26: Add suspicious cut to the SCSF problem
27: **end for**
28: **end for**
29: **end for**
30: *Output: Nash, SuspS*

In the new algorithm, a modified SCSF problem is solved iteratively after a Nash state is found. When $\lambda = 0$ or the problem becomes infeasible, the algorithm proceeds with elimination. The flow chart of the new algorithm is demonstrated in Figure 4.4.

Figure 4.4 pp-SwpN Algorithm



The pp-SwpN algorithm starts solving the SCSF problem with more constraints, which narrows down the solution space. It is expected that a SCSF problem will be solved iteratively with less number of iterations in comparison to the pp-SwpN algorithm. This implies that the new algorithm might be faster.

4.3.3 The Preprocessing-Search with Partial Nash Information and Neighborhood Search Algorithm

We intend to improve the pp-SwpN algorithm by integrating ideas used in the SwpN-NS algorithm. Accordingly, we name this algorithm as pp-SwpN-NS.

As mentioned in Section 4.3.1, searching the neighborhood of a suspicious state might be an opportunity to speed up the SwpN algorithm. Therefore, in the pp-SwpN-NS algorithm, the neighborhood is searched for suspicious state obtained from the solution of a SCSF problem.

The Elimination module excludes some suspicious states from being suspicious according to the elimination criteria, which depends on the payoffs of a Nash state. When a Nash state is found at the initial step of a algorithm, the parameters of the elimination criteria is known while solving a SCSF problem. Therefore, instead of eliminating those suspicious states at the end, a state obtained from a SCSF problem is added to *SuspS* when it satisfies the elimination criteria. In this respect, the Find & Cut with Elimination, in Algorithm 6, is developed as an improved version of the Find & Cut module. On line 3, a suspicious cut is added to the SCSF problem when the minimum payoff of a state is non-zero. On lines 4-6, the state is added to *Post-Elimination SuspS*, when each payoff of the state is greater than or equal to the corresponding payoff of the Nash state.

Algorithm 6 Find & Cut with Elimination

Input: Post-Elimination SuspS, Nash, s
if $\min_i \{r_i^s\} > 0$ or $\lambda > 0$ **then**
  Add suspicious cut to the SCSF problem
  if $r_i^s > r_i^{\mathbb{N}}$, $\forall i$ **then**
    Post-Elimination SuspS $\leftarrow$ *Post-Elimination SuspS* $\cup \{s\}$
  end if
end if
Output: Post-Elimination SuspS

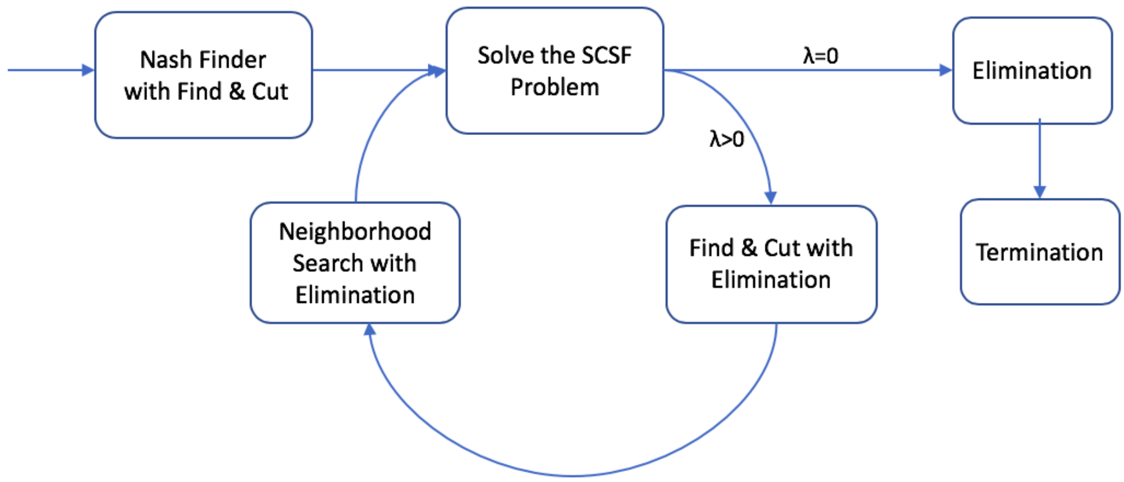
The Neighborhood Search module is also improved; the new module is summarized in the Neighborhood Search with Elimination module, in Algorithm 7. In the new version, on line 7, the neighbor states are added to *Post-Elimination SuspS* instead of *SuspS*, when each payoff of a neighbor is greater than the corresponding payoff of the Nash state.

Algorithm 7 Neighborhood Search with Elimination

1: *Input: SuspS, s*
2: **for** each $i \in IG$ **do**
3:   **for** each ν_s^i of s **do**
4:     Solve the DC-OPF problem
5:     Compute $r_j^{\nu_s^i}$'s , $\forall j \in IG$
6:     **if** $\min_j \{r_j^{\nu_s^i}\} > 0$ **then**
7:       Find & Cut with Elimination
8:     **end if**
9:   **end for**
10: **end for**
11: *Output: Post-Elimination SuspS*

As in the pp-SwpN algorithm, the pp-SwpN-NS algorithm proceeds with the Elimination module after the iterations with SCSFP are completed. Figure 4.5 shows the flow chart of the pp-SwpN-NS algorithm.

Figure 4.5 pp-SwpN-NS Algorithm



In the pp-SwpN-NS algorithm, we improve the pp-SwpN algorithm by using the knowledge of $r_i^{\mathbb{N}}$'s and searching the neighborhood of the suspicious states. In Section 4.3.4, we discuss how to improve the pp-SwpN-NS algorithm.

4.3.4 The Preprocessing-Search with Partial Nash Information and Neighborhood of Neighbors Search Algorithm

As already indicated, searching a neighborhood of a state is an advantage from the computational point of view. Furthermore, since a state and its neighbors are expected to receive similar payoffs, the likelihood of being suspicious for a neighbor of a suspicious state is high. In a variant of the pp-SwpN-NS algorithm, neighborhoods of suspicious neighbor states are scanned while finding a Nash state.

