Saint Petersburg State University

Department of mathematical game theory and statistical decisions

Ivan Orlov

Master's thesis

Hierarchical Model of Corruption: Game-Theoretic Approach

Specialization 01.04.02

Applied Mathematics and Informatics

Master's Program Game Theory and Operations Research

Research advisor,

Kumacheva Suriya Sh.

Saint Petersburg

2021

Contents

1 Introduction

Transparency International [1] defines corruption as *"the abuse of entrusted power for private gain"*. It is a worldwide problem and, sadly, Russian Federation (as shown by Buckley [2]) is not an exception. Quite on the contrary, it is among the "leaders" ranking 129 out of 180 countries in Corruption Perception Index of 2019 [3] (meaning "very corrupt"), which shows the relevance of the problem.

Corruption occurs in relations between people and companies – agents that should make strategic decisions in order to benefit from it. This quality makes it possible to use game-theoretic apparatus to analyze it. There are many scientific works on the topic yet they mostly address the corruption in form of a game between two or three players. This research differs in its approach: it analyzes corrupt officials acting as parts of a bigger hierarchical structure rather than isolated agents in hope of obtaining insights that may help combat corruption in organizations.

Research object is corruption (embezzlement and bribery) within a hierarchy.

Aim of this study is to analyze corruption in hierarchical context and find conditions under which it is minimal.

Objectives:

1. Study the relevant literature.
2. Create and study the hierarchical model of corruption (both non-cooperative and cooperative cases).
3. Write a code simulation for the model.
4. Solve the particular case of the model.

5. Analyze the solution.
6. Find the conditions for corruption minimization.

2 Main part

2.1 Literature review

Spengler [4] in great depth (analysis, two extensions, three player types, laboratory experiments) studies the extensive-form game between Client, Official and Inspector (Figure 2.1) and improves previous models by making probabilities of actions endogenous, suggests mixed equilibrium as solution and asymmetric penalties (with focus on officials) as anti-corruption measure. The carcass of the game inspired the inspection stage of this research.

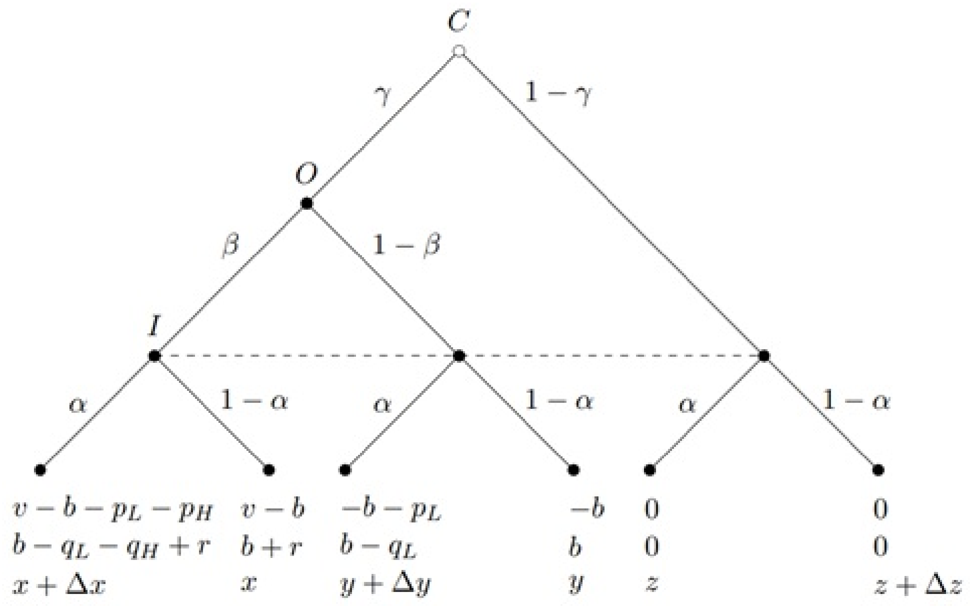


Figure 2.1: Extensive-form game without reporting.

Attanasi et al. [5] focus on the psychological aspect of embezzlement game with player triplet Donor-Intermediary-Recipient. They study what moral presuppositions players have and experimentally derive what their irrational guilt-averse moves are. Their results showcase that stealing intermediary has guilt towards both the recipient, whose payoffs he affects, and the donor, whose he does not. The study suggests that if the results are true producing high public expectations of morality of intermediaries would reduce embezzlement.

Shenje [6] studies Briber-Bribee (based on Zimbabwean public sector corruption) and comes to the mixed Nash Equilibrium solution based on the values of costs and incomes. The way to affect these values is again top-down and varies from policy recommendations to educating the officials. Song et al. [7] focuses on Committee-Department embezzlement game (based on Chinese corruption) and comes to conclusions similar to Shenje.

Zyglidopoulos et al. [8] studies corruption in multinational companies and outlines tetrad of conditions needed for its success:

1. Existence of opportunity for corrupt action.

2. Small risk of negative repercussions.

3. Willingness to engage in corrupt activity.

4. Capability to act in a corrupt way.

Kumacheva [9] presents a multi-stage hierarchical game, which studies corruption in forms of tax evasion and auditor bribing, which inspired the model of this study. The work considers three-level structure: administration, inspector, taxpayers. Taxpayers declare their level of income and choose the size of bribe, administration chooses probabilities of auditing and reauditing, inspector chooses to accept or reject the suggested bribe. The solution suggests that the administration should choose probabilities of auditing and reauditing that depend on the tax, penalty and fine rates and taxpayers should declare their true level of income. The extension for inspection mistakes is also considered.

Gorbaneva et al. [10] analyze corruption via hierarchical control systems, namely investment-construction projects and electricity theft. Hierarchy is comprised of triplet "principal-supervisor-agent". In the first system the supervised competition for resources is considered and allocations in situations of no bribes and Nash equilibrium in simultaneous bribing game of n agents are suggested with comparison between corrupt and non-corrupt cases considered for $n = 2$. The condition for bribing to be unprofitable for the supervisor is provided. In the second system the electricity provider (principal) sends the inspector (supervisor) to check whether the client company (agent) declares their consumption truthfully (which is akin to tax evasion problem). The condition for the agent to report the actual consumption and the ways for the principal to ensure this condition are given.

Gorbaneva and Ougolnitsky [11] study concordance of public and private interests models with different profit functions of the society and individuals. The main parameter of analysis is price of anarchy (ratio of values of the game in the worst Nash equilibrium to the best situation) and social price of anarchy (the same but the public benefit is used instead of values of the game). The utility of using impulsion (economic) and compulsion (administrative) methods to improve these parameters is examined. The ideas of meta-game synthesis (including corrupt version) are suggested.

Vasin and Panova [12] discuss corruption (taxpayers' evasion and bribing the inspector) in transition economies taking Russia as example. Their model depicts a hierarchical game: homogenous population of taxpayers with income distributed according to some density function, each taxpayer declares a level of income that maximizes their utility and the authority chooses the audit probability that does the same for it. The non-corrupt models of progressive tax and linearly dependent on undeclared income fines are studied. The corrupt model includes homogeneous taxpayers with two possible levels of income (high and low), inspecting auditor which can be bribed and center which tries to maximize its payoff – sum of all taxes and fines minus costs of inspection (auditor checks taxpayer) and reinspection (center checks auditor on a declared low taxpayer). Mathematical solutions based on parameters (size of tax, fine, bribe, costs of inspection and reinspection) are suggested. Authors also describe possible applications of their results to the Russian economy, they give the optimal audit probability for the rates of 1997, the cut-off difference between a priori and declared profit, probabilistic cut-off for enterprises to be audited selection, warning on the irrelevance of the assumptions in case of organized corruption.

Savvateev [13] studies corruption and lobbying in transition economies. The first model includes utility-maximizing manufacturers that compete for a production resource. In the first case there is a possibility of lobbying (which costs some amount of resource) to get subsidies which are collected as taxes from manufacturers; in the second case there is no such possibility and there is a free market of the resource; in the third case there is a mix. With the fixed tax rate the second always Pareto dominates the first, nonetheless there are situations in which the majority of agents will vote against the transition to free market (for example, those who have the bigger amounts of resource benefit from subsidies because they can allocate more amounts into lobbying to get it), even though the total production of the latter is lower.
The second model studies "principal-agent" framework of the controlling superior and the working subordinate (subordinates) in a Stackelberg competition. Each subordinate simultaneously chooses the level of corruption knowing what investigation intensities (based on the levels of corruption) the superior allocated. Cut-off strategies that constitute a strong Nash equilibrium (coalitionally or anti-coalitionally stable) are suggested to be the solution of the game. For the one-type subordinates (equal corruption opportunities) the superior can ensure less than absolute level of corruption

(the value depends on size of fine and amount of available resources). In case of two types of subordinates there is a "chain reaction effect": the less corrupt agents choose not to be corrupt at all and the more corrupt agents choose to be corrupt, yet get all the attention of the superior, who does not waste any resources on checking the first type agents, then in second iteration agents of second type reduce their level of corruption, i.e. the choice of less corrupt affects the choice of more corrupt. In case of N types the conditions for "chain reaction effect" to occur are suggested. These suggestions are similar to "broken windows theory": in case of different levels of corruption, the authorities should fight the low-level because it will affect every other level up to the top.

2.2 Model

2.2.1 Description

The corruption is modeled as a hierarchical game consisting of two stages: embezzlement and inspection. The players are supposed to be risk-neutral and utility-maximizing. Only monetary payoffs are considered (although, the monetized value of anything can be used in the formulas).

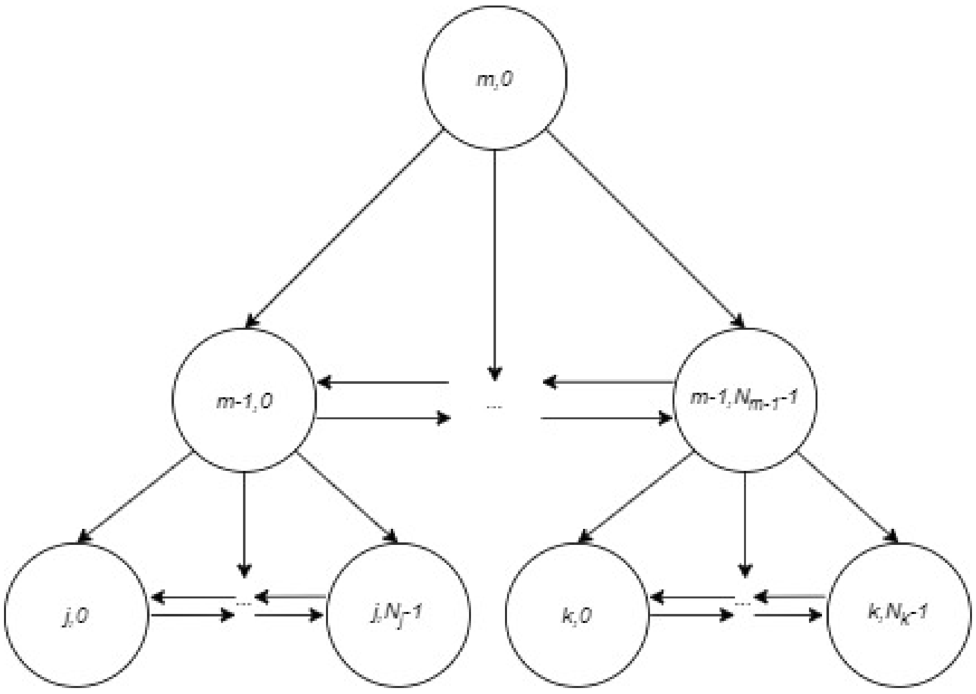


Figure 2.2: Hierarchy of the officials.

Hierarchy is a directed graph with the following meaning of the links:

- XY – X is the superior of Y;
- YX – X is the subordinate of Y;
- both XY and YX – X and Y are colleagues (equals);
- neither XY nor YX – X and Y are unrelated (they are on different levels with no superior-subordinate relationships).

$$(j,k) = boss(n) : \quad (n,i) \in subs(j,k) \quad \forall (n,i) \in C_n$$

In the first stage the company allocates amount of money M_m to solve a problem. This money goes down the hierarchy of officials (Figure 2.2) with each of them having a chance to embezzle some of it before passing it to subordinates. The cut-off value M_n is the minimal amount of money that needs to leave level n in order to create at least semblance of work (before bloating the budget). G_n is the amount of money entering level n. The steal

$$S^*_{n,i} = \frac{G_n - M_n}{N_n}$$

is optimal. Any $S_{n,i} > S^*_{n,i}$ is not optimal since it either breaks the cut-off condition or causes stealing from a colleague on the same level (which creates the possibility of being exposed). Any $S_{n,i} < S^*_{n,i}$ is not optimal since it is possible to get more. It is also important to note that $S^*_{n,i}$ is optimal from the risk-neutral and utility-maximizing perspective only in case it is possible to bribe the inspector with the amount of money less than the stealing; otherwise, it is better not to steal at all.

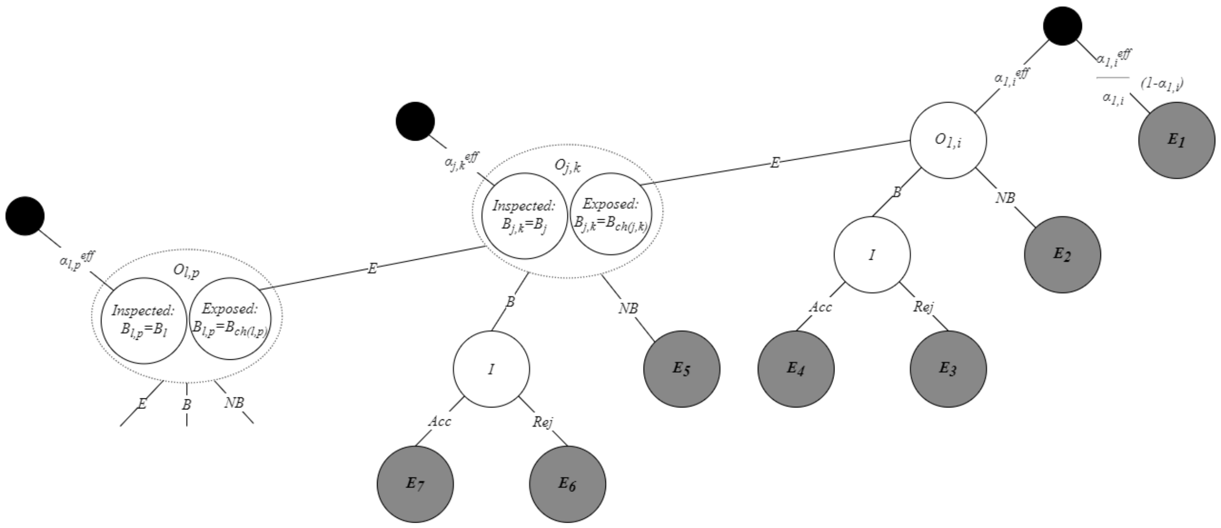


Figure 2.3: Graph of the inspection game.

$$I - inspector; \quad O_{x,y} - official\ (x, y); \quad E_Z - end\ (outcome)\ Z;$$
$$boss(1) = (j, k); \quad boss(j) = (l, p)$$

In the second stage inspector checks some official $O_{n,i}$ for corruption. The inspector has perfect technology so, if there has been an embezzlement, it will be revealed. The probability of inspection is proportional to the total amount of stealing up to this level (formula (2.2)). The inspector goes through hierarchy from top to bottom, from left to right and inspects the next level only if the previous one was not inspected (formula (2.3)). The highest official – the state (the official in the root node of the hierarchy) does not steal and thus is not inspected (formula (2.4)). From the inspector's point of view, all officials on one level are equivalent (formula (2.5)).

$$S_n = \sum_{i=0}^{N_n-1} S_{n,i} \tag{2.1}$$

$$\alpha_n = \frac{\sum_{j=n}^{m-1} S_j}{M_m} \tag{2.2}$$

$$\alpha_n^{eff} = \alpha_n \prod_{k=n+1}^{m} (1-\alpha_k) \tag{2.3}$$

$$S_m = 0 \;\rightarrow\; \alpha_m^{eff} = \alpha_m = 0 \tag{2.4}$$

$$\alpha_{n,i} = \frac{\alpha_n}{N_n} \quad \alpha_{n,i}^{eff} = \frac{\alpha_n^{eff}}{N_n} \tag{2.5}$$

The inspected official has three possible actions (Figure 2.3):

1. B – attempt to bribe the inspector (size is a natural number chosen at will);
2. NB – do not attempt to bribe the inspector;
3. E – expose the stealing of someone who stole more (boss).

In the first case depending on the size of the bribe inspector either accepts or rejects it. If the bribe is accepted, official $O_{n,i}$ loses it but keeps full stealing. Let $0 \leq \kappa_{n,i} \leq 1$ be the part of stealing that official managed to hide (offshore company, friend or relative). Then in case of rejected bribe the official keeps the amount $\kappa_{n,i}S_{n,i}$, loses the bribe and will have to pay fines for steal $F(W_{n,i}, S_{n,i})$ and bribe $Fb(B_{n,i})$. In the second case the official pays full fine and keeps $\kappa_{n,i}S_{n,i}$. In the third case the exposed official $O_{l,j}$ is making a decision. Let $0 \leq \theta_{n,i} \leq 1$ be the part of fine that official will have to pay because of cooperation (he will be pardoned from paying $(1-\theta_{n,i})F(W_{n,i}, S_{n,i})$). If official $O_{l,j}$ does not bribe the inspector or the bribe is rejected, $O_{n,i}$ will have to pay $\theta_{n,i}F(W_{n,i}, S_{n,i})$. If the bribe is accepted then stealing of both officials will be covered up and no fine will be imposed, stealing will be kept in full.

The inspector decides to accept the bribe and cover the stealing up (for the cost $Cu(S_{n,i})$) or to reject the bribe and investigate further (to get the reward $R(S_{n,i})$). They also bear the inspection cost Ci_n in both cases.

Payoffs in each end are as follows (in format $E_X : U_{j,k} \ ; U_{1,i} \ ; U_I$):
$E_1 : W_{j,k} + S_{j,k} \ ; W_{1,i} + S_{1,i} \ ; W_I$
$E_2 : W_{j,k} + S_{j,k} \ ; W_{1,i} + \kappa_{1,i}S_{1,i} - F(W_{1,i}, S_{1,i}) \ ; W_I + R(S_{1,i}) - Ci_1$
$E_3 : W_{j,k} + S_{j,k} \ ; W_{1,i} + \kappa_{1,i}S_{1,i} - (F(W_{1,i}, S_{1,i}) + B_{1,i} + Fb(B_{1,i})) \ ; W_I + R(S_{1,i}) - Ci_1$
$E_4 : W_{j,k} + S_{j,k} \ ; W_{1,i} + S_{1,i} - B_{1,i} \ ; W_I + B_{1,i} - Ci_1 - Cu(S_{1,i})$
$E_5 : W_{j,k} + \kappa_{j,k}S_{j,k} - F(W_{j,k}, S_{j,k}) \ ; W_{1,i} + \kappa_{1,i}S_{1,i} - \theta_{1,i}F(W_{1,i}, S_{1,i}) \ ; W_I - (Ci_1 + Ci_j) + R(S_{1,i}) + R(S_{j,k})$
$E_6 : W_{j,k} + \kappa_{j,k}S_{j,k} - (F(W_{j,k}, S_{j,k}) + B_{j,k} + Fb(B_{j,k})) \ ; W_{1,i} + \kappa_{1,i}S_{1,i} - \theta_{1,i}F(W_{1,i}, S_{1,i}) \ ; W_I - (Ci_1 + Ci_j) + R(S_{1,i}) + R(S_{j,k})$
$E_7 : W_{j,k}+S_{j,k}-B_{j,k} \ ; W_{1,i}+S_{1,i} \ ; W_I+B_{j,k}-(Ci_1+Ci_j+Cu(S_{1,i})+Cu(S_{j,k}))$
All subsequent ends are similar to E_5, E_6, E_7 with the difference in the set of the exposed officials.