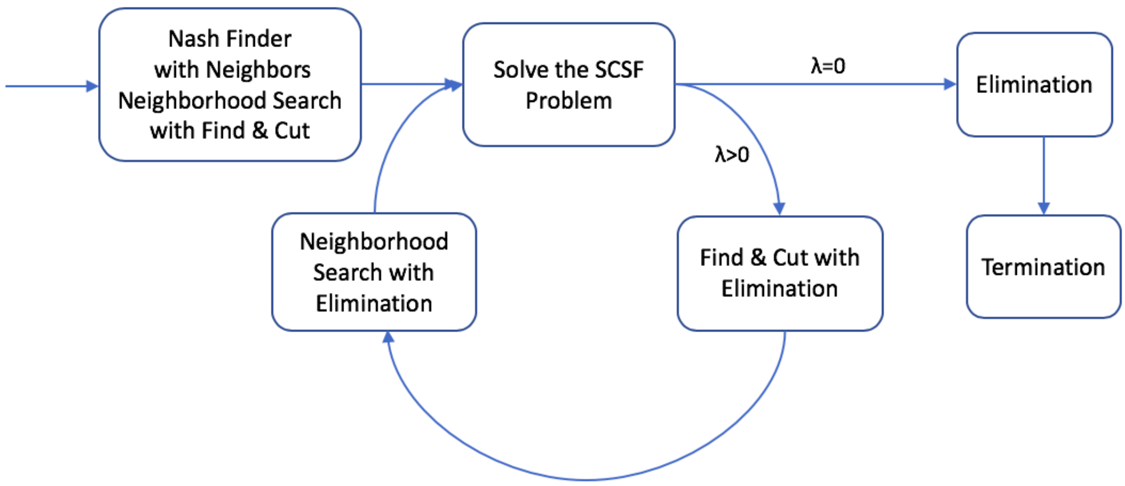
In the new variant, the only difference is due to solving the DC-OPF problem for each neighbor of a suspicious neighbor state and checking the neighbors of the neighbor states for the conditions of being suspicious. The exploratory Nash Finder with Find & Cut module, in Algorithm 8, summarizes the new search process of finding a Nash state. On lines 4-5, the DC-OPF problem is solved for a state and the payoffs are computed. On lines 6-8, the state is checked for being suspicious. If it is suspicious, the Find & Cut module is called. Then, for each neighbor ν_s^i of s based on the bid of GenCo$_i$, $i \in IG$, the DC-OPF problem is solved and the neighbors are checked for being suspicious on lines 11-13. When ν_s^i is suspicious, the Find & Cut module is called on line 14. For each neighbor $\nu\nu_s^{ij}$, the DC-OPF problem is solved and the payoffs are computed (lines 17-18), where $\nu\nu_s^{ij}$ is a neighbor of the neighbor state (ν_s^i) based on the bid of GenCo_j, $j \in IG$. According to the minimum payoff of $\nu\nu_s^{ij}$, the neighbor of a neighbor of the state is checked for being suspicious. If it is suspicious, the Find & Cut module is called (lines 19-21). On lines 23-26, the neighbor (ν_s^i) is checked to find out if it is a Nash state. When the neighbor is a Nash state, then it is added to $Nash$. After each neighbor of a state is checked for being Nash, the state is checked for being a Nash state on lines 30-35. When the state is not a Nash state, then the same procedure is done for another state. On lines 35-42, a suspicious cut for each neighbor of a Nash state is added to the SCSF problem.

Algorithm 8 Nash Finder with Neighbors Neighborhood Search with Find & Cut

1: *Input: Nash, SuspS, PS*
2: **for** $l \in range(PS)$ **do**
3: Generate a state s
4: Solve the DC-OPF problem
5: Compute r_j^s's , $\forall j \in IG$
6: **if** $\min_j\{r_j^s\} > 0$ **then**
7: Find & Cut
8: **end if**
9: **for** each $i \in IG$ **do**
10: **for** each ν_s^i of s **do**
11: Solve the DC-OPF problem for ν_s^i
12: Compute $r_j^{\nu_s^i}$'s, $\forall j \in IG$
13: **if** $\min_j\{r_j^{\nu_s^i}\} > 0$ **then**
14: Find & Cut
15: **for** each $j \in IG$ **do**
16: **for** each $\nu\nu_s^{ij}$ of ν_s^i **do**
17: Solve the DC-OPF problem for $\nu\nu_s^{ij}$
18: Compute $r_k^{\nu\nu_s^{ij}}$, $\forall k \in IG$
19: **if** $\min_k\{r_k^{\nu\nu_s^{ij}}\} > 0$ **then**
20: Find & Cut
21: **end if**
22: **end for**
23: **if** $r_j^{\nu_s^i} \geq r_j^{\nu\nu_s^{ij}}$, $\forall j, \nu\nu_s^{ij}$ **then**
24: $Nash \leftarrow Nash \cup \{\nu_s^i\}$
25: Go to line 35
26: **end if**
27: **end for**
28: **end if**
29: **end for**
30: **if** $r_i^s \geq r_i^{\nu_s^i}$, $\forall i, \nu_s^i$ **then**
31: $Nash \leftarrow Nash \cup \{s\}$
32: Go to line 35
33: **end if**
34: **end for**
35: **end for**
36: **for** $s \in Nash$ **do**
37: **for** each $i \in IG$ **do**
38: **for** each ν_s^i of s **do**
39: Add suspicious cut to the SCSF problem
40: **end for**
41: **end for**
42: **end for**
43: *Output: Nash, SuspS*

In the new version algorithm, a Nash state is found and all the neighbors of the Nash state are prohibited from being suspicious by the Nash Finder with Neighbors Neighborhood Search with Find & Cut module. Then, the algorithm iteratively solves the SCSF problem as in the pp-SwpN-NS algorithm. When λ is a positive value, from one iteration to another, the Find & Cut with Elimination and Neighborhood Search with Elimination modules are applied. When λ is zero, the Elimination module is executed then, the algorithm terminates. We call this version pp-SwpN-NoNS where NoNS stands for "Neighborhood of Neighbors Search". The flow of the pp-SwpN-NoNS algorithm is shown in Figure 4.6.

Figure 4.6 pp-SwpN-NoNS Algorithm



The pp-SwpN-NoNS algorithm, aims to reduce the computational time of the pp-SwpN-NS algorithm. It scans the neighborhoods of the neighbors during the process of finding a Nash state. In comparison to the pp-SwpN-NS algorithm, it might obtain more suspicious states before solving the SCSF problem and add more constraints to the formulation. In this respect, it is expected that the pp-SwpN-NoNS algorithm is cheaper from the computational point of view.

5. COMPUTATIONAL RESULTS

In this chapter, we examine the results of the total enumeration and the search algorithms proposed in Chapter 4. The performance measures of the algorithms are the computational time, the ratio of found collusive and the ratio of collusive coverage since the aim of this work is to improve the solution space characterization and detect the collusive states faster. In order to evaluate the accuracy of the collusive state detection, the actual collusive states are found by the total enumeration method. The ratio of found collusive is calculated as the actual number of collusive states divided by the number of suspicious states. The ratio of collusive coverage is equal to the number of found collusive states divided by the actual number of collusive states.

In order to compare the performance of the algorithms, three problem sizes of instances are created: small, medium, and medium-plus cases. In total 26 instances are created; 10 of them are small, 10 of them are medium, and 6 of them are medium-plus. The small cases have 5 nodes and 3 of them are the power generation nodes, the medium and the medium-plus cases have 7 nodes and 4 of them are the power generation nodes, however, the bid sets of the medium-plus cases include more bidding options than for the medium cases. For the small cases $IG = \{1, 2, 5\}$, and for the medium and medium-plus cases $IG = \{1, 2, 5, 6\}$. In Tables 5.1, 5.2 and 5.3, the bid sets are provided for the small, medium and medium-plus cases, respectively. Other parameters such as C_i, D_i, P_i^{max}, and F_{ij}^{max} of the cases are provided in Appendix A.