Table 2.1: Ends' descriptions.

End	Description
1	No inspection.
2	Subordinate is inspected, no bribe.
3	Subordinate is inspected, bribe is rejected.
4	Subordinate is inspected, bribe is accepted.
5	Boss is exposed by the subordinate, no bribe.
6	Boss is exposed by the subordinate, bribe is rejected.
7	Boss is exposed by the subordinate, bribe is accepted.

The official's total utility of is comprised of wage, stealing and expected loss, which depends on his actions and actions of other players:

$$U_{n,i}(S_{n,i}, B_{n,i}, A_{n,i}) = W_{n,i} + S_{n,i} - \alpha^{+}_{n,i} L(A_{n,i}, A_{-n,i})$$

$$A_{n,i} \in \{B, NB, E\} \quad n \neq m-1, m \;\; i = 0 \dots N_n - 1$$

$$A_{m-1,i} \in \{B, NB\} \quad i = 0 \dots N_{m-1} - 1$$

$$A_{m,0} \in \emptyset$$

$$A_I \in \{Acc, Rej\}$$

$$A_{-n,i} = (A_{k,j}, \dots, A_I) \;\; \forall (k,j) \neq (n,i)$$

$\alpha^{+}_{n,i}$ is the chance of inspection (both direct and via being exposed) that is calculated as follows:

$$\alpha^{+}_{n,i} = \alpha^{eff}_{n,i} + \sum_{(l,j) \in SE(n,i)} \alpha^{+}_{l,j},$$

where $SE(n,i) = \{(v,p)\} : \; (v,p) \in subs(n,i) \;\&\; A_{v,p} = E$

The inspector's utility is as follows:

$$U_I(A_I, A_{n,i}, T) = W_I + \alpha^{+}_{n,i} K(A_I, A_{n,i}, T),$$

where

$$K(A_I, A_{n,i}, T) = \begin{cases} K(A_I, A_{boss(n)}, T \cup \{(n,i)\})\ if A_{n,i} = E \\ B_{n,i} - \sum_{(l,j)\in T}[Cu(S_{l,j}) + Ci_l]\ if A_{n,i} = B\ \&\ A_I = Acc \\ \sum_{(l,j)\in T}[R(S_{l,j}) - Ci_l]\ if A_{n,i} \in \{B, NB\}\ \&\ A_I = Rej \end{cases},$$

where W_I – inspector's wage, $T = \{(v,k)\}$ – set of ids of inspected and exposed officials.

The state's utility is calculated as follows:

$$U_s(A_{n,i}, A_I, T) = M_m - \sum_{j=1}^{m-1} S_j - \sum_{X\in\{I\}\cup H} W_X + \alpha^{+}_{n,i} D(A_{n,i}, A_I, T),$$

where

$$D(A_{n,i}, A_I, T) = \begin{cases} F(S_{n,i}, W_{n,i}) + \sum_{(l,j)\in T}[(1-\kappa_{l,j})S_{l,j} - R(S_{l,j})]+ \\ +\sum_{(v,p)\in T\setminus\{(n,i)\}} \theta_{v,p} F(W_{v,p}, S_{v,p})\ if\ A_{n,i} = NB \\ D(A_{boss(n)}, A_I, T \cup \{(n,i)\})\ if\ A_{n,i} = E \\ D(NB, A_I, T) + B_{n,i} + Fb(B_{n,i})\ if\ A_{n,i} = B\ \&\ A_I = Rej \\ 0\ if\ A_{n,i} = B\ \&\ A_I = Acc \end{cases},$$

The level of corruption is

$$LoC = \frac{\sum_{j=1}^{m-1} S_j}{M_m}$$

In this model, conditions from Zyglidopoulos et al. [8] can be seen incorporated in the following way:

1. Opportunity exists because an official has access to the money flow.
2. Risk of negative repercussions is small since the probability of an official being inspected is small, plus they can always try to bribe the inspector.

3. Willingness to engage in corruption is provided by monetary utility maximization of an agent.

4. Capability to act in a corrupt way is shown in abilities to embezzle and bribe.

2.2.2 Example

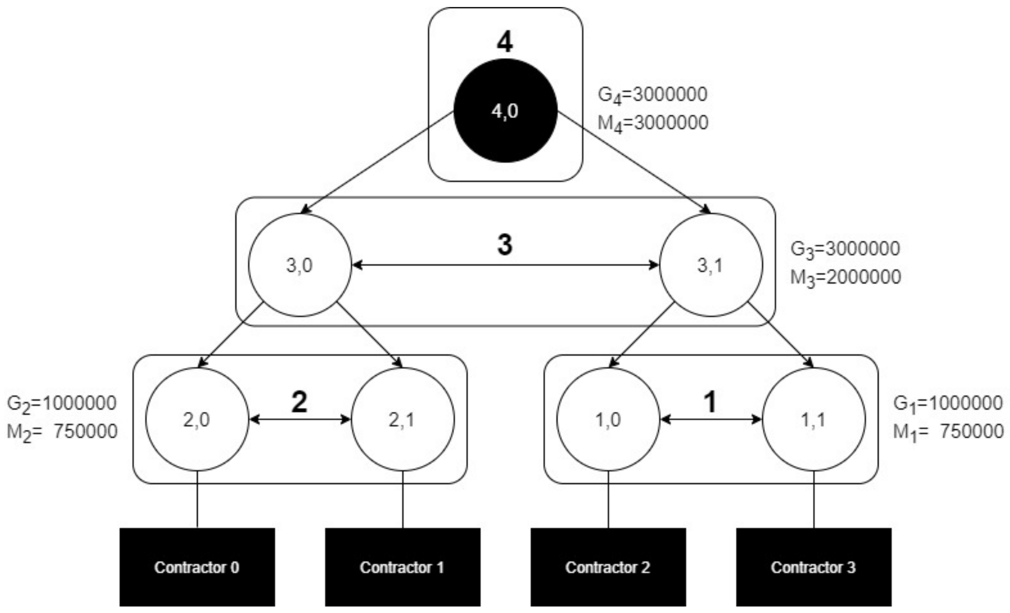


Figure 2.4: The hierarchy of officials in example.

$$boss(n) = \begin{cases} (4,0)\ if\ n = 3 \\ (3,0)\ if\ n = 2 \\ (3,1)\ if\ n = 1 \end{cases}$$

For the constructed scheme, a particular example with two levels and six officials (Figure 2.4) is considered. The company (municipality) allocated 3 million to build a high-quality playground but only half of that sum was given to the contractors, the medium-quality playground is built.

The values for characteristics of players are in Tables 2.2 and 2.3.

Table 2.2: Values of officials' characteristics.

$O_{n,i}$	$W_{n,i}$	$S_{n,i}$	$\kappa_{n,i}$	$\theta_{n,i}$	$\alpha_{n,i}$	$B_{n,i}$	$F(S_{n,i})$	$Fb(B_{n,i})$
$3, i$	90,000	500,000	0.600	–	0.167	150,000	1,620,000	5,625,000
$2, i$	40,000	125,000	0.300	0.010	0.208	62,500	720,000	2,812,500
$1, i$	40,000	125,000	0.300	0.010	0.250	62,500	720,000	2,812,500

Table 2.3: Values of Inspector's characteristics.

W_I	$Ci_{\{1,2\}}$	Ci_3	$R(S_{\{1,2\},i})$	$R(S_{3,i})$	$Cu(S_{\{1,2\},i})$	$Cu(S_{3,i})$
70,000	10,000	25,000	40,000	75,000	5,000	12,500

2.2.3 Solution

The game cannot be solved via backward induction, since official does not know characteristics and utilities of boss and inspector for sure. In order to solve it, the simulation code in Python (the listing is in Appendix A) was written and executed.

Table 2.4: Results of simulation for the initial settings.

	OptOpt_EB	OptOpt_BB	NoneOpt_NBB	OptNone_BNB	NoneNone_NBNB
$(3, 0)$	523,136	564,934	565,055	90,000	90,000
$(3, 1)$	535,972	565,004	564,835	90,000	90,000
$(2, 0)$	165,000	156,277	40,000	162,407	40,000
$(2, 1)$	165,000	156,294	40,000	162,405	40,000
$(1, 0)$	165,000	158,935	40,000	160,187	40,000
$(1, 1)$	165,000	158,975	40,000	160,240	40,000
I	156,602	131,233	105,137	81,219	70,000
State	1,090,000	1,090,000	1,590,000	2,090,000	2,590,000
LoC	0.500	0.500	0.333	0.167	0.000

The analysis of results yields the stable outcome via the following processes (assumption is that all officials are self-interested, utility maximizing and incapable of communicating with each other):

1. Find the action yielding maximal utility for bosses.

2. Find the best response of subordinates to the 1.

3. Find the best response of bosses to the 2.

4. Repeat until there are no deviations.

$OptOpt\_BB \rightarrow OptOpt\_EB \rightarrow OptOpt\_EB$ in case of Table 2.4.

Or:

1. Find the action yielding maximal utility for subordinates.

2. Find the best response of bosses to the 1.

3. Find the best response of subordinates to the 2.

4. Repeat until there are no deviations.

$OptOpt\_EB \rightarrow OptOpt\_EB$ in case of Table 2.4.

The stable outcome is when all officials steal optimally, subordinates expose, bosses bribe and inspector accepts the bribe.

Proposition. The obtained equilibrium cannot be called Nash since due to the lack of information about inspector's payoffs official cannot choose the optimal bribe. We suggest the notion of *Nash-like* equilibrium:

$$(S^*_{n,i}, B^*_{n,i}, A^*_{n,i}) = argmax\{U_{n,i}(S_{n,i}, B_{n,i}, A_{n,i}) \mid B_{n,i} \geq B^v_{n,i}\}.$$

In that equilibrium officials maximize their utility within confines of not knowing three important things: the utility functions of inspector, the action and the bribe size of their boss, and the optimal bribe size. They have only hypothesis $B^v_{n,i}$ of the minimal sufficient bribe – they are not able to suggest the lesser bribe (because they believe it will be rejected).

2.2.4 Corruption Minimization and Sensitivity Analysis

In order to minimize corruption the bribe must be rejected. That will cause official to lose not hidden steal and pay fines, which are supposed to discourage them from stealing in the first place. The ultimate decision (to accept or reject the bribe) is made by the inspector. Since they maximize their utility, it depends on which action yields the most profit, i.e. the sign of the inequality (2.6).

$$U_I(Acc) \gtreqless U_I(Rej) \ \rightarrow \ B_{n,i} - \sum_{(l,j)\in T} Cu(S_{l,j}) \gtreqless \sum_{(l,j)\in T} R_I(S_{l,j}) \tag{2.6}$$

The corruption is minimized when

$$B_{n,i} - \sum_{(l,j)\in T} Cu(S_{l,j}) \leq \sum_{(l,j)\in T} R_I(S_{l,j}) \tag{2.7}$$

$$\sum_{(l,j)\in T} [R(S_{l,j}) + Cu(S_{l,j})] \geq B_{n,i} \tag{2.8}$$

At the same time the size of bribe is chosen by the official: in order to not be corrupt they must get not more from stealing and bribing than from not doing so:

$$U_{n,i}(S^*_{n,i}, B^*_{n,i}, B) - U_{n,i}(0, 0, NB) = S^*_{n,i} - \alpha^+_{n,i} B^*_{n,i} \leq 0 \tag{2.9}$$

By connecting (2.8) and (2.9) we get the anti-corruption setting condition

$$\sum_{(l,j)\in T} [R(S_{l,j}) + Cu(S_{l,j})] \geq \frac{S^*_{n,i}}{\alpha^+_{n,i}} \ \forall T, \tag{2.10}$$

that must be satisfied in the best case for $T = \{O_{n,i}\}$, in the worst case –

$$T = \{O_{n,i}, O_{j,k}, O_{l,p}, \dots\} \ O_{j,k} \in SE(n,i); \ O_{l,p} \in SE(j,k)$$

In order to be accepted, the bribe for inspected chain T must be:

$$B_{optT} > \sum_{(l,j)\in T} [R(S_{l,j}) + Cu(S_{l,j})] \tag{2.11}$$

$$B_{optT}(\zeta) = \sum_{(l,j)\in T} [R(S_{l,j}) + Cu(S_{l,j})] + \zeta \tag{2.12}$$

For the corruption minimization, it must hold that

$$B_{optT}(\zeta) \geq \frac{S^*_{n,i}}{\alpha^+_{n,i}} \tag{2.13}$$

All conclusions valid for $\zeta = x > 0$ are valid for any $\zeta > x$.

Let us provide the example. There are three possible types of chains in the studied hierarchy:

$$T_s = \{O_{2,i}\}; \{O_{1,i}\} \quad T_b = \{O_{3,i}\} \quad T_{ch} = \{O_{2,i}, O_{3,0}\}; \{O_{1,i}, O_{3,1}\} \quad i = 0, 1$$

For simplicity, since levels 1 and 2 are alike (and officials within them are identical), suppose

$$S^*_{1,i} = S^*_{2,i} = S_s \quad B^*_{1,i} = B^*_{2,i} = B_s$$

Since it has already been established that it is optimal for the subordinates to expose their bosses, fighting corruption in chains T_s is senseless: no matter how big the needed bribe is, they will not pay it. It is more useful to fight corruption in chain T_{ch} (make being exposed unprofitable for bosses), then T_b (make being directly inspected unprofitable for bosses) and then T_s under the circumstances of $S_3 = 0$ while following the logic of bigger bribe for bigger stealing. It is possible to formulate three settings, each stricter than the previous.

The height of the dash-dot line on Figure 2.5 is

$$\frac{S^*_{3,i}}{\alpha^+_{3,i}} = \frac{500,000}{\frac{\alpha_3}{2} + min[\alpha^+_{2,0} + \alpha^+_{2,1}; \alpha^+_{1,0} + \alpha^+_{1,1}]} = [\frac{500,000}{0.36111111093055556}] = 1,384,615$$

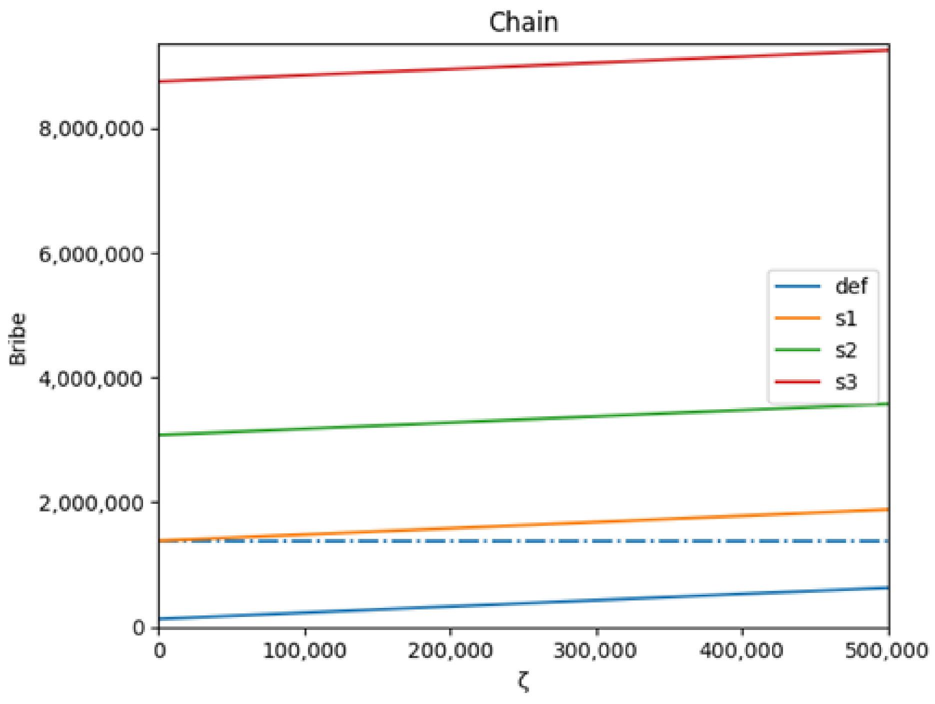


Figure 2.5: The graph of $B_{optT}(\zeta)$ for boss and T_{ch}.

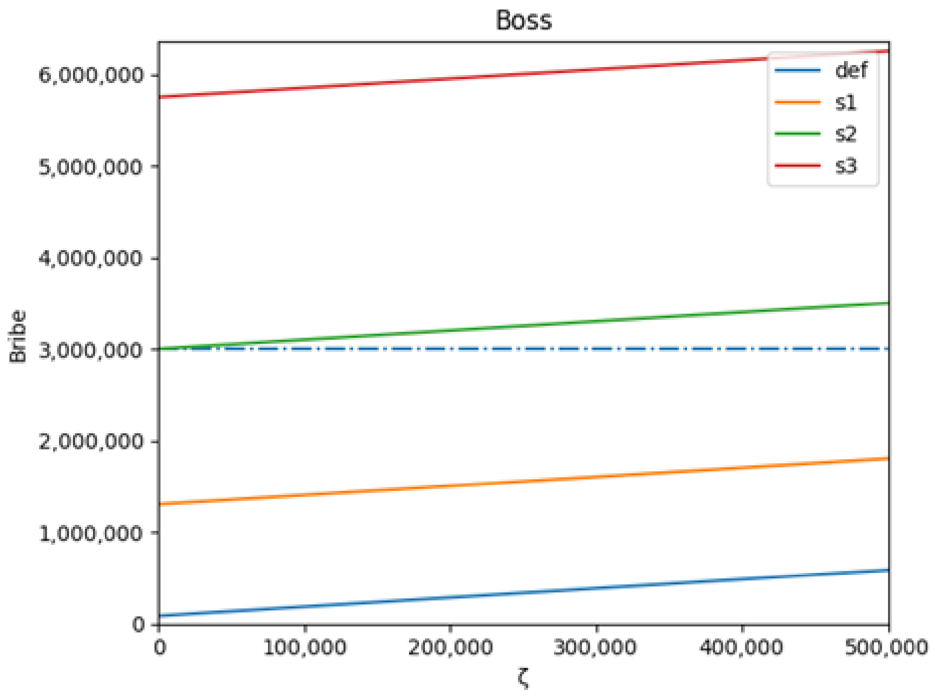


Figure 2.6: The graph of $B_{optT}(\zeta)$ for boss and T_b.