Table 5.1 The Bid Sets of GenCos for the Small Cases

Cases	Bid Sets
GenCo_1	{22,27,32,37,42,47,52}
GenCo_2	{21,26,31,36,41,46,51}
GenCo_5	{30,35,40,45,50}

Table 5.2 The Bid Sets of GenCos for the Medium cases

Cases	Bid Sets
$GenCo_1$	{21,26,31,36,41,46,51}
$GenCo_2$	{22,27,32,37,42,47,52}
$GenCo_5$	{33,38,43,48,53}
$GenCo_6$	{14,19,24,29,34,39,44,49,54}

Table 5.3 The Bid Sets of GenCos for the Medium-Plus Cases

Cases	Bid Sets
$GenCo_1$	{21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51}
$GenCo_2$	{22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52}
$GenCo_5$	{33,35,37,39,41,43,45,47,49,51,53}
$GenCo_6$	{14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44}

It is known that there might be alternative solutions, therefore computed profits from the solution of the DC-OPF problem and the SCSF problem might be different. In this study, we work on cases that have a unique optimal solution from the DC-OPF problem.

We conduct our experiments on an Intel Xeon processor with 2.40 GHz speed and 12 GB RAM, with 64-bit Windows 7 operating system. The total enumeration and the algorithms are coded with Python 3.7, and Gurobi 9.0.2 is used as a solver. The experiments are repeated for 3 times and the average computational time is reported as the computational time. We observe that the algorithms perform faster when we set the Heuristics parameter of Gurobi as zero instead of the default value. Therefore, in this study, the value of the Heuristics parameter is zero while solving MILP and MINLP problems.

5.1 Total Enumeration Results

As described in Chapter 3.1, all collusive states can be detected by enumerating all states and solving the DC-OPF problem. When all the states are enumerated; there are 245 states for the small cases, 2205 states for the medium cases and 7986 states for the medium-plus cases. Tables 5.4, 5.5 and 5.6 show the number of Nash states, the number of collusive states and computational time (in seconds) for small, medium and medium-plus cases, respectively. The average computational time of the small cases is 11.14 seconds, 96.95 seconds for the medium cases, and 1445.82 seconds for the medium-plus cases.

Table 5.4 Total Enumeration Results of Small Cases

Cases	Number of Nash States	Number of Collusive States	Computational Time
Small 1	2	5	13.29
Small 2	2	5	11.1
Small 3	6	5	11.18
Small 4	1	6	13.52
Small 5	2	7	10.29
Small 6	3	9	9.98
Small 7	3	6	9.79
Small 8	2	5	9.55
Small 9	4	3	11.69
Small 10	2	13	10.96

Table 5.5 Total Enumeration Results of Medium Cases

Cases	Number of Nash States	Number of Collusive States	Computational Time
Medium 1	10	28	94.09
Medium 2	12	81	80.85
Medium 3	4	4	107.5
Medium 4	2	52	108.46
Medium 5	3	29	123.49
Medium 6	1	9	113.45
Medium 7	5	16	106.39
Medium 8	10	38	82.02
Medium 9	5	19	76.83
Medium 10	40	54	76.39

Table 5.6 Total Enumeration Results of Medium-Plus Cases

Cases	Number of Nash States	Number of Collusive States	Computational Time
Medium-plus 1	11	1316	1441.7
Medium-plus 2	4	438	1461.1
Medium-plus 3	12	715	1439.34
Medium-plus 4	11	844	1444.87
Medium-plus 5	5	903	1449.41
Medium-plus 6	20	561	1438.5

In the following sections, the identified collusive states are called as the actual collusive states. Moreover, the performance of the proposed algorithms in Chapter 4 will be evaluated according to the actual collusive states and the computational time of the total enumeration.

5.2 The Elementary Search Algorithm Results

In this section, in order to demonstrate that working with an MILP is much faster than an MINLP, the Elementary Search algorithm is executed for both of the formulations in Chapter 3.2.1 and in Chapter 3.2.2. In Tables 5.7, 5.9 and 5.11, the performance of the Elementary Search algorithm applied with the MILP problem is shown for the small, medium and medium-plus cases, respectively. The performance of the Elementary Search algorithm applied with the MINLP problem for the small and medium cases are presented in the tables 5.8 and 5.10. For medium-plus cases the performances Elementary Search algorithm applied with the MINLP problem are not presented because it costs more than 6 hours for each case, which is extremely slower than the performance of the Elementary Search algorithm applied with the MILP problem.

Table 5.7 Results of the Elementary Search Algorithm Applied with the MILP Problem for Small Cases

Cases	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	66	0.076	1	12.16
Small 2	87	0.057	1	12.88
Small 3	14	0.357	1	1.67
Small 4	40	0.150	1	6.89
Small 5	80	0.088	1	12.05
Small 6	68	0.132	1	11.77
Small 7	99	0.061	1	21.33
Small 8	66	0.076	1	10.37
Small 9	56	0.054	1	6.28
Small 10	87	0.149	1	17.17

Table 5.8 Results of the Elementary Search Algorithm Applied with the MINLP Problem for Small Cases

Cases	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	66	0.076	1	39.70
Small 2	87	0.057	1	54.54
Small 3	14	0.357	1	12.10
Small 4	52	0.115	1	39.06
Small 5	85	0.082	1	51.45
Small 6	68	0.132	1	34.21
Small 7	99	0.061	1	52.69
Small 8	69	0.072	1	36.86
Small 9	58	0.052	1	29.93
Small 10	87	0.149	1	44.73

Table 5.9 Results of the Elementary Search Algorithm Applied with the MILP Problem for Medium Cases

Cases	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	38	0.737	1	12.78
Medium 2	398	0.204	1	148.29
Medium 3	570	0.007	1	394.21
Medium 4	927	0.056	1	652.4
Medium 5	271	0.107	1	80.33
Medium 6	398	0.023	1	146.09
Medium 7	38	0.421	1	16.09
Medium 8	38	1	1	17.19
Medium 9	28	0.679	1	19.98
Medium 10	89	0.607	1	46.02

Table 5.10 Results of the Elementary Search Algorithm Applied with the MINLP Problem for Medium Cases

Cases	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	38	0.737	1	50.65
Medium 2	398	0.204	1	445.23
Medium 3	599	0.007	1	655.7
Medium 4	952	0.055	1	1329.74
Medium 5	271	0.107	1	320.84
Medium 6	398	0.023	1	272.96
Medium 7	38	0.421	1	54.5
Medium 8	38	1	1	41.81
Medium 9	28	0.679	1	41.26
Medium 10	89	0.607	1	68.21

Table 5.11 Results of the Elementary Search Algorithm Applied with the MILP Problem for Medium-Plus Cases

Cases	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	1840	0.715	1	5170.81
Medium-plus 2	3359	0.130	1	8764.97
Medium-plus 3	1208	0.592	1	1961.12
Medium-plus 4	1633	0.517	1	4477.55
Medium-plus 5	1633	0.553	1	5315.3
Medium-plus 6	1416	0.396	1	2863.37

There is a significant computational time difference when the MILP problem is used instead of the MINLP problem in the Elementary Search algorithm. Therefore, we compare the performance of the total enumeration and the Elementary Search algorithm applied with the MILP problem. Also, in the following sections, we continue implementing the proposed algorithms by using the MILP problem, which is denoted as SCSFP.