The height of the dash-dot line on Figure 2.6 is

$$\frac{S^*_{3,i}}{\alpha_{3,i}} = \frac{S^*_{3,i}}{\frac{\alpha_3}{2}} = [\frac{2 \cdot 500,000}{0.333}] = 3,000,000$$

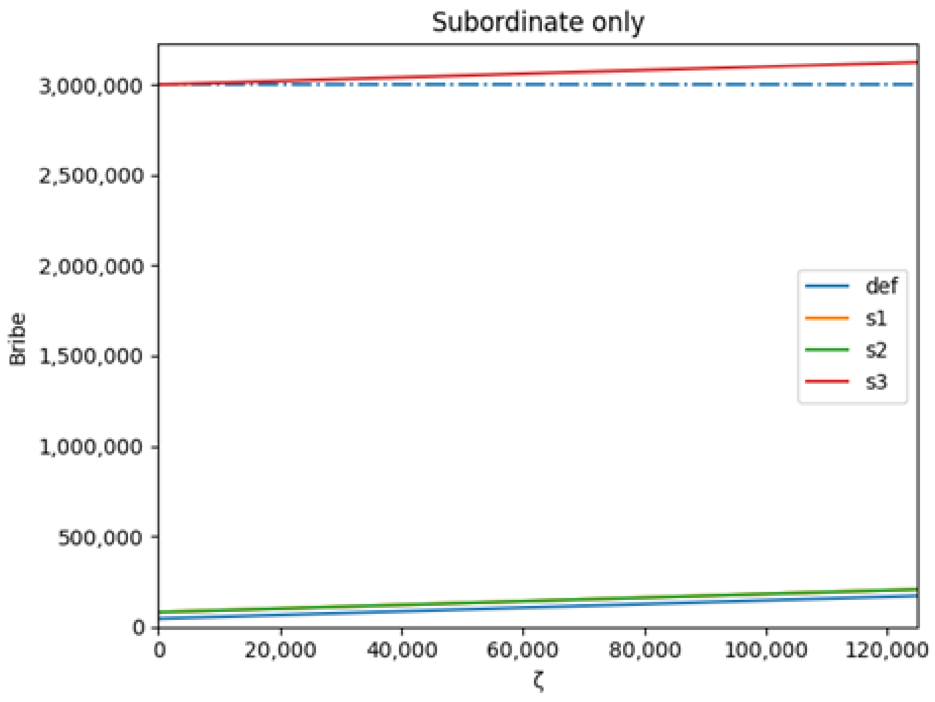


Figure 2.7: The graph of $B_{optT}(\zeta)$ for subordinate and T_s.

The height of the dash-dot line on Figure 2.7 is

$$\frac{S_s}{min[\alpha^0_{2,i};\alpha^0_{1,i}]} = [\frac{125,000}{0.041666667}] = 3,000,000$$

The minimum in the denominator is used to make sure stealing and bribing is not profitable for all officials. The $[x]$ is the integer part of x.

As can be seen from the figures, all possible bribes are above the dash-dot lines of a setting with the point with $\zeta = 1$ being the closest ones to them.

NB: officials with $B^v_{n,i} = B_{optT}(1)$ are playing Nash equilibrium strategies: optimal steals, minimal possible bribes. They are the hardest to discourage from corruption so the corruption minimization should target them.

All obtained settings were simulated $500,000$ times with utilities being averaged. The code execution results are presented in Table 2.6 and Figure

2.8 via charts of "corrupt utility" calculated as

$$CU_X = U_X - W_X \tag{2.14}$$

Due to the assumptions of officials not being able to communicate and not knowing the characteristics of each other and inspector, the averages from the stable solutions are chosen to represent the settings.

Table 2.5: Corruption minimization settings.

Setting	$R(S_{\{1,2\},i})$	$Cu(S_{\{1,2\},i})$	$R(S_{3,i})$	$Cu(S_{3,i})$	$B_{suff-ch}$	B_{suff-b}	B_{suff-s}	T	B_{optT}
Default	40,000.0	5,000.0	75,000.0	11,250.0	131,251.0	86,251.0	45,001.0	-	-
1	60,000.0	20,000.0	875,000.0	429,615.4	1,384,616.4	1,304,615.4	80,000.0	ch	1,384,615.4
2	60,000.0	20,000.0	2,000,000.0	1,000,000.0	3,080,000.0	3,000,000.0	80,000.0	b	3,000,000.0
3	2,000,000.0	1,000,000.0	3,250,000.0	2,500,000.0	8,750,000.0	5,750,000.0	3,000,000.0	s	3,000,000.0

Table 2.6: Change in corrupt utility after corruption minimization.

AVG	def	$s1$	$s2$	$s3$		$def \to s1$	$def \to s2$	$def \to s3$
(3, 0)	143,336.69	0.00	0.00	0.00		-100.00 %	-100.00 %	-100.00 %
(3, 1)	147,691.36	0.00	0.00	0.00		-100.00 %	-100.00 %	-100.00 %
(2, 0)	109,345.65	80,560.62	80,554.08	0.00		-26.32 %	-26.33 %	-100.00 %
(2, 1)	109,236.93	80,548.62	80,554.81	0.00		-26.26 %	-26.26 %	-100.00 %
(1, 0)	96,252.46	78,231.37	78,253.14	0.00		-18.72 %	-18.70 %	-100.00 %
(1, 1)	96,099.14	78,242.55	78,230.66	0.00		-18.58 %	-18.59 %	-100.00 %
Inspector	36,663.69	11,026.42	11,018.90	0.00		-69.93 %	-69.95 %	-100.00 %

The settings changes reduce corruption and it is possible to eliminate the corruption in the model, but the means are extreme.

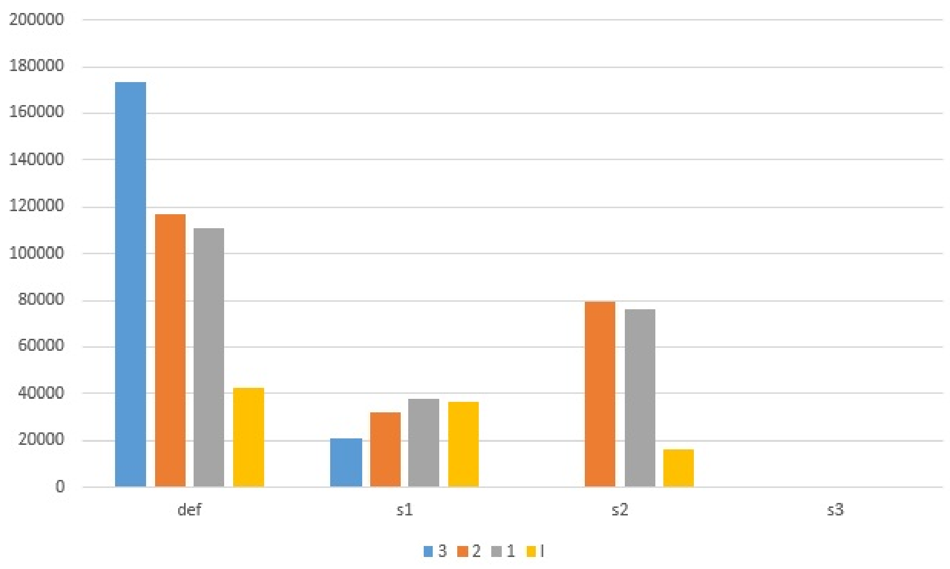


Figure 2.8: Average "corrupt utility" of players after corruption minimization.

2.2.5 Mild Corruption Minimization

The values of settings in Table 2.5 might be considered extreme or impossible to implement in real life, so let us limit the optimal bribe size:

$$B_{optT} \leq S^*_{n,i}$$

With that limitation, we have four possible settings (including default), which we will name *zettings* to avoid confusion:

Table 2.7: Mild corruption minimization zettings.

Zetting	$R(S_{\{1,2\},i})$	$Cu(S_{\{1,2\},i})$	$R(S_{3,i})$	$Cu(S_{3,i})$	$B_{suff-ch}$	B_{suff-b}	B_{suff-s}	T	B_{optT}
Default	40000	5000	75000	11250	131251	86251	45001	-	-
1	70000	35000	270000	124999	500000	395000	105001	ch	500000
2	0	0	300000	199999	500000	500000	1	b	500000
3	85000	39999	250000	125000	500000	375001	125000	s	125000

The settings changes reduce corruption, decrease revenue for $O_{n,i}$ and increase for I, which might also be beneficial since focusing the corrupt money in one place simplifies control. Mild Corruption Minimization is less extreme, effective, but less so than Corruption Minimization.

Table 2.8: Change in utilities after mild corruption minimization.

AVG	def	$z1$	$z3$		$def \to z1$	$def \to z3$	$z1 \to z3$
(3, 0)	143,336.69	69,432.38	69,307.13		-51.56 %	-51.65 %	-0.18 %
(3, 1)	147,691.36	79,857.00	79,864.25		-45.93 %	-45.92 %	0.01 %
(2, 0)	109,345.65	76,485.57	76,168.38		-30.05 %	-30.34 %	-0.41 %
(2, 1)	109,236.93	76,497.06	76,163.28		-29.97 %	-30.28 %	-0.44 %
(1, 0)	96,252.46	75,106.62	74,542.06		-21.97 %	-22.56 %	-0.75 %
(1, 1)	96,099.14	75,127.30	74,531.44		-21.82 %	-22.44 %	-0.79 %
Inspector	36,663.69	70,989.22	71,822.14		93.62 %	95.89 %	1.17 %

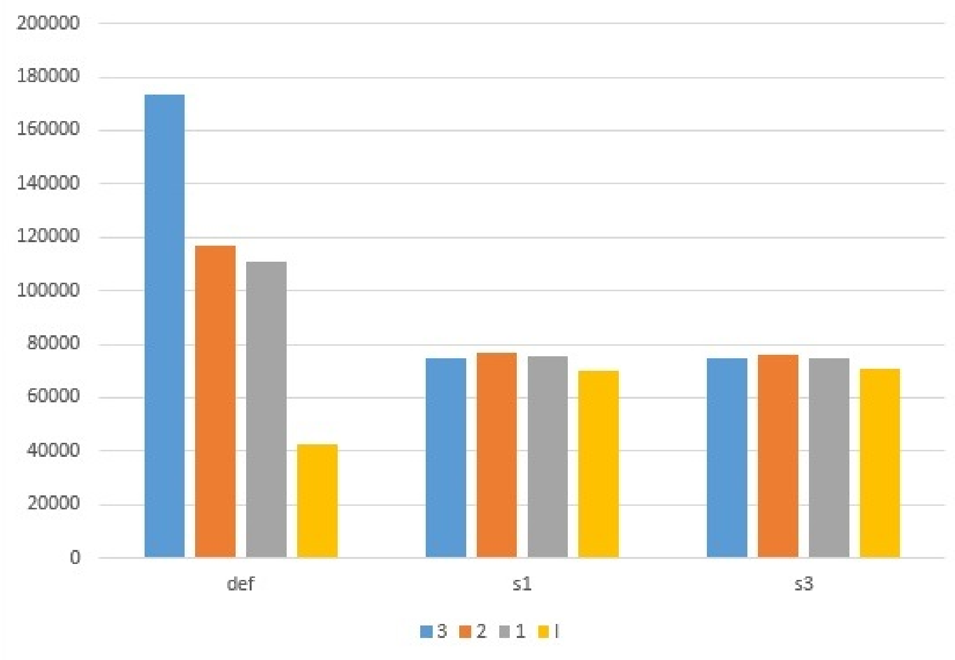


Figure 2.9: Average "corrupt utility" of players after mild corruption minimization.

2.3 Cooperative Extension of the Model

2.3.1 Description

Bosses need some way of protecting themselves from subordinates. One way is to form a coalition of two or more officials in which: members cannot expose each other; members' steals are divided among them according to the stated allocation rule; bribe (in the case when one of the members is inspected) is compiled collectively.

Joining a coalition brings advantages and disadvantages. Advantages are insurance against being exposed; better coordination in terms of stealing amounts (irrelevant in the model, but might be important in real life); more certainty in terms of the sufficient bribe (grand coalition knows exactly how the inspection happened); bigger bribe (thus less chance of being rejected) with less problems conjuring up one for each of the members – at least, potentially. Disadvantages are higher chances of being inspected; higher fines for organized group felonies; allocation might not be favourable for some members.

Not any group of officials can form a coalition. For example, take a pair $\{(1,0),(2,0)\}$. They do not "know" each other – there are no ties connecting them directly, so it must be hard for them to communicate, the former cannot expose the latter because they are not in "superior-subordinate" relationships, forming this coalition is senseless and should not be possible.

We suggest the rule *"any official with direct or indirect connection (path in the hierarchy graph) to another can be in the coalition with them"*. In other words, no disconnected components are allowed in the coalition. For example, coalition $\{(2,0),(3,0),(3,1)\}$ is possible, but $\{(2,0),(3,0),(1,0)\}$ is not. It is possible to build twenty-four different coalitions according to this rule. Coalitions are characterized by:

- set of coalition members, its subsets and their sizes:

$$C = \bigcup_{(n,i)\in C} \{(n,i)\} = \bigcup_{n\in C} C_n, \quad N_C = |C|,$$

$$C_j = \bigcup_{(j,i)\in C} \{(j,i)\}, \quad N_{C,j} = \sum_{(j,i)\in C} 1 = |C_j| \leq N_j,$$

- partial utility of a member according to the rule R (the part official gets from stealing and potentially coalitionally bribing)

$$RU^C_{n,i} = U_{n,i}(S_{n,i}, 0, BC) - W_{n,i},$$

- coalitional actions: members of coalition never expose, always bribe jointly and cannot refrain from stealing (if they do not want it is better for them not to join coalition in the first place)

$$S_{n,i} > 0 \ \& \ A_{n,i} = BC \quad \forall (n,i) \in C,$$

- coalitional stealing

$$S_C = \sum_{(n,i)\in C} S_{n,i},$$

- coalitional bribe

$$B_C,$$

- the chance of inspection

$$\alpha_C = \bigcup_{(n,i)\in C} \alpha^+_{n,i}.$$

This chance can also be portrayed as the vector of probabilities $\alpha_C = (\alpha_{ch}; \alpha_b; \alpha_s)$ since any official but the ultimate subordinate is unsure about the source of inspection (and there is more than one official in the coalition). The same applies to the coalitional bribe: $B_C = (B_{ch}; B_b; B_s)^T$. From that we get:

$$\alpha_C B_C = \alpha_{ch} B_{ch} + \alpha_b B_b + \alpha_s B_s.$$

In the non-cooperative case for boss every term goes into α_{ch}, since they cannot know the source of inspection. Every term goes into α_s for subordinate since there is no other way for them to be inspected but the direct.

If inspector accepts the bribe, coalition loses only it, if he does not, coalition loses the bribe and every coalition member suffers the fine for organized stealing:

$$U^C_{n,i}(A_I) = RU^C_{n,i} - \begin{cases} 0 \; if \; A_I = Acc \\ Fcs(S_C) + Fcb(B_C) \; if \; A_I = Rej \end{cases}$$

where $Fcs(S_C)$ and $Fcb(B_C)$ are fines for coalitional stealing and bribing.

2.3.2 Allocation Rules

Ultimate bosses get all

$$bl : \nexists (n,i) \in C : \; (j,k) \in subs(n,i) \quad \forall (j,k) \in C_{bl}$$

$$BGAU^C_{n,i} = \frac{S_C - \alpha_C B_C}{N_{C,bl}} \quad \forall (n,i) \in C_{bl},$$

$$BGAU^C_{n,i} = 0 \quad \forall (n,i) \notin C_{bl}.$$

Ultimate subordinates get all

$$sl : subs(n,i) = \emptyset \quad \forall (n,i) \in C_{sl}$$

$$SGAU^C_{n,i} = \frac{S_C - \alpha_C B_C}{N_{C,sl}} \quad \forall (n,i) \in C_{sl,}$$

$$SGAU^C_{n,i} = 0 \quad \forall (n,i) \notin C_{sl}.$$

Equity

$$EQU^C_{n,i} = \frac{S_C - \alpha_C B_C}{N_C} \quad \forall (n,i) \in C.$$

Only equally shared bribe

$$ESBU^C_{n,i} = S_{n,i} - \frac{\alpha_C B_C}{N_C} \quad \forall (n,i) \in C.$$

Only proportionally shared bribe

$$PSBU^C_{n,i} = S_{n,i} - \frac{\alpha_C B_{C,n}}{N_{C,n}} \quad \forall (n,i) \in C$$

$$B_C = \sum_{n \in C} B_{C,n}$$

$$B_{C,n} = \gamma_n B_C \quad \gamma_n \in [0,1] \quad \sum_{n \in C} \gamma_n = 1$$

$$k > n : \quad B_{C,k} \geq B_{C,n} \geq 0.$$

Equally shared bribe plus bonus to subordinate

$$ESBBSU^C_{n,i} = S_{n,i} - \frac{\alpha_C B_C}{N_C} + \begin{cases} -|C \cap subs(n,i)| \cdot BS_{n,i} \ if \ n = bl \\ BS_{boss(n)} - |C \cap subs(n,i)| \cdot BS_{n,i} \ if \ n \neq bl, sl \\ BS_{boss(n)} \ if \ n = sl \end{cases}$$

$$\forall (n,i) \in C.$$

Proportionally shared bribe plus bonus to subordinate

$$PSBBSU^C_{n,i} = S_{n,i} - \frac{\alpha_C B_{C,n}}{N_{C,n}} + \begin{cases} -|C \cap subs(n,i)| \cdot BS_{n,i} \ if \ n = bl \\ BS_{boss(n)} - |C \cap subs(n,i)| \cdot BS_{n,i} \ if \ n \neq bl, sl \\ BS_{boss(n)} \ if \ n = sl \end{cases}$$

$$\forall (n,i) \in C$$

$$B_C = \sum_{n \in C} B_{C,n}$$

$$B_{C,n} = \gamma_n B_C \quad \gamma_n \in [0,1] \quad \sum_{n \in C} \gamma_n = 1$$

$$k > n : \quad B_{C,k} \geq B_{C,n} \geq 0.$$

2.3.3 Stability

The payoff is called *individually stable* if it is in the Imputation set

$I(v) = \{X \in R^{N_C} \mid X(C) = v(C), \quad X_{n,i} \geq v(\{(n,i)\})\ \forall (n,i) \in C\}$,

i.e. it is not worse for individual to join the coalition, than to be alone.

The payoff is called *coalitionally stable* if it is in the Core

$C(v) = \{X \in R^{N_C} \mid X(C) = v(C), \quad X(S) \geq v(S)\ \forall S \subset C\}$,

i.e. no subgroup of players has an incentive to deviate.

Since officials on one level have the same characteristics, we can simplify the analysis by categorizing the twenty-four derived coalitions into fourteen coalition types:

Subordinate-subordinate left

$$SSL = \{\{(2,0),(2,1)\}\}.$$

Subordinate-subordinate right

$$SSR = \{\{(1,0),(1,1)\}\}.$$

Boss-boss $BB = \{\{(3,0),(3,1)\}\}$.

Boss-subordinate left

$$1B1SL = \{\{(3,0),(2,0)\},\{(3,0),(2,1)\}\}.$$

Boss-subordinate right

$$1B1SR = \{\{(1,0),(3,1)\},\{(1,1),(3,1)\}\}.$$

Boss-boss-subordinate left

$$BB1SL = \{\{(2,0),(3,0),(3,1)\},\{(2,1),(3,0),(3,1)\}\}.$$

Boss-boss-subordinate right

$$BB1SR = \{\{(1,0),(3,1),(3,0)\},\{(1,1),(3,1),(3,0)\}\}.$$

Boss-2-subordinates left

$$1B2SL = \{\{(3,0),(2,0),(2,1)\}\}.$$

Boss-2-subordinates right

$$1B2SR = \{\{(3,1),(1,0),(1,1)\}\}.$$

2-subordinates-boss-boss left

$$2SBBL = \{\{(2,0),(2,1),(3,0),(3,1)\}\}.$$

2-subordinates-boss-boss right

$$2SBBR = \{\{(1,0),(1,1),(3,1),(3,0)\}\}.$$

Subordinate-boss-boss-subordinate

$$1SBB1S = \{\{(2,0),(3,0),(3,1),(1,0)\},\{(2,0),(3,0),(3,1),(1,1)\},$$
$$\{(2,1),(3,0),(3,1),(1,0)\},\{(2,1),(3,0),(3,1),(1,1)\}\}.$$

2-subordinates-boss-boss-subordinate left

$$2SBB1SL =$$
$$\{\{(2,0),(2,1),(3,0),(3,1),(1,0)\},\{(2,0),(2,1),(3,0),(3,1),(1,1)\}\}.$$

2-subordinates-boss-boss-subordinate right

$$2SBB1SR =$$
$$\{\{(2,0),(3,0),(3,1),(1,0),(1,1)\},\{(2,1),(3,0),(3,1),(1,0),(1,1)\}\}.$$

Grand coalition

$$GC = \{\{(2,0),(2,1),(3,0),(3,1),(1,0),(1,1)\}\}.$$

2.3.4 Analysis of the Rules

Assumptions:

1. Default setting.
2. $S^*_{1,i} = S^*_{2,i} = S_s \quad S^*_{3,i} = S_b$
3. If official is indifferent between being in coalition and not being in one, they choose not being.

From **Assumption 1** we get

$$\sum_{(l,j)\in T} [R(S_{l,j}) + Cu(S_{l,j})] < B^*_{n,i} < \frac{S^*_{n,i}}{\alpha^+_{n,i}} \quad \forall T, \tag{2.15}$$

and that gives us

$$S^*_{n,i} > 0 \quad \forall (n,i) \in H \ \rightarrow \ S_s > 0, \tag{2.16}$$

$$S_s - \alpha^+_{n,i} B_s = S_s - \frac{\alpha^{eff}_n}{2} B_s > 0 \quad n = 1,2 \ \ i = 0,1 \tag{2.17}$$

$$S_b - \alpha^+_{3,j} B_{ch} = S_b - (\frac{\alpha_3}{2} + \alpha^{eff}_k) B_{ch} > 0 \quad (j,k) = (1,1),(2,0) \tag{2.18}$$

$$B_{ch} > B_b > B_s \tag{2.19}$$

For Imputation the test is against (2.16) and (2.18), for Coalition – against any other proper subcoalition.

$$i = 0,1 \text{ unless stated otherwise}$$

Ultimate bosses get all

$$BGAU^{SSL,SSR}_{n,i} = \frac{2S_s - \alpha^{eff}_n B_s}{2} = S_s - \frac{\alpha^{eff}_n}{2} B_s < S_s \quad n = \begin{cases} 1 \ if \ SSR \\ 2 \ if \ SSL \end{cases}$$

$$BGAU^{BB}_{3,j} = S_b - \frac{\alpha_3 + \alpha^{eff}_2 + \alpha^{eff}_1}{2} B_{ch} \gtreqless S_b - (\frac{\alpha_3}{2} + \alpha^{eff}_k) B_{ch} \quad (j,k) = (1,1),(2,0)$$

$$-\frac{\alpha_3 + \alpha^{eff}_2 + \alpha^{eff}_1}{2} \gtreqless -(\frac{\alpha_3}{2} + \alpha^{eff}_k) :$$

$$-\frac{\alpha^{eff}_2 + \alpha^{eff}_1}{2} \gtreqless -\alpha^{eff}_k :$$

$$\begin{array}{c|c} -\frac{\alpha_2^{eff}+\alpha_1^{eff}}{2} \overset{\geq}{<} -\alpha_2^{eff} & -\frac{\alpha_2^{eff}+\alpha_1^{eff}}{2} \overset{\geq}{<} -\alpha_1^{eff} \\ -\frac{\alpha_1^{eff}}{2} \overset{\geq}{<} -\frac{\alpha_2^{eff}}{2} & -\frac{\alpha_2^{eff}}{2} \overset{\geq}{<} -\frac{\alpha_1^{eff}}{2} \end{array}$$

The inequalities are mutually exclusive: being in coalition is only weakly profitable for both officials (they are breaking even) if inequalities turn into equalities, but due to the **Assumption 3** in such case they do not participate, so $BGAU - BB$ pair is neither I, nor C.

If we calculate the actual values deriving from (2.3) formulas

$$\alpha_2^{eff} = (1-\alpha_3)\alpha_2$$
$$\alpha_1^{eff} = (1-\alpha_3)(1-\alpha_2)\alpha_1$$

and values $\alpha_2 = 0,208$, $\alpha_1 = 0,250$, we get

$$\begin{array}{c|c} -0.395833333 < -0.208333333 & -0.208333333 > -0.395833333 \end{array}$$

It means that being in coalition is profitable for $O_{3,1}$, but not profitable for $O_{3,0}$, thus it is indeed neither I, nor C.

$$BGAU_{n,i}^{1B1SL,1B1SR} = 0 < S_s \quad n = \begin{cases} 1 \; if \; 1B1SR \\ 2 \; if \; 1B1SL \end{cases}$$

Not I and not C since there is a possible deviation for subordinate – leave the coalition to earn more by exposing the boss. The same applies to other coalition types.

Ultimate subordinates get all

Reasoning for SSR, SSL and BB is analogous to the respective one in BGA.

$$SGAU_{n,i}^{1B1SL,1B1SR} = 0 < S_b - (\frac{\alpha_3}{2} + \alpha_k^{eff})B_{ch} \quad n = \begin{cases} 1 \; if \; 1B1SR \\ 2 \; if \; 1B1SL \end{cases}$$

Not I and not C since there is a possible deviation for boss – leave the coalition to earn more by paying the bribe. The same applies to other coalition types.

Equity

Reasoning for SSR, SSL and BB is analogous to the respective one in BGA.

$$EQU_{n,i}^{1B1SL,1B1SR} = \frac{S_b + S_s - (\frac{\alpha_3+\alpha_j^{eff}}{2} B_{ch} + \frac{\alpha_j^{eff}}{2} B_s)}{2} \quad j = \begin{cases} 1 \; if \; 1B1SR \\ 2 \; if \; 1B1SL \end{cases}$$

$$EQU_{n,i}^{BB1SL,BB1SR} = \frac{2S_b + S_s - [(\alpha_3 + \frac{\alpha_j^{eff}}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_j^{eff}}{2} B_s]}{3}$$

$$j = \begin{cases} 1 \; if \; BB1SR \\ 2 \; if \; BB1SL \end{cases}$$

$$EQU_{n,i}^{1B2SL,1B2SR} = \frac{S_b + 2S_s - (\frac{\alpha_3}{2} B_b + \alpha_j^{eff} B_s)}{3} \quad j = \begin{cases} 1 \; if \; 1B2SR \\ 2 \; if \; 1B2SL \end{cases}$$

$$EQU_{n,i}^{2SBBL,2SBBR} = \frac{2S_b + 2S_s - [(\frac{\alpha_3}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_3}{2} B_b + \alpha_j^{eff} B_s)]}{4}$$

$$j = \begin{cases} 1 \; if \; 2SBBR \\ 2 \; if \; 2SBBL \end{cases}$$

$$EQU_{n,i}^{1SBB1S} = \frac{2S_b + 2S_s - [(\alpha_3 + \frac{\alpha_2^{eff}}{2} + \frac{\alpha_1^{eff}}{2}) B_{ch} + (\frac{\alpha_2^{eff}}{2} + \frac{\alpha_1^{eff}}{2})) B_s]}{4}$$

$$EQU_{n,i}^{2SBB1SL,2SBB1SR} = \frac{2S_b + 3S_s - [\frac{\alpha_3+\alpha_{3-j}^{eff}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2}) B_s]}{5}$$

$$j = \begin{cases} 1 \; if \; 2SBB1SR \\ 2 \; if \; 2SBB1SL \end{cases}$$

$$EQU^{GC}_{n,i} = \frac{2S_b + 4S_s - [\alpha_3 B_b + (\alpha_2^{eff} + \alpha_1^{eff})B_s]}{6}$$

The analysis of the remaining rules can be found in Appendix B.

Subordinate-stable

In order to make a coalition stable (since in the model there is no representation of punishment for exposing which might happen in the real life) bonus and shared bribe part must be chosen to cover subordinate's part of the bribe (or proportion of shared bribe must be zero):

$$S_s - \frac{\alpha_C B_{C,j}}{N_{C,j}} + BS_{3,j\%2} > S_s \;\rightarrow\; BS_{3,j\%2} > \frac{\alpha_C B_{C,j}}{N_{C,j}}$$

$$BS_{3,j\%2}(\xi) = \frac{\alpha_C B_{C,j}}{N_{C,j}} + \xi$$

Following that, PSB, $ESBBS$, $PSBBS$ effectively become

$$SSU^C_{n,i} = S_{n,i} - \begin{cases} \frac{\alpha_C B_C + |C \cap \bigcup_{(n,i)\notin C_{bl}} \{(n,i)\}| \cdot \xi}{N_{C,bl}} \; if \; n = bl \\ -\xi \; otherwise \end{cases} \quad \forall (n,i) \in C$$

SSL, SSR, BB: there is only one level; reasoning is identical to the respective BGA.

$$C = \{1B1SL, 1B1SR, BB1SL, BB1SR, 1B2SL, 1B2SR, 2SBBL,$$
$$2SBBR, 2SBBL, 2SBBR, 2SBB1SL, 2SBB1SR, GC\}$$
$$SSU^C_{j,i} = S_s + \xi \quad j \neq 3$$

$$SSU_{3,j\%2}^{1B1SL,1B1SR} = S_b - [\frac{\alpha_3 + \alpha_j^{eff}}{2} B_{ch} + \frac{\alpha_j^{eff}}{2} B_s + \xi] \quad j = \begin{cases} 1 \; if \; 1B2SR \\ 2 \; if \; 1B2SL \end{cases}$$

$$SSU_{3,k}^{BB1SL,BB1SR} = S_b - \frac{(\alpha_3 + \frac{\alpha_j^{eff}}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_j^{eff}}{2} B_s + \xi}{2} \quad j = \begin{cases} 1 \; if \; BB1SR \\ 2 \; if \; BB1SL \end{cases}$$

$$SSU_{3,k}^{1B2SL,1B2SR} = S_b - [\frac{\alpha_3}{2} B_b + \alpha_j^{eff} B_s + 2\xi] \quad j = \begin{cases} 1 \; if \; 1B2SR \\ 2 \; if \; 1B2SL \end{cases}$$

$$SSU_{3,k}^{1SBB1S} = S_b - \frac{(\alpha_3 + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2}) B_{ch} + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2} B_s + 2\xi}{2}$$

$$SSU_{3,k}^{2SBB1SL,2SBB1SR} = S_b - \frac{\frac{\alpha_3 + \alpha_{3-j}^{eff}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2}) B_s + 3\xi}{2}$$

$$j = \begin{cases} 1 \; if \; 1B2SR \\ 2 \; if \; 1B2SL \end{cases}$$

$$SSU_{3,k}^{GC} = S_b - \frac{\alpha_3 B_b + (\alpha_2^{eff} + \alpha_1^{eff}) B_s + 4\xi}{2}$$

Since bosses maximize their profits, they minimize expenses by choosing the minimal possible bonus. By the reasoning identical to the sensitivity analysis of corruption minimization we take $\xi = 1$ (since it can be seen as a bribe to the subordinate) and any conclusions made for it will be valid for any $\xi > 1$, which it will certainly be in the real world according to the ultimatum bargaining games studies [15, 16].

From the analysis we have:

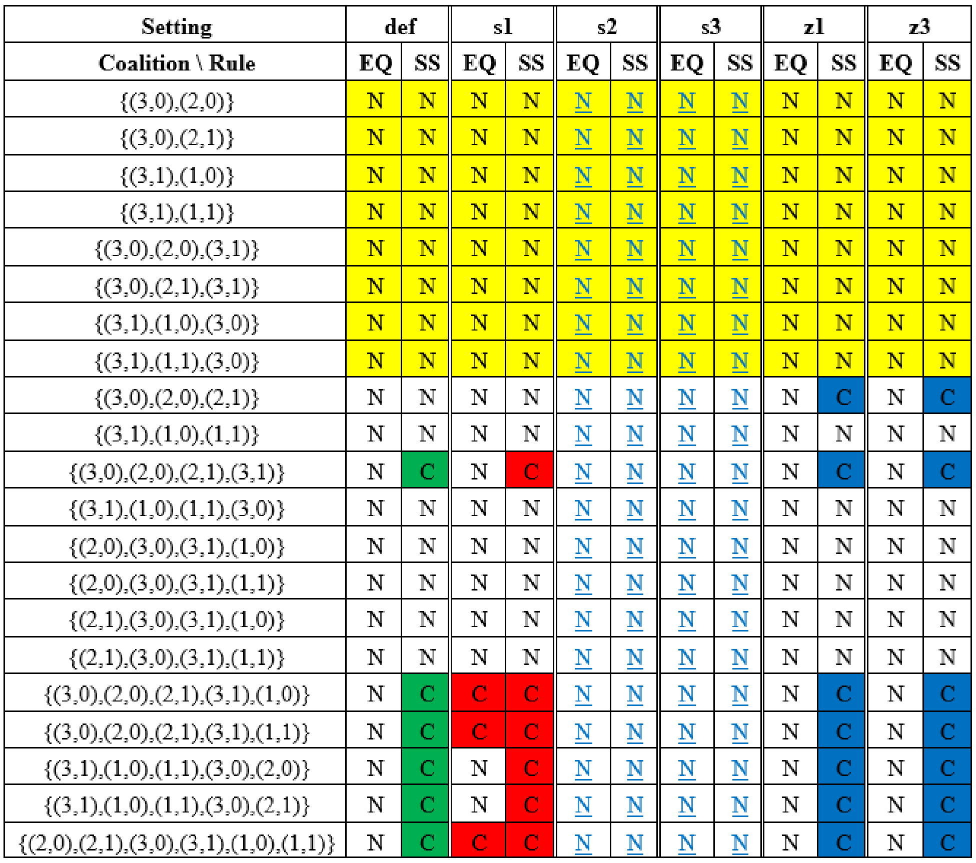
1. SSL, SSR, BB coalition types cannot provide either individual or coalitional stability under any rule.
2. Rules BGA, SGA, ESB cannot provide either individual or coalitional stability for any coalition type.
3. PSB, $ESBBS$, $PSBBS$ can provide stability only if they are transformed into SS rule.
4. The only coalition-rule pairs that are not analytically proven to be unstable are in the Table 2.9.

Table 2.9: "Testable" coalition-rule pairs.

Rule	EQ		SS	
Coalition	I	C	I	C
1B1SL	MB	MB	MB	MB
1B1SR	MB	MB	MB	MB
BB1SL	MB	MB	MB	MB
BB1SR	MB	MB	MB	MB
1B2SL	MB	MB	MB	MB
1B2SR	MB	MB	MB	MB
2SBBL	MB	MB	MB	MB
2SBBR	MB	MB	MB	MB
1SBB1S	MB	MB	MB	MB
2SBB1SL	MB	MB	MB	MB
2SBB1SR	MB	MB	MB	MB
GC	MB	MB	MB	MB

2.3.5 Simulation Results

The both models will be compared with the minimal necessary bribe given: we will compare only the best possible cases because in the case of not sufficient bribe the non-cooperative model officials have an advantage of defaulting to the "not stealing and not bribing strategy (None_NB)" while members of coalition do not. It is also quite computation-heavy. The simulation was run 500,000 times for each coalition-rule pair for the default and all anti-corruption settings (normal and mild) with $\xi = 1$. Its code can be found in Appendix C.



Setting	def		s1		s2		s3		z1		z3	
Coalition \ Rule	**EQ**	**SS**	**EQ**	**SS**	**EQ**	**SS**	**EQ**	**SS**	**EQ**	**SS**	**EQ**	**SS**
{(3,0),(2,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,0),(2,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,1),(1,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,1),(1,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,0),(2,0),(3,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,0),(2,1),(3,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,1),(1,0),(3,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,1),(1,1),(3,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,0),(2,0),(2,1)}	N	N	N	N	N	N	N	N	N	C	N	C
{(3,1),(1,0),(1,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,0),(2,0),(2,1),(3,1)}	N	C	N	C	N	N	N	N	N	C	N	C
{(3,1),(1,0),(1,1),(3,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(2,0),(3,0),(3,1),(1,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(2,0),(3,0),(3,1),(1,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(2,1),(3,0),(3,1),(1,0)}	N	N	N	N	N	N	N	N	N	N	N	N
{(2,1),(3,0),(3,1),(1,1)}	N	N	N	N	N	N	N	N	N	N	N	N
{(3,0),(2,0),(2,1),(3,1),(1,0)}	N	C	C	C	N	N	N	N	N	C	N	C
{(3,0),(2,0),(2,1),(3,1),(1,1)}	N	C	C	C	N	N	N	N	N	C	N	C
{(3,1),(1,0),(1,1),(3,0),(2,0)}	N	C	N	C	N	N	N	N	N	C	N	C
{(3,1),(1,0),(1,1),(3,0),(2,1)}	N	C	N	C	N	N	N	N	N	C	N	C
{(2,0),(2,1),(3,0),(3,1),(1,0),(1,1)}	N	C	C	C	N	N	N	N	N	C	N	C

Figure 2.10: Simulation results analysis.

In the Figure 2.10 N means "not stable", C means "coalitionally stable (inside Core)". Conclusions from the analysis:

1. 1B1SL, 1B1SR, BB1SL and BB1SR types of coalitions do not provide stable divisions under any setting (yellow fill).

2. The 2SBBL, 2SBB1SL, 2SBB1SR and GC with SS rule coalition-rule pairs are coalitionally stable in the default setting (green fill).

3. Under setting s1 all pairs from point 2 plus 2SBB1SL and GC with EQ rule are coalitionally stable (red fill). The coalitions are effective in that

case because they are less affected by the change in B_{ch} than individual players.

4. No rule provides a stable division under settings s2 and s3 (underlined blue font) – the corruption minimization settings work even in case of cooperation because they make direct inspections impossible to bribe profitably so even the extra information does not help.

5. Stability results under similar zettings z1 and z2 are similar: only SS provides stable outcomes in 1B2SL, 2SBBL, 2SBB1SL, 2SBB1SR and GC (blue fill).