Since the Elementary Search algorithm guarantees to find all of the collusive states, the ratio of collusive coverage is equal to 1 for each of the cases. For some of the cases (especially for the medium-plus cases), the computational times are higher than the computational time of the total enumeration. Moreover, for a great majority, the ratio of found collusive is low. Therefore, one will prefer the total enumeration in order to ensure all of the find states are guaranteed to be the collusive states. Note that for the Elementary Search algorithm, the number of suspicious states is equal to how many times the SCSFP problem is iteratively solved.

5.3 The SwpN Algorithm Results

In this section, the SwpN algorithm is executed for all of the cases. The performance measures for all of the cases are shown in Tables 5.12, 5.13, 5.14 for small, medium and medium-plus cases, respectively.

Table 5.12 The SwpN Algorithm Results for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	66	5	1	1	25.49
Small 2	87	22	0.227	1	22.43
Small 3	14	12	0.417	1	4.94
Small 4	40	6	1	1	17.82
Small 5	80	7	1	1	20.60
Small 6	68	14	0.643	1	20.75
Small 7	99	6	1	1	30.02
Small 8	66	5	1	1	20.68
Small 9	56	5	0.6	1	12.78
Small 10	87	22	0.591	1	23.06

Table 5.13 The SwpN Algorithm Results for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	38	28	1	1	14.39
Medium 2	398	81	1	1	149.74
Medium 3	570	4	1	1	301.83
Medium 4	927	52	1	1	592.76
Medium 5	271	29	1	1	115.75
Medium 6	398	9	1	1	208.98
Medium 7	38	16	1	1	17.53
Medium 8	38	38	1	1	15.27
Medium 9	28	19	1	1	19.25
Medium 10	89	54	1	1	37.04

Table 5.14 The SwpN Algorithm Results for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	1840	1316	1	1	4847.44
Medium-plus 2	3359	895	0.489	1	10259.64
Medium-plus 3	1208	781	0.915	1	2434.96
Medium-plus 4	1633	951	0.887	1	4745.43
Medium-plus 5	1633	1018	0.887	1	5965.63
Medium-plus 6	1416	599	0.937	1	3311.66

As mentioned in Chapter 4, in the SwpN algorithm there is an addition to the Elementary Search algorithm, which is the elimination part. Therefore, for the most of the cases, the computational time of the SwpN algorithm is higher. But the ratios of found collusive are improved and most of the found suspicious states are the actual collusive states (the ratios of found collusive are equal or close to 1). In Tables 5.12, 5.13 and 5.14, the number of solved SCSFP also represents the number of suspicious states before the elimination and it is equal to the number of suspicious states in the Elementary Search algorithm.

Since it is guaranteed that the Elimination module cannot eliminate any actual collusive state, the ratio of collusive coverage for each state is equal to 1 as in the Elementary Search algorithm.

5.4 The Results for the Variations of the SwpN Algorithm

In this section, we discuss the results of the alternative versions of the SwpN algorithm and find out if the algorithms reach to actual collusive states faster than the total enumeration.

5.4.1 The SwpN-NS Algorithm Results

Tables 5.15, 5.16 and 5.17 present the performances of the SwpN-NS algorithm. As noted before, in the SwpN-NS algorithm, the aim is to reduce the number of times an SCSF problem is solved in comparison to the SwpN algorithm by solving the DC-OPF problem for neighboring states. As a result, the number of times an SCSFP solved is decreased. Therefore, the computational time is reduced for each case.

Since the elimination criteria is not changed, the ratio of found collusive and the ratio of collusive coverage remain the same as in the SwpN algorithm.

Table 5.15 The SwpN-NS Algorithm Results for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	11	5	1	1	9.55
Small 2	18	22	0.227	1	14.29
Small 3	4	12	0.417	1	4.57
Small 4	11	6	1	1	11.10
Small 5	13	7	1	1	10.96
Small 6	11	14	0.643	1	8.83
Small 7	18	6	1	1	10.24
Small 8	13	5	1	1	8.98
Small 9	13	5	0.6	1	9.42
Small 10	17	22	0.591	1	9.65

Table 5.16 The SwpN-NS Algorithm Results for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	6	28	1	1	11.48
Medium 2	54	81	1	1	56.59
Medium 3	81	4	1	1	97.08
Medium 4	123	52	1	1	150.89
Medium 5	45	29	1	1	53.86
Medium 6	56	9	1	1	81.27
Medium 7	6	16	1	1	8.89
Medium 8	6	38	1	1	9.05
Medium 9	6	19	1	1	10.22
Medium 10	19	54	1	1	24.58

Table 5.17 The SwpN-NS Algorithm Results for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	131	1316	1	1	561.08
Medium-plus 2	1551	883	0.496	1	6031.35
Medium-plus 3	197	783	0.913	1	795.80
Medium-plus 4	427	951	0.887	1	2149.65
Medium-plus 5	439	1016	0.889	1	2089.94
Medium-plus 6	124	564	0.995	1	916.52

5.4.2 The pp-SwpN Algorithm Results

In Chapter 4, in the pp-SwpN algorithm, a Nash state is found before iteratively solving an SCSF problem. The performance of the pp-SwpN algorithm for each small, medium and medium-plus cases are presented in Tables 5.18, 5.19 and 5.20.

Table 5.18 The pp-SwpN Algorithm Results for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	53	5	1	1	23.33
Small 2	65	22	0.227	1	20.21
Small 3	13	12	0.417	1	4.51
Small 4	1	6	1	1	10.43
Small 5	63	7	1	1	16.46
Small 6	53	14	0.643	1	18.14
Small 7	66	6	1	1	21.02
Small 8	47	5	1	1	15.95
Small 9	43	5	0.6	1	10.97
Small 10	65	22	0.591	1	18.01

Table 5.19 The pp-SwpN Algorithm Results for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	38	28	1	1	18.46
Medium 2	362	81	1	1	149.49
Medium 3	538	4	1	1	287.64
Medium 4	756	52	1	1	498.04
Medium 5	253	29	1	1	106.53
Medium 6	285	9	1	1	169.87
Medium 7	38	16	1	1	17.15
Medium 8	38	38	1	1	15.65
Medium 9	28	19	1	1	17.47
Medium 10	89	54	1	1	38.79

Table 5.20 The pp-SwpN Algorithm Results for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	1840	1316	1	1	5052.09
Medium-plus 2	2757	902	0.486	1	8904.93
Medium-plus 3	1208	782	0.914	1	2772.90
Medium-plus 4	1633	951	0.887	1	4861.44
Medium-plus 5	1633	1018	0.887	1	5369.48
Medium-plus 6	1416	599	0.937	1	3306.68

The ratio of found collusive and the ratio of collusive coverage remain the same as in the SwpN algorithm and the SwpN-NS algorithm. But the computational time increases for most of the cases. Therefore , the SwpN-NS algorithm definitely performs better than the pp-SwpN algorithm. Especially for a medium-plus case, one will prefer the total enumeration instead of the pp-SwpN algorithm.

5.4.3 The pp-SwpN-NS Algorithm Results

As proposed before, in the pp-SwpN-NS algorithm, initially a Nash state is found as in the pp-SwpN algorithm and the neighborhood of a suspicious state is searched, while solving an SCSF problem iteratively. In Tables 5.21, 5.22 and 5.23, the results for the pp-SwpN-NS algorithm are presented.

Table 5.21 The pp-SwpN-NS Algorithm Results for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	12	5	1	1	12.60
Small 2	13	22	0.227	1	8.83
Small 3	4	12	0.417	1	4.47
Small 4	1	6	1	1	9.66
Small 5	12	7	1	1	9.97
Small 6	10	14	0.643	1	9.00
Small 7	15	6	1	1	11.05
Small 8	12	5	1	1	12.13
Small 9	9	5	0.6	1	6.43
Small 10	12	22	0.591	1	9.78

Table 5.22 The pp-SwpN-NS Algorithm Results for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	6	28	1	1	8.71
Medium 2	56	81	1	1	59.44
Medium 3	84	4	1	1	89.35
Medium 4	106	52	1	1	127.62
Medium 5	43	29	1	1	53.41
Medium 6	51	9	1	1	82.26
Medium 7	6	16	1	1	8.61
Medium 8	6	38	1	1	7.61
Medium 9	6	19	1	1	10.64
Medium 10	19	54	1	1	22.36

Table 5.23 The pp-SwpN-NS Algorithm Results for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	131	1316	1	1	584.12
Medium-plus 2	1552	900	0.487	1	6102.11
Medium-plus 3	199	783	0.913	1	955.08
Medium-plus 4	428	951	0.887	1	1966.24
Medium-plus 5	445	1017	0.888	1	1926.68
Medium-plus 6	124	564	0.995	1	628.32

The ratio of found collusive and the ratio of collusive coverage are the same as in the other variations of the SwpN algorithm. But the computational times of the cases are better than they are with the pp-SwpN algorithm. The number of solved SCSF problem is less than or equal to the number of times in the pp-SwpN algorithm. This clearly shows that solving the DC-OPF problem is computationally less expensive than solving a SCSF problem. Therefore pp-SwpN-NS performs better than the pp-SwpN algorithm.

5.4.4 The pp-SwpN-NoNS Algorithm Results

Tables 5.24, 5.25 and 5.26, show the number of times an SCSF problem is solved, the number of suspicious states and also the other performance measures for the pp-SwpN-NoNS algorithm for each created cases. For the small and the medium cases, the pp-SwpN-NoNS algorithm performs as the fastest. However, the performance of the medium-plus cases for the pp-SwpN-NoNS algorithm cannot exceed the performance for the pp-SwpN-NS algorithm.

Table 5.24 The pp-SwpN-NoNS Algorithm Results for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	7	5	1	1	8.56
Small 2	13	22	0.227	1	10.38
Small 3	3	12	0.417	1	3.75
Small 4	1	6	1	1	10.49
Small 5	12	7	1	1	9.54
Small 6	5	14	0.643	1	8.12
Small 7	13	6	1	1	9.40
Small 8	6	5	1	1	6.83
Small 9	9	5	0.6	1	7.57
Small 10	14	22	0.591	1	9.18

Table 5.25 The pp-SwpN-NoNS Algorithm Results for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	6	28	1	1	11.07
Medium 2	43	81	1	1	56.16
Medium 3	74	4	1	1	95.16
Medium 4	88	52	1	1	125.90
Medium 5	43	29	1	1	53.38
Medium 6	26	9	1	1	67.20
Medium 7	6	16	1	1	8.43
Medium 8	6	38	1	1	9.10
Medium 9	6	19	1	1	10.39
Medium 10	19	54	1	1	23.39

Table 5.26 The pp-SwpN-NoNS Algorithm Results for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	131	1316	1	1	768.59
Medium-plus 2	1522	905	0.484	1	6313.62
Medium-plus 3	199	783	0.913	1	1080.62
Medium-plus 4	428	951	0.887	1	1550.49
Medium-plus 5	445	1017	0.888	1	2108.35
Medium-plus 6	124	564	0.995	1	839.73

5.5 Discussions

Since the ratio of found collusive and the ratio of collusive coverage are the same or really close to each other, we compare the SwpN algorithm and its variations according to the computational time.

The computational times of the SwpN-NS, pp-SwpN-NS, and pp-SwpN-NoNS algorithms are close to each other. Moreover, these algorithms perform better than the total enumeration from the computational point of view. Especially for the small and the medium cases, the pp-SwpN-NoNS algorithm executes faster. For the medium-plus cases, mostly the SwpN-NS algorithm performs better.

As mentioned before, we convert the bi-level problem to a single-level problem by applying the Strong Duality theorem. However, there are several ways to convert a bi-level problem. In order to observe how the reformulations affect the performance of the SwpN algorithm and its variations, we execute the algorithms by using the MILP model based on SOS type 1 variables, which is defined in Ebadi Torkayesh (2020). The formulation is provided in Appendix B and the results of the algorithms executed with the reformulation based on SOS type 1 variables are provided in Appendix C. According to the results, for the medium and medium-plus cases, the algorithms perform faster with the reformulation based on SOS type 1 variables. However, the computational times of the algorithms with the SCSF problem are better for the small cases. Moreover, the performance of the pp-SwpN-NS algorithm exceeds the performance of the other algorithms.

6. CONCLUSION

In this study, we present a search algorithm and its variations to detect collusion in deregulated electricity markets; our aim is to develop effective and efficient heuristic algorithms, particularly in comparison to the total enumeration. Therefore, we create 26 cases that belong to 3 different problem sizes in order to compare the performance of the algorithms to the total enumeration results and among themselves. The performances are compared based on accuracy and speed.

Initially, the algorithms are executed by using the SCSF problem. While comparing the results of the Elementary Search and SwpN algorithms, we observe that adding elimination criteria based on partial Nash information increases the accuracy of detecting the actual collusive states as suspicious. Since the elimination criteria are not changed among the SwpN algorithm and its variations, the accuracy of detecting the actual collusive states as suspicious is not changed. However, the computational time differs. According to the results of the SwpN algorithm and its variations, the pp-SwpN algorithm performs as the slowest. The computational times increase as the numbers of SCSF problem solved increase. For the majority, the pp-SwpN-NoNS algorithm performs as the fastest for the small and medium cases, and the SwpN-NS algorithm is the fastest for the medium-plus cases. Moreover, these computational times are much better than the computational times of the total enumeration, except in some cases.