2.3.6 Myerson Value

Different approach to disconnected components is provided by the *Myerson value.* It is an adaptation of *Shapley value* to restricted communication graph stated in Caulier et al. [14] as

$$v^g(S) = \sum_{C \in S|_g} v(C)$$

$S|_g$ *denotes the set of connected coalitions of* g*, i.e., those sets* C *which are maximal subcoalitions of* S *such that all pairs of players in* C *are connected. If* S *is connected, then its players can communicate and therefore they obtain their initial payoff* $v(S)$*. Otherwise, players in coalition* S *can only communicate among members of the same connected component. As there is no possible communication between different components, players in* S *can only get the sum of payoffs obtained by each component independently.*

$$M_i(v,g) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-1-|S|)!}{n!}(v^g(S \cup \{i\}) - v^g(S))$$

For this model the changed and simplified (since there is only one g studied) notation is

$$M_{n,i}(v) = \sum_{S \subseteq H \setminus \{(n,i)\}} \frac{|S|!(|H|-1-|S|)!}{|H|!}[v(S \cup \{(n,i)\}) - v(S)], \quad (2.20)$$

where H – hierarchy, set of all officials.

Two types are studied: the Myerson for the original (formula (2.20)) and modified versions of the game (formula (2.21)) where the assumption of "playing nice" is made: subordinates choose to bribe instead of exposing turning $v(C)$ into $v^*(C)$. Let us call the latter *Theirson value.*

$$T_{n,i}(v) = M_{n,i}(v^*) = \sum_{S \subseteq H \setminus \{(n,i)\}} \frac{|S|!(|H|-1-|S|)!}{|H|!}[v^*(S \cup \{(n,i)\}) - v^*(S)] \quad (2.21)$$

To calculate these values, we need to write values of all "whole" (fully formable) coalitions:

$$R = \{(1,0)\}; \{(1,1)\}$$

$$\begin{aligned} v(R) &= S_s \\ v^*(R) = S_s - (1-\alpha_3)(1-\alpha_2)\frac{\alpha_1}{2}B_s &= \\ &= S_s - \frac{\alpha_1^{eff}}{2}B_s \end{aligned} \quad (2.22)$$

$$L = \{(2,0)\}; \{(2,1)\}$$

$$\begin{aligned} v(L) &= S_s \\ v^*(L) &= S_s - (1-\alpha_3)\frac{\alpha_2}{2}B_s = \\ &= S_s - \frac{\alpha_2^{eff}}{2}B_s \end{aligned} \tag{2.23}$$

$$\begin{aligned} v(\{(3,0)\}) &= S_b - (\frac{\alpha_3}{2} + \alpha_2^{eff})B_{ch} \\ v^*(\{(3,0)\}) &= S_b - \frac{\alpha_3}{2}B_{ch} \end{aligned} \tag{2.24}$$

$$\begin{aligned} v(\{(3,1)\}) &= S_b - (\frac{\alpha_3}{2} + \alpha_1^{eff})B_{ch} \\ v^*(\{(3,1)\}) &= S_b - \frac{\alpha_3}{2}B_{ch} \end{aligned} \tag{2.25}$$

$$\begin{aligned} v(SSR) &= 2S_s - \alpha_1^{eff}B_s \\ v^*(SSR) &= v(SSR) \end{aligned} \tag{2.26}$$

$$\begin{aligned} v(SSL) &= 2S_s - \alpha_2^{eff}B_s \\ v^*(SSL) &= v(SSL) \end{aligned} \tag{2.27}$$

$$\begin{aligned} v(BB) &= 2S_b - [\alpha_3 + \alpha_2^{eff} + \alpha_1^{eff}]B_{ch} \\ v^*(BB) &= 2S_b - \alpha_3 B_b \end{aligned} \tag{2.28}$$

$$\begin{aligned} v(1B1SL) &= S_b + S_s - [(\frac{\alpha_3}{2} + \frac{\alpha_2^{eff}}{2})B_{ch} + \frac{\alpha_2^{eff}}{2}B_s] \\ v^*(1B1SL) &= S_b + S_s - [\frac{\alpha_3}{2}B_b + \frac{\alpha_2^{eff}}{2}B_s] \end{aligned} \tag{2.29}$$

$$
\begin{aligned}
v(1B1SR) &= S_b + S_s - [(\frac{\alpha_3}{2} + \frac{\alpha_1^{eff}}{2})B_{ch} + \frac{\alpha_1^{eff}}{2}B_s] \\
v^*(1B1SR) &= S_b + S_s - [\frac{\alpha_3}{2}B_b + \frac{\alpha_1^{eff}}{2}B_s]
\end{aligned}
\tag{2.30}
$$

$$
\begin{aligned}
v(BB1SL) &= 2S_b + S_s - [(\alpha_3 + \frac{\alpha_2^{eff}}{2} + \alpha_1^{eff})B_{ch} + \frac{\alpha_2^{eff}}{2}B_s] \\
v^*(BB1SL) &= 2S_b + S_s - [\alpha_3 B_b + \frac{\alpha_2^{eff}}{2}B_s]
\end{aligned}
\tag{2.31}
$$

$$
\begin{aligned}
v(BB1SR) &= 2S_b + S_s - [(\alpha_3 + \alpha_2^{eff} + \frac{\alpha_1^{eff}}{2})B_{ch} + \frac{\alpha_1^{eff}}{2}B_s] \\
v^*(BB1SR) &= 2S_b + S_s - [\alpha_3 B_b + \frac{\alpha_1^{eff}}{2}B_s]
\end{aligned}
\tag{2.32}
$$

$$
\begin{aligned}
v(1B2SL) &= S_b + 2S_s - [\frac{\alpha_3}{2}B_b + \alpha_2^{eff}B_s] \\
v^*(1B2SL) &= v(1B2SL)
\end{aligned}
\tag{2.33}
$$

$$
\begin{aligned}
v(1B2SR) &= S_b + 2S_s - [\frac{\alpha_3}{2}B_b + \alpha_1^{eff}B_s] \\
v^*(1B2SR) &= v(1B2SR)
\end{aligned}
\tag{2.34}
$$

$$
\begin{aligned}
v(2SBBL) &= 2S_b + 2S_s - [(\frac{\alpha_3}{2} + \alpha_1^{eff})B_{ch} + \frac{\alpha_3}{2}B_b + \alpha_2^{eff}B_s \\
v^*(2SBBL) &= 2S_b + 2S_s - [\alpha_3 B_b + \alpha_2^{eff}B_s]
\end{aligned}
\tag{2.35}
$$

$$
\begin{aligned}
v(2SBBR) &= 2S_b + 2S_s - [(\frac{\alpha_3}{2} + \alpha_2^{eff})B_{ch} + \frac{\alpha_3}{2}B_b + \alpha_1^{eff}B_s \\
v^*(2SBBR) &= 2S_b + 2S_s - [\alpha_3 B_b + \alpha_1^{eff}B_s]
\end{aligned}
\tag{2.36}
$$

$$\begin{aligned} v(1SBB1S) &= 2S_b + 2S_s - [(\alpha_3 + \frac{\alpha_2^{eff}}{2} + \frac{\alpha_1^{eff}}{2})B_{ch} + (\frac{\alpha_2^{eff}}{2} + \frac{\alpha_1^{eff}}{2})B_s] \\ v^*(1SBB1S) &= 2S_b + 2S_s - [\alpha_3 B_b + (\frac{\alpha_2^{eff}}{2} + \frac{\alpha_1^{eff}}{2})B_s] \end{aligned} \tag{2.37}$$

$$\begin{aligned} v(2SBB1SR) &= 2S_b + 3S_s - [(\frac{\alpha_3}{2} + \frac{\alpha_1^{eff}}{2})B_{ch} + \frac{\alpha_3}{2}B_b + (\alpha_2^{eff} + \frac{\alpha_1^{eff}}{2})B_s] \\ v^*(2SBB1SR) &= 2S_b + 3S_s - [\alpha_3 B_b + (\alpha_2^{eff} + \frac{\alpha_1^{eff}}{2})B_s] \end{aligned} \tag{2.38}$$

$$\begin{aligned} v(2SBB1SL) &= 2S_b + 3S_s - [(\frac{\alpha_3}{2} + \frac{\alpha_2^{eff}}{2})B_{ch} + \frac{\alpha_3}{2}B_b + (\frac{\alpha_2^{eff}}{2} + \alpha_1^{eff})B_s] \\ v^*(2SBB1SL) &= 2S_b + 3S_s - [\alpha_3 B_b + (\frac{\alpha_2^{eff}}{2} + \alpha_1^{eff})B_s] \end{aligned} \tag{2.39}$$

$$\begin{aligned} v(GC) &= 2S_b + 4S_s - [\alpha_3 B_b + (\alpha_2^{eff} + \alpha_1^{eff})B_s]; \\ v^*(GC) &= v(GC) \end{aligned} \tag{2.40}$$

The formulas for values of all 63 coalitions can be found in the Appendix D. The code for calculation – in the Appendix E.

Table 2.10: Myerson/Theirson analysis for the corruption minimization settings.

Setting	def		s1		s2		s3	
O	My > BST	Th > BST	My > BST	Th > BST	My > BST	Th > BST	My > BST	Th > BST
(3, 0)	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
(3, 1)	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
(2, 0)	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
(2, 1)	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
(1, 0)	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
(1, 1)	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
Conv_fail	274	1044	306	982	308	888	348	1028

Table 2.11: Myerson/Theirson analysis for the mild corruption minimization zettings.

Zetting	z1		z3	
O	My > BST	Th > BST	My > BST	Th > BST
(3, 0)	TRUE	TRUE	TRUE	TRUE
(3, 1)	FALSE	TRUE	FALSE	TRUE
(2, 0)	TRUE	FALSE	TRUE	FALSE
(2, 1)	TRUE	FALSE	TRUE	FALSE
(1, 0)	TRUE	FALSE	TRUE	FALSE
(1, 1)	TRUE	FALSE	TRUE	FALSE
Conv_fail	344	856	290	910

The results of analysis for all the settings are presented in Tables 2.10 and 2.11. "BST" is the average utility of the strategies providing the biggest utilities for respective settings. The column "My > Th" was deleted from the results due to always being TRUE for bosses and FALSE for subordinates. Conclusions from the analysis are as follows:

1. Neither Myerson nor Theirson game is convex: out of all possible $63 \cdot 62 = 3906$ coalition pairs S, T the number of pairs for which the condition $v(S) + v(T) \leq v(S \cup T) + v(S \cap T)$ does not hold is in the Conv_fail row.

2. Theirson always undervalues subordinates compared to Myerson, which is to be expected since in the former they "give up" their ability to expose.
3. Neither rule provides a stable allocation for the default setting.
4. Myerson rule provides a stable allocation in the settings s1, z1 and z3 (in the last two the differences in the only "FALSE" are 0.69% and 0.55% respectively).
5. Theirson rule does not provide a stable allocation in any setting: it satisfies either bosses only (s1, s2, z1, z3) or no one (def, s3).

2.4 Limitations and Further Work

In this work no analysis of the effect of parameters κ and θ was carried out. Doing so or measuring them in an organization or a country might be a prospect. Real-life experiments (post-hoc or real-time) also might also be useful for tuning the model.

Fine functions' effect analysis is another prospect. It was not done since in the current model an official would rather their bribe were not rejected to avoid losing a part of steal, so it might not have been very informative.

Studying a larger hierarchy might introduce new effects and open the possibility of the coalitional wars: multiple corrupt coalitions exposing each other (or bribing inspector to fix the evidence such that the other coalition is fined). It was not done due to the limited computing resources.

Introducing the mechanism of repeated game into the model is another interesting prospect. The one who does so will have to solve the problem of orphans in a hierarchy (if the uncovered corrupt official is fired, their subordinates become *bossless*) and players' different values of the future. It also creates opportunity for punishment strategies (bosses finding out who exposed them and taking revenge in the next iteration), which will surely change the equilibrium situation. It was not done due to the unwillingness to add yet another layer of complexity to the model.

Studying the effect of imperfect technology of inspection might be another interesting prospect, which was not yet done for the aforementioned reason.

Change of inspection direction can be done quite easily in code simulation but rather hard in the formulae. The current top-down approach is based on the inspection works [9, 10, 11] and the idea of "following the money": when the inspection is checking the organization that received the money, it may try to recreate its path to find the exact stage where everything went awry. On the other hand, bottom-up approach can be seen as a "reaction to malfunction": something happened and the inspection is reacting to it. The inspector using the first (proactive) approach deals with the corruption before something happened and thus is easier to bribe, while in the second (reactive) case something has already happened and it is much harder to cover up.

2.5 Approbation

The work was presented at Control Processes and Stability (CPS'20) [17], MCTaIA-2020 [18] and was published in their respective proceedings. The study was also presented at the Fourteenth International Conference on Game Theory and Management (GTM2020) and Control Processes and Stability (CPS'21) and is being published at the moment.

3 Conclusion

The study of the literature shows most researches do not take hierarchical relations of players into account and analyze "simple" games between two-three agents. The similar claim is made by Gorbaneva et al. [10].

The difference between the this study and hierarchical studies [9, 10, 11] lies in the construction of hierarchy: in the works mentioned above hierarchies are of "administration-inspector-client" type with no differentiation in the last class, while this work focuses on the "superior-subordinate" type (which provides a feature of subordinate having the ability to expose the bigger stealer, for example, their superior) with inspector being outside the hierarchy. Another difference is the development of cooperative element. The semblance can be found in absence of corruption on the highest level of the hierarchy and the very use of hierarchy.

The model of hierarchical corruption was built. It consists of two stages: at the first stage each official in the hierarchy decides how much money they embezzle, at the second stage inspector investigates the stealing and the inspected official chooses the action (bribe, not bribe or expose) and the size of bribe.

The notion of *Nash-like* equilibrium as the situation in which officials optimize under uncertainty about inspector's payoffs was proposed.

The particular case with two levels and six officials was built and solved via computer simulation. The result is an equilibrium in which each inspected subordinate (official from level 1 or 2) exposes their boss who then gives the inspector sufficient bribe and each inspected official from level 3 gives sufficient bribe. This equilibrium situation is pessimistic because corruption is not punished, but causes even greater corruption.

The inequalities connecting the decision-making of inspector and official in general form were suggested and used to find the corruption minimization settings in the example under consideration. Their simulations were carried out: two settings decrease corruption and one eradicates it. Mild cooperation minimization *zettings* with a sufficient bribe being capped by the steal were also suggested and simulated.

The cooperative element was introduced; rules for forming coalition and allocating the steal and bribe were suggested; criteria for stability were described (being inside Imputation set for individual stability and inside the Core for coalitional). Code simulation was run under all settings that had not been analytically proven to be unstable under any circumstances, the results were analyzed: big enough coalitions (from four to six officials) can act corrupt effectively under the first corruption-diminishing setting yet fail to do so under harsher ones.

The convexity of the cooperative extension was checked. The Myerson for the original and modified versions of the game (Theirson) values were calculated. Myerson provides individually stable allocation only under the first corruption-diminishing setting and Theirson never provides stable allocation.

References

1. What is Corruption? // Transparency International 2020, URL: https://www.transparency.org/en/what-is-corruption (accessed 03.06.2020).

2. Buckley, N. *Corruption and Power in Russia.* Foreign Policy Research Institute: Philadelphia (2018).

3. Corruption Perceptions Index // Transparency International 2021, URL: https://www.transparency.org/en/cpi/2020/index/rus (accessed 15.05.2021).

4. Spengler, D. *Detection and Deterrence in the Economics of Corruption: a Game Theoretic Analysis and some Experimental Evidence.* University of York: York (2014).

5. Attanasi, G., Rimbaud, C. & Villeval, M. *Embezzlement and guilt aversion.* IZA Discussion Papers 11956. Bonn: Institute of Labor Economics (2016).

6. Shenje, T. *Investigating the mechanism of corruption and bribery behavior: a game-theoretical methodology.* Dynamic Research Journals. Journal of Economics and Finance, Vol. 1, pp. 1–6 (2016).

7. Song, Y., Zhu, M. & Wang, H. *Game-theoretic approach for anti-corruption policy between investigating committee and inspected departments in China.* International Conference on Applied Mathematics, Simulation and Modelling, pp. 452–455. Atlantis Press: Beijing (2016).

8. Zyglidopoulos, S., Dieleman, M. & Hirsch, P. *Playing the Game: Unpacking the Rationale for Organizational Corruption in MNCs.* Journal of Management Inquiry, Vol. 29, pp. 338–349 (2019).

9. Kumacheva, S. Sh. *The Strategy of Tax Control in Conditions of Possible Mistakes and Corruption of Inspectors.* Contributions to Game Theory and Management (Petrosyan, L. A., Zenkevich, N. A. eds), Vol. 6, pp. 264–273. St. Petersburg University: St. Petersburg (2013).

10. Gorbaneva, O. I., Ougolnitsky, G. A. & Usov, A. B. *Models of corruption in hierarchical control systems* [in Russian]. Control Sciences, Vol. 1, pp. 2–10. Russian Academy of Sciences: Moscow (2015).

11. Gorbaneva, O. I., Ougolnitsky, G. A. *Price of anarchy and control mechanisms in models of concordance of public and private interests* [in Russian]. Mathematical game theory and applications, Vol. 7, pp. 50–73. Inst. of App. Math. Res. of the KarRC RAS: Petrozavodsk (2015).

12. Vasin, A., Panova, E. *Tax Collection and Corruption in Fiscal Bodies.* Economics Education and Research Consortium Working Paper Series, No. 99/10 (2000).

13. Savvateev, A. V. *Models of corruption and lobbying activities in transition economies* [in Russian]. PhD thesis in Economics. Moscow (2003).

14. Caulier, J.-F., Skoda, A. & Tanimura, E. *Allocation Rules for Networks Inspired by Cooperative Game-Theory.* Revue d'ecomie politique (2017).

15. Halko, M-L., Seppala, T. *Ultimatum game experiments.* Discussion papers, No. 140. Helsinki Center of Economic Research (2006).

16. Ramsay, K., Signorino, C. *A Statistical Model of the Ultimatum Game.* Working Papers (2009).

17. Orlov I. M. *Example of Solving a Corruption Game with Hierarchical Structure* [in Russian]. Control Processes and Stability, Vol. 7, pp. 402–407. Publishing House Fedorova G.V.: St. Petersburg (2020).

18. Orlov I. M., Kumacheva S. Sh. *Hierarchical Model of Corruption: Game-theoretic Approach* [in Russian]. Mathematical Control Theory and Its Applications Proceedings, pp. 269–271. CSRI ELEKTROPRIBOR: St. Petersburg (2020).

Appendices

Appendix A. Code listing for the non-cooperative simulation

```
import random as r
import statistics as s


class Official:
    def __init__(self, hier_id, wage, strategy, kappa, theta):
        self.hier_id = hier_id
        self.wage = wage
        # Strategy is a 3-tuple: (stealing_strategy, action_if_inspected, bribe_coeff)
        self.stealing_strategy = strategy[0]
        self.action = strategy[1]
        self.bribe = strategy[2]
        self.kappa = kappa
        self.theta = theta
        self.stealing = 0
        self.acc_win = 0
        # self.coal_id

    def steal(self, opt_stealing):
        if self.stealing_strategy == "None":
            self.stealing = 0
        elif self.stealing_strategy == "Opt":
            self.stealing = opt_stealing
        return self.stealing

    def pay_bribe(self):
        return self.bribe


class Hierarchy:
    def __init__(self, scheme, officials, cutoff_values, inspector):
        self.scheme = scheme
        self.officials = officials
        self.cutoff_values = cutoff_values
        self.inspector = inspector

    def get_with_id(self, hier_id):
        return next((x for x in self.officials if x.hier_id == hier_id), None)

    def get_boss_of_id(self, hier_id):
        for boss in self.scheme:
            if hier_id in self.scheme[boss]:
                return self.get_with_id(boss)


class Inspector:
    def __init__(self, wage, inspection_cost_func, coverup_cost_func):
        self.wage = wage
        self.acc_win = 0
        self.inspection_cost_func = inspection_cost_func
        self.coverup_cost_func = coverup_cost_func


def true_with_prob(prob):
    return r.random() < prob


# Criminal Code of Russia 160
def ru_steal_fine160(wage, stealing, is_in_coal=False):
    if stealing == 0:
        return 0

    if is_in_coal or stealing >= 1000000:
        return max(1000000, 3 * 12 * wage)
    if stealing >= 250000:
        return max(s.mean((1, 5)) * 100000, s.mean((1, 3)) * 12 * wage)
    if stealing >= 5000:
        return max(300 * 1000, 2 * 12 * wage)
    return max(120 * 1000, 1 * 12 * wage)


# Criminal Code of Russia 285.1
def ru_steal_fine(wage, stealing, is_in_coal=False):
    if stealing == 0:
        return 0

    if is_in_coal or stealing >= 7500000:
        return max(s.mean((2, 5)) * 100000, s.mean((1, 3)) * 12 * wage)
    return max(s.mean((1, 3)) * 100000, s.mean((1, 2)) * 12 * wage)


# Criminal Code of Russia 291
def ru_bribe_fine(wage, bribe, is_in_coal=False):
    if bribe >= 1000000:
        return max(s.mean((2, 4)) * 1000000, s.mean((2, 4)) * 12 * wage, s.mean((70, 90))
             * bribe)
    elif is_in_coal or bribe >= 150000:
        return max(s.mean((1, 3)) * 1000000, s.mean((1, 3)) * 12 * wage, s.mean((60, 80))
             * bribe)
    elif bribe >= 25000:
        return max(1 * 1000000, 2 * 12 * wage, s.mean((10, 40)) * bribe)
    else:
        return max(0.5 * 1000000, 1 * 12 * wage, s.mean((5, 30)) * bribe)



def threshold_func(stealing, thresholds):
    if stealing == 0:
        return 0

    for th in thresholds:
        if stealing >= th[0]:
            return th[1]


def reward_func_def(stealing):
    return threshold_func(stealing, ((400000, 75000), (100000, 40000)))


def coverup_cost_func_def(stealing):
    return threshold_func(stealing, ((400000, 11250), (100000, 5000)))

def reward_func_s1(stealing):
    return threshold_func(stealing, ((400000, 875000), (100000, 60000)))


def coverup_cost_func_s1(stealing):
    return threshold_func(stealing, ((400000, 429615.3846), (100000, 20000)))


def reward_func_s2(stealing):
    return threshold_func(stealing, ((400000, 2000000), (100000, 60000)))


def coverup_cost_func_s2(stealing):
    return threshold_func(stealing, ((400000, 1000000), (100000, 20000)))


def reward_func_s3(stealing):
    return threshold_func(stealing, ((400000, 3250000), (100000, 2000000)))


def coverup_cost_func_s3(stealing):
    return threshold_func(stealing, ((400000, 2500000), (100000, 999999.976)))


def reward_func_z1(stealing):
    return threshold_func(stealing, ((400000, 270000), (100000, 70000)))


def coverup_cost_func_z1(stealing):
    return threshold_func(stealing, ((400000, 124999), (100000, 35000)))


def reward_func_z3(stealing):
    return threshold_func(stealing, ((400000, 250000), (100000, 85000)))

```

```

def coverup_cost_func_z3(stealing):
    return threshold_func(stealing, ((400000, 125000), (100000, 39999)))


def inspection_cost_func_example(off):
    if off.hier_id[0] >= 3:
        return 22500
    if off.hier_id[0] >= 1:
        return 10000


def simulate(N, hierarchy, steal_fine_func, bribe_fine_func, reward_func):
    acc_state_util = 0
    for _ in range(N):
        # Play the game N times.
        stealing = {}
        for off_level in hierarchy.scheme.values():
            stealing[off_level] = 0

        sum_stealing = 0

        inspected_off = None
        exposers = []
        init_money = list(hierarchy.cutoff_values.values())[0][0]

        def calc_coverup_reward_inspect(exposers_list):
            coverup = 0
            reward = 0
            inspect = 0

            for exposer in exposers_list:
                coverup += hierarchy.inspector.coverup_cost_func(exposer.stealing)
                reward += reward_func(exposer.stealing)
                inspect += hierarchy.inspector.inspection_cost_func(exposer)

            return coverup, reward, inspect

        def end(x):
            state_ut = init_money
            # print(x)

            if x == 1:
                # No inspection
                for off in hierarchy.officials:
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u
                    # print("{}\t{}".format(off.hier_id, off.acc_win))

```

```python
            hierarchy.inspector.acc_win += hierarchy.inspector.wage
            state_ut -= hierarchy.inspector.wage

            return state_ut
        else:
            # print("{}\t{}".format(inspected_off.hier_id, inspected_off.acc_win))

            if x == 2:
                # No bribe
                u = inspected_off.wage + inspected_off.kappa * inspected_off.stealing - steal_fine_func(inspected_off.wage, inspected_off.stealing)
                inspected_off.acc_win += u
                state_ut -= u

                for off in set(hierarchy.officials) - {inspected_off}:
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u

                hierarchy.inspector.acc_win += hierarchy.inspector.wage - hierarchy.inspector.inspection_cost_func(inspected_off) + reward_func(inspected_off.stealing)
                state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.stealing))

                return state_ut
            elif x == 3:
                # Rejected bribe
                u = inspected_off.wage + inspected_off.kappa * inspected_off.stealing - (
                        inspected_off.pay_bribe() + steal_fine_func(inspected_off.wage, inspected_off.stealing) +
                        bribe_fine_func(inspected_off.wage, inspected_off.pay_bribe()))
                inspected_off.acc_win += u
                state_ut -= u

                for off in set(hierarchy.officials) - {inspected_off}:
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u

                hierarchy.inspector.acc_win += hierarchy.inspector.wage - hierarchy.inspector.inspection_cost_func(
                    inspected_off) + reward_func(inspected_off.stealing)

                state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.stealing))

                return state_ut
            elif x == 4:
                # Accepted bribe
                inspected_off.acc_win += inspected_off.wage + inspected_off.stealing - inspected_off.pay_bribe()
                state_ut -= (inspected_off.wage + inspected_off.stealing)

                for off in set(hierarchy.officials) - {inspected_off}:
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u

                hierarchy.inspector.acc_win += hierarchy.inspector.wage + inspected_off.pay_bribe() - (
                        hierarchy.inspector.inspection_cost_func(inspected_off) + hierarchy.inspector.coverup_cost_func(inspected_off.stealing))
                state_ut -= hierarchy.inspector.wage

                return state_ut
            else:
                sum_coverup, sum_reward, sum_inspect = calc_coverup_reward_inspect(exposers)

                if x == 5:
                    # Exposed, no bribe
                    u = inspected_off.wage + inspected_off.kappa * inspected_off.stealing - steal_fine_func(
                        inspected_off.wage, inspected_off.stealing)
                    inspected_off.acc_win += u
                    state_ut -= u

                    for exposer in exposers:
                        u = exposer.wage + exposer.kappa * exposer.stealing - exposer.theta * steal_fine_func(exposer.wage, exposer.stealing)
                        exposer.acc_win += u
                        state_ut -= u

                    for off in set(hierarchy.officials) - {inspected_off} - set(exposers):
                        u = off.wage + off.stealing
                        off.acc_win += u
                        state_ut -= u

                    hierarchy.inspector.acc_win += hierarchy.inspector.wage  + reward_func(inspected_off.stealing) + sum_reward - (
                        hierarchy.inspector.inspection_cost_func(inspected_off) + sum_inspect)
                    state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.stealing) + sum_reward)

                    return state_ut
                elif x == 6:
                    # Exposed, rejected bribe
                    u = inspected_off.wage + inspected_off.kappa * inspected_off.stealing - (steal_fine_func(
                        inspected_off.wage, inspected_off.stealing) + inspected_off.pay_bribe() + bribe_fine_func(inspected_off.wage, inspected_off.pay_bribe()))
                    inspected_off.acc_win += u
                    state_ut -= u


                    for exposer in exposers:
                        u = exposer.wage + exposer.kappa * exposer.stealing - exposer.theta * steal_fine_func(
                        exposer.wage, exposer.stealing)
                        exposer.acc_win += u
                        state_ut -= u

                    for off in set(hierarchy.officials) - {inspected_off} - set(exposers):
                        u = off.wage + off.stealing
                        off.acc_win += u
                        state_ut -= u

                    hierarchy.inspector.acc_win += hierarchy.inspector.wage + reward_func(inspected_off.stealing) + sum_reward - (
                        hierarchy.inspector.inspection_cost_func(inspected_off) + sum_inspect)
                    state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.stealing) + sum_reward)

                    return state_ut
                elif x == 7:
                    # Exposed, accepted bribe
                    inspected_off.acc_win += inspected_off.wage + inspected_off.stealing - inspected_off.pay_bribe()
                    state_ut -= (inspected_off.wage + inspected_off.stealing)

                    for off in set(hierarchy.officials) - {inspected_off}:
                        u = off.wage + off.stealing
                        off.acc_win += u
                        state_ut -= u

                    hierarchy.inspector.acc_win += hierarchy.inspector.wage + inspected_off.pay_bribe() - (hierarchy.inspector.inspection_cost_func(
                        inspected_off) + hierarchy.inspector.coverup_cost_func(inspected_off.stealing) + sum_coverup + sum_inspect)
                    state_ut -= hierarchy.inspector.wage
```

```

                    return state_ut

        # Stealing stage
        for off_level in hierarchy.scheme.values():
            cutoff_value = hierarchy.cutoff_values[off_level]
            optimal_stealing = (cutoff_value[0] - cutoff_value[1]) / len(off_level)
            for off in off_level:
                stealing[off_level] += hierarchy.get_with_id(off).steal(optimal_stealing)

        # Inspection stage: from top to bottom, from left to right

        for off_level in stealing:
            sum_stealing += stealing[off_level]
            if true_with_prob(1 - sum_stealing / init_money):
                pass
            else:
                inspected_off = hierarchy.get_with_id(r.choice(off_level))
                action = inspected_off.action
                if action == "NB":
                    acc_state_util += end(2)
                    break
                if action == "B":
                    acc_part_util = inspected_off.pay_bribe() - hierarchy.inspector.
                        coverup_cost_func(inspected_off.stealing)
                    rej_part_util = reward_func(inspected_off.stealing)
                    if acc_part_util <= rej_part_util:
                        acc_state_util += end(3)
                    else:
                        acc_state_util += end(4)
                    break
                if action == "E":
                    while True:
                        exposers.append(inspected_off)
                        inspected_off = hierarchy.get_boss_of_id(inspected_off.hier_id)
                        action = inspected_off.action
                        if action == "NB":
                            acc_state_util += end(5)
                            break
                        if action == "B":
                            exposers_coverup, exposers_reward, exposers_inspect =
                                calc_coverup_reward_inspect(exposers)
                            acc_part_util = inspected_off.pay_bribe() - hierarchy.
                                inspector.coverup_cost_func(
                                inspected_off.stealing) - exposers_coverup
                            rej_part_util = reward_func(inspected_off.stealing) +
                                exposers_reward
                            if acc_part_util <= rej_part_util:
                                acc_state_util += end(6)
                            else:
                                    acc_state_util += end(7)
                                break
                    break

            if inspected_off is None:
                acc_state_util += end(1)

    LoC = sum(stealing.values()) / init_money

    # End of N cycles, Results
    for official in hierarchy.officials:
        print("{}".format(official.acc_win / N))
    print("{}\n{}\n{}".format(hierarchy.inspector.acc_win / N, acc_state_util / N, LoC))


def run_5_str(off_scheme, in_and_out_values, funcs, b12s, b3s):
    def level_12_official(hier_id, strat):
        return Official(hier_id=hier_id, wage=40000, strategy=strat, kappa=0.3, theta
            =0.01)

    def level_3_official(hier_id, strat):
        return Official(hier_id=hier_id, wage=90000, strategy=strat, kappa=0.6, theta=1)

    def build_hier(str1, str2):
        offs = [
            level_3_official((3, 0), str2), level_3_official((3, 1), str2),
            level_12_official((2, 0), str1), level_12_official((2, 1), str1),
            level_12_official((1, 0), str1), level_12_official((1, 1), str1)
        ]
        return offs

    for b12 in b12s:
        for b3 in b3s:
            print("({}, {})".format(b12, b3))
            off_hiers = [build_hier(("Opt", "E", b12), ("Opt", "B", b3)),
                         build_hier(("Opt", "B", b12), ("Opt", "B", b3)),
                         build_hier(("None", "NB", b12), ("Opt", "B", b3)),
                         build_hier(("Opt", "B", b12), ("None", "NB", b3)),
                         build_hier(("None", "NB", b12), ("None", "NB", b3)),
                         ]

            for off_hier in off_hiers:
                inspector = Inspector(70000, inspection_cost_func_example, funcs[0])
                hierarchy = Hierarchy(off_scheme, off_hier, in_and_out_values, inspector)
                simulate(N=500000, hierarchy=hierarchy, steal_fine_func=ru_steal_fine,
                         bribe_fine_func=ru_bribe_fine, reward_func=funcs[1])


def main():
    strategies = (("None", "NB", 0), ("Opt", "E", 0), ("Opt", "B", 0.99))

    off_scheme = {
        (4, 0): ((3, 0), (3, 1)),
        (3, 0): ((2, 0), (2, 1)),
        (3, 1): ((1, 0), (1, 1)),
    }
    in_and_out_values = {
        ((3, 0), (3, 1)): (3000000, 2000000),
        ((2, 0), (2, 1)): (2000000 / 2, 750000),
        ((1, 0), (1, 1)): (2000000 / 2, 750000)
    }

    B12_d = (22500.5, 45001, 67501.5)
    B3_d = (43125.5, 86251, 108751, 131251, 196876.5)

    B12_s1 = (40000.5, 80001, 120001.5)
    B3_s1 = (652308.192307692, 1304616.38461538, 1344616.38461538, 1384616.38461538,
        2076924.57692308)

    B12_s2 = (40000.5, 80001, 120001.5)
    B3_s2 = (1500000.5, 3000001, 3040001, 3080001, 4620001.5)

    B12_s3 = (1500000.488, 3000000.976, 4500001.464)
    B3_s3 = (2875000.5, 5750001, 7250000.988, 8750000.976, 13125001.464)

    B12_z1 = (78750.75, 105001)
    B3_z1 = (197500, 395000, 447500, 500000)

    B12_z3 = (62500, 125000)
    B3_z3 = (187500.5, 375001, 437500.5, 500000)

    B12_ex = (62500,)
    B3_ex = (150000, )

    run_5_str(off_scheme=off_scheme, in_and_out_values=in_and_out_values, funcs=(
        coverup_cost_func_def, reward_func_def), b12s=B12_ex, b3s=B3_ex)


if __name__ == "__main__":
    main()
```

Appendix B. Rules analysis

Only equally shared bribe

$$ESBU_{n,i}^{SSL,SSR} = S_s - \frac{\alpha_n^{eff}}{2} B_s < S_s \quad n = \begin{cases} 1 \, if \; SSR \\ 2 \, if \; SSL \end{cases}$$

The further reasoning regards subordinate due to the easier proof and the fact that coalition-rule pair is not stable if there is at least one member for whom the conditions do not hold.

$$ESBU_{j,i}^{BB1SL,BB1SR} = S_s - \frac{(\alpha_3 + \frac{\alpha_j^{eff}}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_j^{eff}}{2} B_s}{3} < S_s$$

$$j = \begin{cases} 1 \, if \; BB1SR \\ 2 \, if \; BB1SL \end{cases}$$

$$ESBU_{j,i}^{1B2SL,1B2SR} = S_s - \frac{\frac{\alpha_3}{2} B_b \alpha_j^{eff} B_s}{3} < S_s \quad j = \begin{cases} 1 \, if \; 1B2SR \\ 2 \, if \; 1B2SL \end{cases}$$

$$ESBU_{j,i}^{2SBBL,2SBBR} = S_s - \frac{(\frac{\alpha_3}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_3}{2} B_b + \alpha_j^{eff} B_s}{4} < S_s$$

$$j = \begin{cases} 1 \, if \; 2SBBR \\ 2 \, if \; 2SBBL \end{cases}$$

$$ESBU_{j,i}^{1SBB1S} = S_s - \frac{(\frac{\alpha_3}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_3}{2} B_b + \alpha_j^{eff} B_s}{4} < S_s$$

$$ESBU_{j,i}^{2SBB1SL,2SBB1SR} = S_s - \frac{[\frac{\alpha_3+\alpha_{3-j}^{eff}}{2}B_{ch} + \frac{\alpha_3}{2}B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2})B_s]}{5} < S_s$$

$$j = \begin{cases} 1 \; if \; 2SBB1SR \\ 2 \; if \; 2SBB1SL \end{cases}$$

$$ESBU_{j,i}^{GC} = S_s - \frac{\alpha_3 B_b + (\alpha_2^{eff} + \alpha_1^{eff})B_s}{6} < S_s$$

Only proportionally shared bribe

SSL, SSR: the only way to share the bribe in this type of coalition is $B_{C,n} = B_C$

$$PSBU_{n,i}^{SSL,SSR} = S_s - \frac{\alpha_n^{eff}}{2} B_s < S_s \quad n = \begin{cases} 1\; if\; SSR \\ 2\; if\; SSL \end{cases}$$

BB: in the similar manner we get $B_{C,3} = B_C$ the reasoning from there is identical to the respective BGA.