Furthermore, we execute the SwpN algorithm and its variations with the MILP model based on SOS type 1 variables reformulation. The reformulation is developed in Ebadi Torkayesh (2020) as an alternative formulation to the SCSF problem. Even though the performances of the small cases cannot exceed the performance of the total enumeration (in terms of accuracy and speed), for the medium and medium-plus cases the computational times are decreased and the accuracy of detecting the actual collusive states as suspicious are close to 1. Moreover, in comparison between the variations of the SwpN algorithm, the pp-SwpN-NS algorithm performs the fastest.

In conclusion, the ISO as the decision-maker may change its algorithm preference based on the problem size and the method of conversion from a bi-level to a single-level problem. For small cases, the pp-SwpN-NoNS algorithm with the SCSF problem will be more useful. For larger cases, the pp-SwpN-NS algorithm with the reformulation based on SOS type 1 variables should be preferred.

As future work, the SwpN algorithm and its variations can be implemented with alternative formulations. An example of such a reformulation is converting a bi-level problem to a single-level by using KKT conditions. Furthermore, an alternative formulation to the DC-OPF problem can be used to solve the market-clearing problem.

Another future work area might be extensions of the bid sets or/and the number of power generation companies. Total enumeration might not be able to detect collusion in polynomial time as the problem size gets larger. However, the SwpN algorithm and its variations might solve the extended problem in a reasonable time.

BIBLIOGRAPHY

Aliabadi, D. E., Kaya, M., & Şahin, G. (2016). Determining collusion opportunities in deregulated electricity markets. *Electric Power Systems Research*, *141*, 432–441.

Anderson, E. J. & Cau, T. D. H. (2011). Implicit collusion and individual market power in electricity markets. *European Journal of Operational Research*, *211*, 403–414.

Bernheim, B. D. & Whinston, M. D. (1990). Multimarket contact and collusive behavior. *The RAND Journal of Economics*, *21*, 1–26.

Blume, A. & Heidhues, P. (2008). Modeling tacit collusion in auctions. *Journal of Institutional and Theoretical Economics*, *164*, 163–184.

Bunn, D. W. & Oliveira, F. S. (2001). Agent-based simulation- an application to the new electricity trading arrangements of england and wales. *IEEE Transactions on Evolutionary Computation*, *5*, 493–503.

Ebadi Torkayesh, A. (2020). Reformulations of a bi-level optimization problemdetecting collusions in deregulated electricity markets.

Fabra, N. & Toro, J. (2005). Price wars and collusion in the spanish electricity market. *International Journal of Industrial Organization*, *23*, 155–181.

Guan, X., Ho, Y., & Pepyne, D. (2001). Gaming and price spikes in electric power markets. *IEEE Transactions on Power Systems*, *16*, 402–408.

Harrington, J., Hobbs, B., Pang, J., Liu, A., & Roch, G. (2005). Collusive game solutions via optimization. *Mathematical programming*, *104*, 407–435.

Krause, T., Beck, E. V., Cherkaoui, R., Germond, A., Andersson, G., & Ernst, D. (2006). A comparison of nash equilibria analysis and agent-based modelling for power markets. *Electrical Power and Energy Systems*, *28*, 599–607.

Lee, K. H. & Baldick, R. (2003). Solving three-player games by the matrix approach with application to an electric power market. *IEEE TRANSACTIONS ON POWER SYSTEMS*, *18*, 1573–1580.

Liu, A. L. & Hobbs, B. F. (2013). Tacit collusion games in pool-based electricity markets under transmission constraints. *Mathematical programming*, *140*, 351–379.

Moiseeva, E., Hesamzadeh, M. R., & Dimoulkas, I. (2014). Tacit collusion with imperfect information: Ex-ante detection. In *2014 IEEE PES General Meeting , Conference & Exposition*, (pp. 1–5). IEEE.

Pozo, D., Sauma, E. E., & Contreras, J. (2013). A three-level static milp model for generation and transmission expansion planning. *IEEE Transactions on Power Systems*, *28*, 202–210.

Ruiz, C., Conejo, A. J., & Arcos, R. (2010). Some analytical results on conjectural variation models for short-term electricity markets. *IET Generation, Transmission & Distribution*, *4*, 257 – 267.

Ruiz, C. & Conejo, A. J. a. G.-B. R. (2008). Some analytical results pertaining to cournot models for short-term electricity markets. *Electric Power Systems Research*, *78*, 1672–1678.

Ruiz, C., Kazempour, J., & Conejo, A. J. (2012). Equilibria in futures and spot electricity markets. *Electric Power Systems Research*, *84*, 1–9.

Shafie-khah, M., Moghaddam, M. P., & Sheikh-El-Eslami, M. K. (2013). Development of a virtual power market model to investigate strategic and collusive behavior of market players. *Energy Policy*, *61*, 717–728.

Sweeting, A. (2007). Market power in the england and wales wholesale electricity market. *The Economic Journal*, *117*, 654–685.

Tellidou, A. C. & Bakirtzis, A. G. (2007). Agent-based analysis of capacity withholding and tacit collusion in electricity markets. *IEEE Transactions on Power Systems*, *22*, 1735 – 1742.

Vanderbei, R. J. (2014). *Linear Programming: Foundations and Extensions* (4th ed.). New York: Springer.

Appendix A

Table 1 The Demand and Cost Values for the Small Cases

Parameter Types	$GenCo_1$	$GenCo_2$	$GenCo_3$	$GenCo_4$	$GenCo_5$
Demand (D_i)	3	10	276	34	201
Cost (C_i)	20	20	-	-	30

Table 2 The Demand and Cost Values for the Medium and Medium-Plus Cases

Parameter Types	$GenCo_1$	$GenCo_2$	$GenCo_3$	$GenCo_4$	$GenCo_5$	$GenCo_6$	$GenCo_7$
Demand (D_i)	17	10	8	6	13	0	12
Cost (C_i)	20	20	-	-	30	10	-

Table 3 The P^{max} Values for the Small Cases

Cases	P_1^{max}	P_2^{max}	P_5^{max}
Small 1	320	258	214
Small 2	262	495	249
Small 3	371	202	362
Small 4	363	450	415
Small 5	353	260	346
Small 6	345	259	272
Small 7	355	326	236
Small 8	340	278	234
Small 9	234	302	232
Small 10	234	487	327

Table 4 The P^{max} Values for the Medium Cases

Cases	P_1^{max}	P_2^{max}	P_5^{max}	P_6^{max}
Medium 1	43	43	43	31
Medium 2	21	23	28	23
Medium 3	36	34	30	31
Medium 4	36	46	33	41
Medium 5	24	32	39	37
Medium 6	65	25	33	47
Medium 7	62	45	45	44
Medium 8	80	42	31	32
Medium 9	25	69	74	57
Medium 10	55	31	15	69

Table 5 The P^{max} Values for the Medium-Plus Cases

Cases	P_1^{max}	P_2^{max}	P_5^{max}	P_6^{max}
Medium-Plus 1	132	291	31	85
Medium-Plus 2	61	34	24	21
Medium-Plus 3	33	54	48	26
Medium-Plus 4	29	52	30	31
Medium-Plus 5	31	54	21	33
Medium-Plus 6	27	41	43	28

Table 6 The F^{max} Values for the Small Cases

Cases	F_{12}^{max}& F_{21}^{max}	F_{13}^{max}& F_{31}^{max}	F_{24}^{max}& F_{42}^{max}	F_{25}^{max}& F_{52}^{max}	F_{34}^{max}& F_{43}^{max}	F_{45}^{max}& F_{54}^{max}
Small 1	170	234	195	186	285	367
Small 2	156	466	171	423	207	26
Small 3	124	348	432	473	300	401
Small 4	80	224	251	222	380	300
Small 5	76	227	239	306	150	208
Small 6	103	369	375	253	198	59
Small 7	100	226	492	443	128	35
Small 8	170	234	195	186	285	367
Small 9	367	210	167	324	264	185
Small 10	264	450	462	478	306	31

Table 7 The F^{max} Values for the Medium Cases

Cases	F_{12}^{max}& F_{21}^{max}	F_{13}^{max}& F_{31}^{max}	F_{24}^{max}& F_{42}^{max}	F_{25}^{max}& F_{52}^{max}	F_{34}^{max}& F_{43}^{max}	F_{45}^{max}& F_{54}^{max}	F_{56}^{max}& F_{65}^{max}	F_{71}^{max}& F_{17}^{max}
Medium 1	29	25	32	17	40	18	43	37
Medium 2	17	29	26	18	23	42	43	37
Medium 3	20	15	32	8	17	11	6	30
Medium 4	16	36	25	8	8	44	7	13
Medium 5	17	25	20	36	39	38	12	33
Medium 6	47	45	13	12	45	31	6	17
Medium 7	29	25	32	17	40	18	43	37
Medium 8	69	99	24	22	34	21	45	35
Medium 9	56	63	33	16	48	18	46	65
Medium 10	23	79	64	59	88	37	97	68

Table 8 The F^{max} Values for the Medium-Plus Cases

Cases	F_{12}^{max}& F_{21}^{max}	F_{13}^{max}& F_{31}^{max}	F_{24}^{max}& F_{42}^{max}	F_{25}^{max}& F_{52}^{max}	F_{34}^{max}& F_{43}^{max}	F_{45}^{max}& F_{54}^{max}	F_{56}^{max}& F_{65}^{max}	F_{71}^{max}& F_{17}^{max}
Medium-Plus 1	278	68	185	45	13	281	24	92
Medium-Plus 2	278	68	185	45	13	281	24	92
Medium-Plus 3	29	14	8	42	10	16	35	27
Medium-Plus 4	29	14	8	42	10	16	35	27
Medium-Plus 5	29	14	8	42	10	16	35	27
Medium-Plus 6	29	14	8	42	10	16	35	27

Appendix B

$$
\begin{array}{lll}
\text{(1a)} & \text{Maximize} \quad \lambda & \\
\text{(1b)} & \text{s.t.} \quad \lambda \leq \sum_{k \in K} \mathrm{Bid}_{ik} V_{ik} - P_i C_i & \forall i \\
\text{(1c)} & b_i = \sum_{k \in K} \mathrm{Bid}_{ik} B_{ik} & \forall i \\
\text{(1d)} & \sum_{k \in K} B_{ik} = 1 & \forall i \\
\text{(1e)} & V_{ik} \leq P_i^{max} B_{ik} & \forall i \text{ and } k \\
\text{(1f)} & V_{ik} \leq P_i & \forall i \text{ and } k \\
\text{(1g)} & V_{ik} \geq P_i - P_i^{max}[1 - B_{ik}] & \forall i \text{ and } k \\
\text{(1h)} & V_{ik} \geq 0 & \forall i \text{ and } k \\
\text{(1i)} & \sum_{ij \in BR} \gamma_{ij}(LMP_j - LMP_i) + \sum_{ij \in BR} \gamma_{ij}(\psi_{ij}^- - \psi_{ij}^+) + \sum_{ji \in BR} \gamma_{ji}(\psi_{ji}^+ - \psi_{ji}^-) = 0 & \forall i \\
\text{(1j)} & P_i - D_i = \sum_{ij \in BR} \gamma_{ij}(\theta_i - \theta_j) & \forall i \\
\text{(1k)} & P_i \leq P_i^{max} & \forall i \\
\text{(1l)} & b_i - LMP_i + \phi_i \geq 0 & \forall i \\
\text{(1m)} & v_i^{11} = P_i & \forall i \\
\text{(1n)} & v_i^{12} = b_i - LMP_i + \phi_i & \forall i \\
\text{(1o)} & v_i^{21} = \phi_i & \forall i \\
\text{(1p)} & v_i^{22} = P_i^{max} - P_i & \forall i \\
\text{(1q)} & v_{ij}^{31} = \psi_{ij}^+ & \forall ij \in BR \\
\text{(1r)} & v_{ij}^{32} = F_{ij}^{max} - \gamma_{ij}(\theta_i - \theta_j) & \forall ij \in BR \\
\text{(1s)} & v_{ij}^{41} = \psi_{ij}^- & \forall ij \in BR \\
\text{(1t)} & v_{ij}^{42} = F_{ij}^{max} + \gamma_{ij}(\theta_i - \theta_j) & \forall ij \in BR \\
\text{(1u)} & \gamma_{ij}(\theta_i - \theta_j) \leq F_{ij}^{max} & \forall ij \in BR
\end{array}
$$

(1v) $\gamma_{ij}(\theta_i - \theta_j) \geq -F_{ij}^{max}$ $\forall ij \in BR$

(1w) $-\pi \leq \theta_i \leq \pi$ $\forall i$

(1x) $v_i^{11}, v_i^{12}, v_i^{21}, v_i^{22} \quad SOS1$ $\forall i$

(1y) $v_{ij}^{31}, v_{ij}^{32}, v_{ij}^{41}, v_{ij}^{42} \quad SOS1$ $\forall ij \in BR$

(1z) $b_i \in Bid_i$ $\forall i$

(1aa) $P_i \geq 0$ $\forall i$

(1ab) LMP_i free $\forall i$

(1ac) $B_{ik} \in \{0,1\}$ $\forall i$ and k

(1ad) $V_{ik}, \psi_{ij}^{+}, \psi_{ij}^{-} \geq 0$ $\forall i$ and k

Appendix C

Table 9 The Results of the SwpN Algorithm with the Reformulation based on SOS Type 1 Variables for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	66	5	1	1	16.00
Small 2	87	22	0.227	1	17.84
Small 3	14	12	0.417	1	3.96
Small 4	40	6	1	1	16.03
Small 5	80	7	1	1	26.51
Small 6	68	14	0.643	1	26.12
Small 7	99	6	1	1	41.18
Small 8	66	5	1	1	23.77
Small 9	56	5	0.6	1	14.53
Small 10	87	22	0.591	1	17.68