$$PSBU_{j,i}^{1B1SL,1B1SR} = S_s - \gamma_j [\frac{\alpha_3 + \alpha_j^{eff}}{2} B_{ch} + \frac{\alpha_j^{eff}}{2} B_s]$$

$$PSBU_{3,2\%j}^{1B1SL,1B1SR} = S_b - \gamma_3 [\frac{\alpha_3 + \alpha_j^{eff}}{2} B_{ch} + \frac{\alpha_j^{eff}}{2} B_s]$$

$$j = \begin{cases} 1\; if\; 1B1SR \\ 2\; if\; 1B1SL \end{cases}$$

$$PSBU_{j,i}^{BB1SL,BB1SR} = S_s - \gamma_j [(\alpha_3 + \frac{\alpha_j^{eff}}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_j^{eff}}{2} B_s]$$

$$PSBU_{3,i}^{BB1SL,BB1SR} = S_b - \frac{\gamma_3 [(\alpha_3 + \frac{\alpha_j^{eff}}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_j^{eff}}{2} B_s]}{2}$$

$$j = \begin{cases} 1\; if\; BB1SR \\ 2\; if\; BB1SL \end{cases}$$

$$PSBU_{j,i}^{1B2SL,1B2SR} = S_s - \frac{\gamma_j [\frac{\alpha_3}{2} B_b + \frac{\alpha_j^{eff}}{2} B_s]}{2}$$

$$PSBU_{3,2\%j}^{1B2SL,1B2SR} = S_b - \gamma_3 [\frac{\alpha_3}{2} B_b + \frac{\alpha_j^{eff}}{2} B_s]$$

$$j = \begin{cases} 1 \; if \; 1B2SR \\ 2 \; if \; 1B2SL \end{cases}$$

$$BC^{2SBBL,2SBBR} = [\frac{\alpha_3 + \alpha_{3-j}^{eff}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2}) B_s]$$

$$PSBU_{j,i}^{2SBBL,2SBBR} = S_s - \frac{\gamma_j BC^{2SBBL,2SBBR}}{2}$$

$$PSBU_{3,i}^{2SBBL,2SBBR} = S_b - \frac{\gamma_3 BC^{2SBBL,2SBBR}}{2}$$

$$j = \begin{cases} 1 \; if \; 2SBBR \\ 2 \; if \; 2SBBL \end{cases}$$

$$BC^{1SBB1S} = (\alpha_3 + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2}) B_{ch} + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2} B_s$$

$$PSBU_{j,i}^{1SBB1S} = S_s - \gamma_j BC^{1SBB1S}$$

$$PSBU_{3,i}^{1SBB1S} = S_b - \frac{\gamma_3 BC^{1SBB1S}}{2}$$

$$BC^{2SBB1SL,2SBB1SR} = \frac{\alpha_3 + \alpha_{3-j}^{eff}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2}) B_s$$

$$PSBU_{j,i}^{2SBB1SL,2SBB1SR} = S_s - \frac{\gamma_j BC^{2SBB1SL,2SBB1SR}}{2}$$

$$PSBU_{3-j,i}^{2SBB1SL,2SBB1SR} = S_s - \gamma_{3-j} BC^{2SBB1SL,2SBB1SR}$$

$$PSBU_{3,i}^{2SBB1SL,2SBB1SR} = S_b - \frac{\gamma_3 BC^{2SBB1SL,2SBB1SR}}{2}$$

$$j = \begin{cases} 1 \; if \; 2SBB1SR \\ 2 \; if \; 2SBB1SL \end{cases}$$

$$PSBU_{n,i}^{GC} = S_{n,i} - \frac{\gamma_n [\alpha_3 B_b + (\alpha_2^{eff} + \alpha_1^{eff}) B_s]}{2} \quad n = 1, 2, 3$$

Equally shared bribe plus bonus to subordinate

SSL, SSR: there is only one level, so

$$ESBBSU_{n,i}^{SSL,SSR} = S_s - \frac{\alpha_n^{eff}}{2} B_s < S_s \quad n = \begin{cases} 1\, if\ SSR \\ 2\, if\ SSL \end{cases}$$

BB: there is only one level; reasoning is identical to the respective BGA.

$$BC^{1B1SL,1B1SR} = \frac{\alpha_3 + \alpha_j^{eff}}{2} B_{ch} + \frac{\alpha_j^{eff}}{2} B_s$$

$$ESBBSU_{j,i}^{1B1SL,1B1SR} = S_s - \frac{BC^{1B1SL,1B1SR}}{2} + BS_{3,2\%j}$$

$$ESBBSU_{3,2\%j}^{1B1SL,1B1SR} = S_b - \frac{BC^{1B1SL,1B1SR}}{2} - BS_{3,2\%j}$$

$$j = \begin{cases} 1\, if\ 1B1SR \\ 2\, if\ 1B1SL \end{cases}$$

$$BC^{BB1SL,BB1SR} = (\alpha_3 + \frac{\alpha_j^{eff}}{2} + \alpha_{3-j}^{eff}) B_{ch} + \frac{\alpha_j^{eff}}{2} B_s$$

$$ESBBSU_{j,i}^{BB1SL,BB1SR} = S_s - \frac{BC^{1B1SL,1B1SR}}{3} + BS_{3,2\%j}$$

$$ESBBSU_{3,2\%j}^{BB1SL,BB1SR} = S_b - \frac{BC^{1B1SL,1B1SR}}{3} - BS_{3,2\%j}$$

$$ESBBSU_{3,j-1}^{BB1SL,BB1SR} = S_b - \frac{BC^{1B1SL,1B1SR}}{3}$$

$$j = \begin{cases} 1\, if\ BB1SR \\ 2\, if\ BB1SL \end{cases}$$

$$BC^{1B2SL,1B2SR} = \frac{\alpha_3}{2} B_b + \alpha_j^{eff} B_s$$

$$ESBBSU_{j,i}^{1B2SL,1B2SR} = S_s - \frac{BC^{1B2SL,1B2SR}}{3} + BS_{3,2\%j}$$

$$ESBBSU^{1B2SL,1B2SR}_{3,2\%j} = S_b - \frac{BC^{1B2SL,1B2SR}}{3} - 2BS_{3,2\%j}$$

$$j = \begin{cases} 1\; if\; 1B2SR \\ 2\; if\; 1B2SL \end{cases}$$

$$BC^{2SBBL,2SBBR} = \frac{\alpha_3 + \alpha^{eff}_{3-j}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha^{eff}_j + \frac{\alpha^{eff}_{3-j}}{2}) B_s$$

$$ESBBSU^{2SBBL,2SBBR}_{j,i} = S_s - \frac{BC^{2SBBL,2SBBR}}{4} + BS_{3,2\%j}$$

$$ESBBSU^{2SBBL,2SBBR}_{3-j,i} = S_s - \frac{BC^{2SBBL,2SBBR}}{4} + BS_{3,j-1}$$

$$ESBBSU^{2SBBL,2SBBR}_{3,2\%j} = S_b - \frac{BC^{2SBBL,2SBBR}}{4} - 2BS_{3,2\%j}$$

$$ESBBSU^{2SBBL,2SBBR}_{3,j-1} = S_b - \frac{BC^{2SBBL,2SBBR}}{4} - BS_{3,j-1}$$

$$j = \begin{cases} 1\; if\; 2SBBR \\ 2\; if\; 2SBBL \end{cases}$$

$$BC^{1SBB1S} = (\alpha_3 + \frac{\alpha^{eff}_2 + \alpha^{eff}_1}{2}) B_{ch} + \frac{\alpha^{eff}_2 + \alpha^{eff}_1}{2} B_s$$

$$ESBBSU^{1SBB1S}_{j,i} = S_s - \frac{BC^{1SBB1S}}{4} + BS_{3,2\%j}$$

$$ESBBSU^{1SBB1S}_{3,2\%j} = S_b - \frac{BC^{1SBB1S}}{4} - BS_{3,2\%j}$$

$$j = 1, 2$$

$$BC^{2SBB1SL,2SBB1SR} = \frac{\alpha_3 + \alpha^{eff}_{3-j}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha^{eff}_j + \frac{\alpha^{eff}_{3-j}}{2}) B_s$$

$$ESBBSU^{2SBB1SL,2SBB1SR}_{j,i} = S_s - \frac{BC^{2SBBL,2SBBR}}{5} + BS_{3,2\%j}$$

$$ESBBSU_{3-j,i}^{2SBB1SL,2SBB1SR} = S_s - \frac{BC^{2SBBL,2SBBR}}{5} + BS_{3,j-1}$$

$$ESBBSU_{3,2\%j}^{2SBB1SL,2SBB1SR} = S_b - \frac{BC^{2SBBL,2SBBR}}{5} - 2BS_{3,2\%j}$$

$$ESBBSU_{3,j-1}^{2SBB1SL,2SBB1SR} = S_b - \frac{BC^{2SBBL,2SBBR}}{5} - BS_{3,j-1}$$

$$j = \begin{cases} 1\ if\ 2SBB1SR \\ 2\ if\ 2SBB1SL \end{cases}$$

$$BC^{GC} = (\alpha_3 + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2})B_{ch} + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2}B_s$$

$$ESBBSU_{j,i}^{GC} = S_s - \frac{BC^{GC}}{6} + BS_{3,2\%j}$$

$$ESBBSU_{3,2\%j}^{GC} = S_b - \frac{BC^{GC}}{6} - 2BS_{3,2\%j}$$

$$j = 1, 2$$

Proportionally shared bribe plus bonus to subordinate

SSL, SSR: there is only one level, so

$$PSBBSU^{SSL,SSR}_{n,i} = S_s - \frac{\alpha^{eff}_n}{2} B_s < S_s \quad n = \begin{cases} 1 \; if \; SSR \\ 2 \; if \; SSL \end{cases}$$

BB: there is only one level; reasoning is identical to the respective BGA.

$$BC^{1B1SL,1B1SR} = \frac{\alpha_3 + \alpha^{eff}_j}{2} B_{ch} + \frac{\alpha^{eff}_j}{2} B_s$$

$$PSBBSU^{1B1SL,1B1SR}_{j,i} = S_s - \gamma_j BC^{1B1SL,1B1SR} + BS_{3,2\%j}$$

$$PSBBSU^{1B1SL,1B1SR}_{3,2\%j} = S_b - \gamma_3 BC^{1B1SL,1B1SR} - BS_{3,2\%j}$$

$$j = \begin{cases} 1 \; if \; 1B1SR \\ 2 \; if \; 1B1SL \end{cases}$$

$$BC^{BB1SL,BB1SR} = (\alpha_3 + \frac{\alpha^{eff}_j}{2} + \alpha^{eff}_{3-j}) B_{ch} + \frac{\alpha^{eff}_j}{2} B_s$$

$$PSBBSU^{BB1SL,BB1SR}_{j,i} = S_s - \gamma_j BC^{BB1SL,BB1SR} + BS_{3,2\%j}$$

$$PSBBSU^{BB1SL,BB1SR}_{3,2\%j} = S_b - \frac{\gamma_3 BC^{BB1SL,BB1SR}}{2} - BS_{3,2\%j}$$

$$PSBBSU^{BB1SL,BB1SR}_{3,j-1} = S_b - \frac{\gamma_3 BC^{BB1SL,BB1SR}}{2}$$

$$j = \begin{cases} 1 \; if \; BB1SR \\ 2 \; if \; BB1SL \end{cases}$$

$$BC^{1B2SL,1B2SR} = \frac{\alpha_3}{2} B_b + \frac{\alpha^{eff}_j}{2} B_s$$

$$PSBBSU^{1B2SL,1B2SR}_{j,i} = S_s - \frac{\gamma_j BC^{1B2SL,1B2SR}}{2} + BS_{3,2\%j}$$

$$PSBBSU^{1B2SL,1B2SR}_{3,2\%j} = S_b - \gamma_3 BC^{1B2SL,1B2SR} - 2BS_{3,2\%j}$$

$$j = \begin{cases} 1\ if\ 1B2SR \\ 2\ if\ 1B2SL \end{cases}$$

$$BC^{2SBBL,2SBBR} = \frac{\alpha_3 + \alpha_{3-j}^{eff}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2}) B_s$$

$$PSBBSU_{j,i}^{2SBBL,2SBBR} = S_s - \frac{\gamma_j BC^{2SBBL,2SBBR}}{2} + BS_{3,2\%j}$$

$$PSBBSU_{3,2\%j}^{2SBBL,2SBBR} = S_b - \frac{\gamma_3 BC^{2SBBL,2SBBR}}{2} - 2BS_{3,2\%j}$$

$$PSBBSU_{3,j-1}^{2SBBL,2SBBR} = S_b - \frac{\gamma_3 BC^{2SBBL,2SBBR}}{2}$$

$$j = \begin{cases} 1\ if\ 2SBBR \\ 2\ if\ 2SBBL \end{cases}$$

$$BC^{1SBB1S} = (\alpha_3 + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2}) B_{ch} + \frac{\alpha_2^{eff} + \alpha_1^{eff}}{2} B_s$$

$$PSBBSU_{j,i}^{1SBB1S} = S_s - \gamma_j BC^{1SBB1S} + BS_{3,i} \quad j = 1, 2$$

$$PSBBSU_{3,i}^{1SBB1S} = S_b - \frac{\gamma_3 BC^{1SBB1S}}{2} - BS_{3,i}$$

$$BC^{2SBB1SL,2SBB1SR} = \frac{\alpha_3 + \alpha_{3-j}^{eff}}{2} B_{ch} + \frac{\alpha_3}{2} B_b + (\alpha_j^{eff} + \frac{\alpha_{3-j}^{eff}}{2}) B_s$$

$$PSBBSU_{j,i}^{2SBB1SL,2SBB1SR} = S_s - \frac{\gamma_j BC^{2SBB1SL,2SBB1SR}}{2} + BS_{3,2\%j}$$

$$PSBBSU_{3-j,i}^{2SBB1SL,2SBB1SR} = S_s - \frac{\gamma_j BC^{2SBB1SL,2SBB1SR}}{2} + BS_{3,j-1}$$

$$PSBBSU_{3,2\%j}^{2SBB1SL,2SBB1SR} = S_s - \gamma_{3-j} BC^{2SBB1SL,2SBB1SR} - 2BS_{3,2\%j}$$

$$PSBBSU_{3,j-1}^{2SBB1SL,2SBB1SR} = S_b - \frac{\gamma_3 BC^{2SBB1SL,2SBB1SR}}{2} - BS_{3,j-1}$$

$$j = \begin{cases} 1 \; if \; 2SBB1SR \\ 2 \; if \; 2SBB1SL \end{cases}$$

$$BC^{GC} = \alpha_3 B_b + (\alpha_2^{eff} + \alpha_1^{eff}) B_s$$

$$PSBBSU_{j,i}^{GC} = S_s - \frac{\gamma_j BC^{GC}}{2} + BS_{3,i} \quad j = 1, 2$$

$$PSBBSU_{3,i}^{GC} = S_b - \frac{\gamma_3 BC^{GC}}{2} - 2BS_{3,i}$$

Appendix C. Code listing for the cooperative simulation

```
import random as r
import statistics as s
import matplotlib.pyplot as plt
import numpy as np
from itertools import chain
from matplotlib.ticker import FuncFormatter


class Official:
    def __init__(self, hier_id, wage, strategy, kappa, theta, is_in_coal):
        self.hier_id = hier_id
        self.wage = wage
        # Strategy is a 3-tuple: (stealing_strategy, action_if_inspected, bribes)
        self.stealing_strategy = strategy[0]
        self.action = strategy[1]
        self.bribe = strategy[2]
        self.kappa = kappa
        self.theta = theta
        self.is_in_coal = is_in_coal
        self.stealing = 0
        self.acc_win = 0

    def steal(self, opt_stealing):
        if self.stealing_strategy == "None":
            self.stealing = 0
        elif self.stealing_strategy == "Opt":
            self.stealing = opt_stealing
        return self.stealing

    def pay_bribe(self, sure=False):
        return self.bribe[sure]

# sure = all(sub_id in coal_offs for sub_id in off_scheme[off_id])
# sure = False = 0 -> B[ch]
# sure = True = 1 -> B[b]
# subs don't care


class Hierarchy:
    def __init__(self, scheme, officials, cutoff_values, inspector):
        self.scheme = scheme
        self.officials = officials
        self.cutoff_values = cutoff_values
        self.inspector = inspector

    def get_with_id(self, hier_id):
        return next((x for x in self.officials if x.hier_id == hier_id), None)

    def get_boss_of_id(self, hier_id):
        for boss in self.scheme:
            if hier_id in self.scheme[boss]:
                return self.get_with_id(boss)


class Coalition:
    def __init__(self, scheme_tuple, hierarchy, rule):
        self.scheme_name = scheme_tuple[0]
        self.off_ids = scheme_tuple[1]
        self.hierarchy = hierarchy
        self.rule = rule
        self.bribe = 0
        self.total_stealing = 0  # Do I really need this?
        self.utils = {}

    def calc_stealing(self):
        if self.total_stealing == 0:
            for off_id in self.off_ids:
                self.total_stealing = self.total_stealing + self.hierarchy.get_with_id(off_id).stealing

        return self.total_stealing

    def pay_bribe(self, inspected_id):
        self.bribe = self.hierarchy.get_with_id(inspected_id).pay_bribe(all(sub_id in self.off_ids for sub_id in self.hierarchy.scheme.get(inspected_id, [])))
        return self.bribe

    def calc_utils(self):
        for off_id in self.off_ids:
            self.utils[off_id] = self.rule(off_id, self.off_ids, self.bribe, self.total_stealing, self.hierarchy)

        return sum(self.utils.values()) == (self.total_stealing - self.bribe)


def EQ_rule(off_id, coal_off_ids, bribe, coal_stealing, hier_scheme):
    return (coal_stealing - bribe) / len(coal_off_ids)


def SS_with_xi(xi):
    def SS_rule(off_id, coal_off_ids, bribe, coal_stealing, hier):
        U = 0
        bl = 3
        subs = set()
        for off in hier.scheme.keys():
            if off[0] == bl:
                subs = subs.union(set(hier.scheme[off]))

        N_bl = len([1 for off in coal_off_ids if off[0] == bl])

        if off_id[0] == bl:
            U = hier.get_with_id(off_id).stealing - (bribe + xi * len(subs.intersection(
                set(coal_off_ids)))) / N_bl

        elif off_id[0] in (1, 2):
            U = hier.get_with_id(off_id).stealing + xi

# Hard-coded and works only on the hierarchy suggested in the work: 3 levels with 2
    officials on each.
        return U

    return SS_rule

class Inspector:
    def __init__(self, wage, inspection_cost_func, coverup_cost_func):
        self.wage = wage
        self.acc_win = 0
        self.inspection_cost_func = inspection_cost_func
        self.coverup_cost_func = coverup_cost_func


def true_with_prob(prob):
    return r.random() < prob


# Criminal Code of Russia 160
def ru_steal_fine160(wage, stealing, is_in_coal=False):
    if stealing == 0:
        return 0

    if is_in_coal or stealing >= 1000000:
        return max(1000000, 3 * 12 * wage)
    if stealing >= 250000:
        return max(s.mean((1, 5)) * 100000, s.mean((1, 3)) * 12 * wage)
    if stealing >= 5000:
        return max(300 * 1000, 2 * 12 * wage)
    return max(120 * 1000, 1 * 12 * wage)


# Criminal Code of Russia 285.1
def ru_steal_fine(wage, stealing, is_in_coal=False):
    if stealing == 0:
        return 0

    if is_in_coal or stealing >= 7500000:
        return max(s.mean((2, 5)) * 100000, s.mean((1, 3)) * 12 * wage)
    return max(s.mean((1, 3)) * 100000, s.mean((1, 2)) * 12 * wage)