Table 10 The Results of the SwpN Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	38	28	1	1	11.43
Medium 2	398	81	1	1	149.62
Medium 3	570	4	1	1	254.43
Medium 4	927	52	1	1	558.93
Medium 5	271	29	1	1	110.73
Medium 6	398	9	1	1	157.55
Medium 7	38	16	1	1	12.34
Medium 8	38	38	1	1	12.60
Medium 9	28	19	1	1	9.46
Medium 10	89	54	1	1	26.46

Table 11 The Results of the SwpN Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	1840	1316	1	1	1429.19
Medium-plus 2	3359	845	0.518	1	3075.82
Medium-plus 3	1208	784	0.912	1	1621.77
Medium-plus 4	1633	951	0.888	1	1788.04
Medium-plus 5	1633	1018	0.887	1	1165.94
Medium-plus 6	1416	598	0.938	1	1773.54

Table 12 The Results of the SwpN-NS Algorithm with the Reformulation based on SOS Type 1 Variables for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	12	5	1	1	16.14
Small 2	17	22	0.227	1	16.80
Small 3	4	12	0.417	1	6.64
Small 4	11	6	1	1	16.71
Small 5	15	7	1	1	17.23
Small 6	11	14	0.643	1	13.64
Small 7	18	6	1	1	13.80
Small 8	13	5	1	1	15.20
Small 9	12	5	0.6	1	11.75
Small 10	18	22	0.591	1	14.82

Table 13 The Results of the SwpN-NS Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	6	28	1	1	14.03
Medium 2	55	81	1	1	85.69
Medium 3	81	4	1	1	129.18
Medium 4	126	52	1	1	207.97
Medium 5	46	29	1	1	93.09
Medium 6	56	9	1	1	139.96
Medium 7	6	16	1	1	11.96
Medium 8	6	38	1	1	10.58
Medium 9	6	19	1	1	8.13
Medium 10	19	54	1	1	22.09

Table 14 The Results of the SwpN-NS Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	131	1316	1	1	356.20
Medium-plus 2	1547	849	0.516	1	2875.14
Medium-plus 3	199	782	0.914	1	686.44
Medium-plus 4	432	951	0.888	1	1692.36
Medium-plus 5	441	1016	0.889	1	788.71
Medium-plus 6	124	564	0.995	1	688.43

Table 15 The Results of the pp-SwpN Algorithm with the Reformulation based on SOS Type 1 Variables for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	53	5	1	1	16.77
Small 2	65	22	0.227	1	16.15
Small 3	13	12	0.417	1	4.37
Small 4	1	6	1	1	7.98
Small 5	63	7	1	1	18.92
Small 6	53	14	0.643	1	16.32
Small 7	66	6	1	1	18.73
Small 8	47	5	1	1	15.85
Small 9	43	5	0.6	1	10.62
Small 10	65	22	0.591	1	15.33

Table 16 The Results of the pp-SwpN Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	38	28	1	1	10.76
Medium 2	362	81	1	1	122.22
Medium 3	538	4	1	1	184.08
Medium 4	756	52	1	1	352.37
Medium 5	253	29	1	1	88.79
Medium 6	285	9	1	1	119.91
Medium 7	38	16	1	1	10.11
Medium 8	38	38	1	1	10.44
Medium 9	28	19	1	1	8.06
Medium 10	89	54	1	1	20.05

Table 17 The Results of the pp-SwpN Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	1840	1316	1	1	1445.67
Medium-plus 2	2757	847	0.518	1	3324.92
Medium-plus 3	1208	785	0.911	1	1087.95
Medium-plus 4	1633	951	0.888	1	1561.48
Medium-plus 5	1633	1017	0.888	1	1470.10
Medium-plus 6	1416	598	0.938	1	1375.20

Table 18 The Results of the pp-SwpN-NS Algorithm with the Reformulation based on SOS Type 1 Variables for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	10	5	1	1	13.11
Small 2	14	22	0.227	1	14.98
Small 3	4	12	0.417	1	5.54
Small 4	1	6	1	1	10.99
Small 5	14	7	1	1	14.73
Small 6	10	14	0.643	1	12.16
Small 7	14	6	1	1	13.95
Small 8	12	5	1	1	11.62
Small 9	11	5	0.6	1	11.46
Small 10	14	22	0.591	1	15.44

Table 19 The Results of the pp-SwpN-NS Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	6	28	1	1	8.96
Medium 2	56	81	1	1	66.92
Medium 3	75	4	1	1	73.75
Medium 4	108	52	1	1	117.59
Medium 5	42	29	1	1	50.59
Medium 6	52	9	1	1	77.06
Medium 7	6	16	1	1	8.15
Medium 8	6	38	1	1	8.13
Medium 9	6	19	1	1	8.26
Medium 10	19	54	1	1	19.28

Table 20 The Results of the pp-SwpN-NS Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	131	1316	1	1	295.41
Medium-plus 2	1546	844	0.519	1	2734.65
Medium-plus 3	201	784	0.912	1	739.62
Medium-plus 4	431	951	0.888	1	1061.03
Medium-plus 5	442	1016	0.889	1	1018.45
Medium-plus 6	124	564	0.995	1	643.26

Table 21 The Results of the pp-SwpN-NoNS Algorithm with the Reformulation based on SOS Type 1 Variables for the Small Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Small 1	8	5	1	1	15.01
Small 2	12	22	0.227	1	15.22
Small 3	3	12	0.417	1	6.16
Small 4	1	6	1	1	15.27
Small 5	12	7	1	1	13.57
Small 6	5	14	0.643	1	12.38
Small 7	13	6	1	1	17.19
Small 8	6	5	1	1	13.29
Small 9	11	5	0.6	1	11.35
Small 10	13	22	0.591	1	14.97

Table 22 The Results of the pp-SwpN-NoNS Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium 1	6	28	1	1	13.32
Medium 2	43	81	1	1	93.21
Medium 3	76	4	1	1	121.03
Medium 4	91	52	1	1	195.37
Medium 5	42	29	1	1	88.89
Medium 6	26	9	1	1	126.33
Medium 7	6	16	1	1	14.46
Medium 8	6	38	1	1	12.41
Medium 9	6	19	1	1	8.30
Medium 10	19	54	1	1	20.62

Table 23 The Results of the pp-SwpN-NoNS Algorithm with the Reformulation based on SOS Type 1 Variables for the Medium-Plus Cases

Cases	Number of solved SCSFP	Number of Suspicious States	Ratio of Found Collusive	Ratio of Collusive Coverage	Computational Time
Medium-plus 1	131	1316	1	1	321.08
Medium-plus 2	1521	852	0.514	1	2744.03
Medium-plus 3	201	784	0.912	1	772.86
Medium-plus 4	431	951	0.882	1	432.42
Medium-plus 5	442	1016	0.889	1	1207.19
Medium-plus 6	124	564	0.995	1	566.30