```

```
# Criminal Code of Russia 291
def ru_bribe_fine(wage, bribe, is_in_coal=False):
    if bribe >= 1000000:
        return max(s.mean((2, 4)) * 1000000, s.mean((2, 4)) * 12 * wage, s.mean((70, 90))
            * bribe)
    elif is_in_coal or bribe >= 150000:
        return max(s.mean((1, 3)) * 1000000, s.mean((1, 3)) * 12 * wage, s.mean((60, 80))
            * bribe)
    elif bribe >= 25000:
        return max(1 * 1000000, 2 * 12 * wage, s.mean((10, 40)) * bribe)
    else:
        return max(0.5 * 1000000, 1 * 12 * wage, s.mean((5, 30)) * bribe)


def threshold_func(stealing, thresholds):
    if stealing == 0:
        return 0

    for th in thresholds:
        if stealing >= th[0]:
            return th[1]


def reward_func_def(stealing):
    return threshold_func(stealing, ((400000, 75000), (100000, 40000)))


def coverup_cost_func_def(stealing):
    return threshold_func(stealing, ((400000, 11250), (100000, 5000)))


def reward_func_s1(stealing):
    return threshold_func(stealing, ((400000, 875000), (100000, 60000)))


def coverup_cost_func_s1(stealing):
    return threshold_func(stealing, ((400000, 429615.3846), (100000, 20000)))


def reward_func_s2(stealing):
    return threshold_func(stealing, ((400000, 2000000), (100000, 60000)))


def coverup_cost_func_s2(stealing):
    return threshold_func(stealing, ((400000, 1000000), (100000, 20000)))


def reward_func_s3(stealing):
    return threshold_func(stealing, ((400000, 3250000), (100000, 2000000)))

```

```python

def coverup_cost_func_s3(stealing):
    return threshold_func(stealing, ((400000, 2500000), (100000, 999999.976)))


def reward_func_z1(stealing):
    return threshold_func(stealing, ((400000, 270000 ), (100000, 70000)))


def coverup_cost_func_z1(stealing):
    return threshold_func(stealing, ((400000, 124999 ), (100000, 35000)))


def reward_func_z3(stealing):
    return threshold_func(stealing, ((400000, 250000 ), (100000, 85000)))


def coverup_cost_func_z3(stealing):
    return threshold_func(stealing, ((400000, 125000 ), (100000, 39999)))


def inspection_cost_func_example(off):
    if off.hier_id[0] >= 3:
        return 22500
    if off.hier_id[0] >= 1:
        return 10000


def simulate(N, hierarchy, steal_fine_func, bribe_fine_func, reward_func, coalition):
    acc_state_util = 0
    for _ in range(N):
        # Play the game N times.
        stealing = {}
        for off_level in hierarchy.scheme.values():
            stealing[off_level] = 0

        sum_stealing = 0

        inspected_off = None
        exposers = []
        init_money = list(hierarchy.cutoff_values.values())[0][0]
        # print(hierarchy.officials)
        coal_officials = []
        for i in range(len(hierarchy.officials)):
            for off_id in coalition.off_ids:
                if hierarchy.officials[i].hier_id == off_id:
                    coal_officials.append(hierarchy.officials[i])

        coal_officials = set(coal_officials)
        non_coal_officials = set(hierarchy.officials) ^ coal_officials

        def calc_coverup_reward_inspect(exposers_list):
            coverup = 0
            reward = 0
            inspect = 0

            for exposer in exposers_list:
                coverup += hierarchy.inspector.coverup_cost_func(exposer.stealing)
                reward += reward_func(exposer.stealing)
                inspect += hierarchy.inspector.inspection_cost_func(exposer)

            return coverup, reward, inspect

        def end(x):
            utils_correct = coalition.calc_utils()
            # Returns False in case of fine?

            if not utils_correct:
                print("ERROR in calculating coalitional utilities, review the rule!")
                exit(-1)

            state_ut = init_money

            if x == 1:
                # No inspection
                for off in coal_officials:
                    u = off.wage + coalition.utils[off.hier_id]
                    off.acc_win += u
                    state_ut -= u

                for off in non_coal_officials:
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u

                hierarchy.inspector.acc_win += hierarchy.inspector.wage
                state_ut -= hierarchy.inspector.wage

                return state_ut
            else:
                if x == 2:
                    # No bribe
                    if inspected_off.is_in_coal:
                        for off in coal_officials:
                            u = off.wage + coalition.utils[off.hier_id] - steal_fine_func
                                (
                                off.wage, coalition.total_stealing, True)
                            off.acc_win += u
                            state_ut -= u

                    else:
                        u = inspected_off.wage + inspected_off.kappa * inspected_off.
                            stealing - steal_fine_func(
                            inspected_off.wage, inspected_off.stealing, False)
                        inspected_off.acc_win += u
                        state_ut -= u

                        for off in coal_officials:
                            u = off.wage + coalition.utils[off.hier_id]
                            off.acc_win += u
                            state_ut -= u

                    for off in non_coal_officials - {inspected_off}:
                        u = off.wage + off.stealing
                        off.acc_win += u
                        state_ut -= u

                    hierarchy.inspector.acc_win += hierarchy.inspector.wage - hierarchy.
                        inspector.inspection_cost_func(
                            inspected_off) + reward_func(inspected_off.stealing)
                    state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.
                        stealing))

                    return state_ut
                elif x == 3:
                    # Rejected bribe
                    if inspected_off.is_in_coal:
                        for off in coal_officials:
                            bribe = coalition.pay_bribe(inspected_off.hier_id)
                            u = off.wage + coalition.utils[off.hier_id] - (
                                steal_fine_func(
                                off.wage, coalition.total_stealing, True) +
                                    bribe_fine_func(off.wage, bribe, True))
                            off.acc_win += u
                            state_ut -= u

                    else:
                        u = inspected_off.wage + inspected_off.kappa * inspected_off.
                            stealing - (
                                inspected_off.pay_bribe(False) + steal_fine_func(
                                    inspected_off.wage, inspected_off.stealing, False) +
                                bribe_fine_func(inspected_off.wage, inspected_off.
                                    pay_bribe(False), False))
                        inspected_off.acc_win += u
                        state_ut -= u

                        for off in coal_officials:
                            u = off.wage + coalition.utils[off.hier_id]
                            off.acc_win += u
                            state_ut -= u
```

```python

            for off in non_coal_officials - {inspected_off}:
                u = off.wage + off.stealing
                off.acc_win += u
                state_ut -= u

            hierarchy.inspector.acc_win += hierarchy.inspector.wage - hierarchy.inspector.inspection_cost_func(
                inspected_off) + reward_func(inspected_off.stealing)

            state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.stealing))

            return state_ut
        elif x == 4:
            # Accepted bribe
            if inspected_off.is_in_coal:
                hierarchy.inspector.acc_win += hierarchy.inspector.wage + coalition.pay_bribe(inspected_off.hier_id) - (
                        hierarchy.inspector.inspection_cost_func(inspected_off) + hierarchy.inspector.coverup_cost_func(inspected_off.stealing))

            else:
                inspected_off.acc_win += inspected_off.wage + inspected_off.stealing - inspected_off.pay_bribe(False)
                state_ut -= (inspected_off.wage + inspected_off.stealing)
                hierarchy.inspector.acc_win += hierarchy.inspector.wage + inspected_off.pay_bribe() - (
                        hierarchy.inspector.inspection_cost_func(
                            inspected_off) + hierarchy.inspector.coverup_cost_func(inspected_off.stealing))

            for off in coal_officials:
                u = off.wage + coalition.utils[off.hier_id]
                off.acc_win += u
                state_ut -= u

            for off in non_coal_officials - {inspected_off}:
                u = off.wage + off.stealing
                off.acc_win += u
                state_ut -= u

            state_ut -= hierarchy.inspector.wage

            return state_ut
        else:
            sum_coverup, sum_reward, sum_inspect = calc_coverup_reward_inspect(exposers)

            if x == 5:
                # Exposed, no bribe
                if inspected_off.is_in_coal:
                    for off in coal_officials:
                        u = off.wage + coalition.utils[off.hier_id] - steal_fine_func(
                            off.wage, coalition.total_stealing, True)
                        off.acc_win += u
                        state_ut -= u

                else:
                    u = inspected_off.wage + inspected_off.kappa * inspected_off.stealing - steal_fine_func(
                        inspected_off.wage, inspected_off.stealing, False)
                    inspected_off.acc_win += u
                    state_ut -= u

                    for off in coal_officials:
                        u = off.wage + coalition.utils[off.hier_id]
                        off.acc_win += u
                        state_ut -= u

                for exposer in exposers:
                    u = exposer.wage + exposer.kappa * exposer.stealing - exposer.theta * steal_fine_func(
                        exposer.wage, exposer.stealing, False)
                    exposer.acc_win += u
                    state_ut -= u

                for off in non_coal_officials - {inspected_off} - set(exposers):
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u

                hierarchy.inspector.acc_win += hierarchy.inspector.wage + reward_func(
                    inspected_off.stealing) + sum_reward - (
                                                    hierarchy.inspector.inspection_cost_func(
                                                        inspected_off) + sum_inspect)
                state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off.stealing) + sum_reward)

                return state_ut
            elif x == 6:
                # Exposed, rejected bribe
                if inspected_off.is_in_coal:
                    bribe = coalition.pay_bribe(inspected_off.hier_id)
                    hierarchy.inspector.acc_win += hierarchy.inspector.wage +
                        reward_func(
                        inspected_off.stealing) + sum_reward - (hierarchy.
                            inspector.inspection_cost_func(inspected_off) +
                            sum_inspect)

                    for off in coal_officials:
                        u = off.wage + coalition.utils[off.hier_id] - (
                            steal_fine_func(
                            off.wage, coalition.total_stealing, True) +
                                bribe_fine_func(off.wage, bribe, True))
                        off.acc_win += u
                        state_ut -= u

                else:
                    hierarchy.inspector.acc_win += hierarchy.inspector.wage +
                        reward_func(
                        inspected_off.stealing) + sum_reward - (hierarchy.
                            inspector.inspection_cost_func(inspected_off) +
                            sum_inspect)
                    u = inspected_off.wage + inspected_off.kappa * inspected_off.
                        stealing - (steal_fine_func(
                        inspected_off.wage, inspected_off.stealing, False) +
                            inspected_off.pay_bribe() + bribe_fine_func(
                        inspected_off.wage, inspected_off.pay_bribe(False), False
                            ))
                    inspected_off.acc_win += u
                    state_ut -= u

                    for off in coal_officials:
                        u = off.wage + coalition.utils[off.hier_id]
                        off.acc_win += u
                        state_ut -= u

                for exposer in exposers:
                    u = exposer.wage + exposer.kappa * exposer.stealing - exposer
                        .theta * steal_fine_func(
                        exposer.wage, exposer.stealing, False)
                    exposer.acc_win += u
                    state_ut -= u

                for off in non_coal_officials - {inspected_off} - set(exposers):
                    u = off.wage + off.stealing
                    off.acc_win += u
                    state_ut -= u

                state_ut -= (hierarchy.inspector.wage + reward_func(inspected_off
                    .stealing) + sum_reward)

                return state_ut
            elif x == 7:
```

```
                        # Exposed, accepted bribe
                        if inspected_off.is_in_coal:
                            hierarchy.inspector.acc_win += hierarchy.inspector.wage + coalition.pay_bribe(inspected_off.hier_id) - (
                                    hierarchy.inspector.inspection_cost_func(
                                        inspected_off) + hierarchy.inspector.coverup_cost_func(
                                inspected_off.stealing) + sum_coverup + sum_inspect)

                        else:
                            inspected_off.acc_win += inspected_off.wage + inspected_off.stealing - inspected_off.pay_bribe(False)
                            state_ut -= (inspected_off.wage + inspected_off.stealing)

                            hierarchy.inspector.acc_win += hierarchy.inspector.wage + inspected_off.pay_bribe(False) - (
                                    hierarchy.inspector.inspection_cost_func(inspected_off) + hierarchy.inspector.coverup_cost_func(
                                inspected_off.stealing) + sum_coverup + sum_inspect)

                        for off in coal_officials:
                            u = off.wage + coalition.utils[off.hier_id]
                            off.acc_win += u
                            state_ut -= u

                        for off in non_coal_officials - {inspected_off}:
                            u = off.wage + off.stealing
                            off.acc_win += u
                            state_ut -= u

                        state_ut -= hierarchy.inspector.wage

                        return state_ut

        # Stealing stage
        for off_level in hierarchy.scheme.values():
            cutoff_value = hierarchy.cutoff_values[off_level]
            optimal_stealing = (cutoff_value[0] - cutoff_value[1]) / len(off_level)
            for off in off_level:
                stealing[off_level] += hierarchy.get_with_id(off).steal(optimal_stealing)

        coalition.calc_stealing()
        # Inspection stage: from top to bottom, from left to right


        for off_level in stealing:
            sum_stealing += stealing[off_level]
            if true_with_prob(1 - sum_stealing / init_money):
                pass
```

```
else:
    inspected_off = hierarchy.get_with_id(r.choice(off_level))
    action = inspected_off.action

    if inspected_off.is_in_coal:

        Acc_part_util = coalition.pay_bribe(inspected_off.hier_id) - hierarchy.inspector.coverup_cost_func(inspected_off.stealing)
        Rej_part_util = reward_func(inspected_off.stealing)

        if Acc_part_util <= Rej_part_util:
            acc_state_util += end(3)
        else:
            acc_state_util += end(4)
        break
    else:
        if action == "NB":
            acc_state_util += end(2)
            break
        if action == "B":
            Acc_part_util = inspected_off.pay_bribe() - hierarchy.inspector.coverup_cost_func(inspected_off.stealing)
            Rej_part_util = reward_func(inspected_off.stealing)
            if Acc_part_util <= Rej_part_util:
                acc_state_util += end(3)
            else:
                acc_state_util += end(4)
            break
        if action == "E":
            while True:
                exposers.append(inspected_off)
                inspected_off = hierarchy.get_boss_of_id(inspected_off.hier_id)
                action = inspected_off.action

                if inspected_off.is_in_coal:
                    Acc_part_util = coalition.pay_bribe(
                        inspected_off.hier_id) - hierarchy.inspector.coverup_cost_func(
                        inspected_off.stealing)
                    Rej_part_util = reward_func(inspected_off.stealing)

                    if Acc_part_util <= Rej_part_util:
                        acc_state_util += end(6)
                    else:
                        acc_state_util += end(7)
                    break

                else:
                    if action == "NB":
```

```python
                                    acc_state_util += end(5)
                                    break
                                if action == "B":
                                    exposers_coverup, exposers_reward, exposers_inspect = calc_coverup_reward_inspect(exposers)
                                    Acc_part_util = inspected_off.pay_bribe(False) - hierarchy.inspector.coverup_cost_func(inspected_off.stealing) - exposers_coverup
                                    Rej_part_util = reward_func(inspected_off.stealing) + exposers_reward
                                    if Acc_part_util <= Rej_part_util:
                                        acc_state_util += end(6)
                                    else:
                                        acc_state_util += end(7)
                                    break
                    break

        if inspected_off is None:
            acc_state_util += end(1)

    LoC = sum(stealing.values()) / init_money
    # End of N cycles, Results
    for official in hierarchy.officials:
        print("{}".format(official.acc_win / N))
    print("{}\n{}\n{}".format(hierarchy.inspector.acc_win / N, acc_state_util / N, LoC))


def run_coals(off_scheme, in_and_out_values, funcs, wages, bribes, coal_scheme_tuples, rules):
    def level_12_official(hier_id, strat, is_in_coal):
        return Official(hier_id=hier_id, wage=wages[0], strategy=strat, kappa=0.3, theta=0.01, is_in_coal=is_in_coal)

    def level_3_official(hier_id, strat, is_in_coal):
        return Official(hier_id=hier_id, wage=wages[1], strategy=strat, kappa=0.6, theta=1, is_in_coal=is_in_coal)

    def build_hier(str1, str2, coal):
        offs = [
            level_3_official((3, 0), str2, ((3, 0) in coal)), level_3_official((3, 1), str2, ((3, 1) in coal)),
            level_12_official((2, 0), str1, ((2, 0) in coal)), level_12_official((2, 1), str1, ((2, 1) in coal)),
            level_12_official((1, 0), str1, ((1, 0) in coal)), level_12_official((1, 1), str1, ((1, 1) in coal))
        ]
        return offs

    for rule in rules:
        for sc_tuple in coal_scheme_tuples:
            print(sc_tuple[0])
            off_hier = build_hier(("Opt", "E", [bribes[2], bribes[2]]), ("Opt", "B", [bribes[0], bribes[1]]), sc_tuple[1])
            inspector = Inspector(70000, inspection_cost_func_example, funcs[0])
            hierarchy = Hierarchy(off_scheme, off_hier, in_and_out_values, inspector)
            coalition = Coalition(scheme_tuple=sc_tuple, hierarchy=hierarchy, rule=rule)
            simulate(N=500000, hierarchy=hierarchy, steal_fine_func=ru_steal_fine,
                    bribe_fine_func=ru_bribe_fine, reward_func=funcs[1], coalition=coalition)


def analyze_sensitivity_B(stealings, a_p, reward_and_coverup_funcs, title):
    # X is zeta, Y is bribe.
    max_st = max(stealings)
    x = np.linspace(1, max_st, 10)
    print(x)
    ys = {}
    for type_funcs in reward_and_coverup_funcs:
        reward_and_coverup_costs = 0
        for stealing in stealings:
            reward_and_coverup_costs += type_funcs[1][0](stealing) + type_funcs[1][1](stealing)

        ys[type_funcs[0]] = reward_and_coverup_costs + x

    for k in ys:
        plt.plot(x, ys[k], label=k)

    plt.hlines(max_st / a_p, 1, max_st, linestyles='dashdot')

    print(max_st / a_p)

    plt.title(title)
    plt.ylabel('Bribe')
    plt.xlabel('O¶')

    plt.xlim(0, max_st)
    plt.ylim(0, max(list(chain.from_iterable([l.tolist() for l in ys.values()])))+100000)

    ax = plt.subplot()
    ax.get_xaxis().set_major_formatter(FuncFormatter(lambda x, p: format(int(x), ',')))
    ax.get_yaxis().set_major_formatter(FuncFormatter(lambda y, p: format(int(y), ',')))

    plt.legend()
    plt.show()

def main():
    coal_scheme_tuples = [
            ('1B1SL0', [(3, 0), (2, 0)],),
            ('1B1SL1', [(3, 0), (2, 1)],),
                ('1B1SR0', [(3, 1), (1, 0)],),
                ('1B1SR1', [(3, 1), (1, 1)],),
                ('BB1SL0', [(3, 0), (2, 0), (3, 1)],),
                ('BB1SL1', [(3, 0), (2, 1), (3, 1)],),
                ('BB1SR0', [(3, 1), (1, 0), (3, 0)],),
                ('BB1SR1', [(3, 1), (1, 1), (3, 0)],),
                ('1B2SL', [(3, 0), (2, 0), (2, 1)],),
                ('1B2SR', [(3, 1), (1, 0), (1, 1)],),
                ('2SBBL', [(3, 0), (3, 1), (2, 0), (2, 1)],),
                ('2SBBR', [(3, 0), (3, 1), (1, 0), (1, 1)],),
                ('1SBB1S0', [(2, 0), (3, 0), (3, 1), (1, 0)],),
                ('1SBB1S1', [(2, 0), (3, 0), (3, 1), (1, 1)],),
                ('1SBB1S2', [(2, 1), (3, 0), (3, 1), (1, 0)],),
                ('1SBB1S3', [(2, 1), (3, 0), (3, 1), (1, 1)],),
                ('2SBBL0', [(3, 0), (2, 0), (2, 1), (3, 1), (1, 0)],),
                ('2SBBL1', [(3, 0), (2, 0), (2, 1), (3, 1), (1, 1)],),
                ('2SBBR0', [(3, 1), (1, 0), (1, 1), (3, 0), (2, 0)],),
                ('2SBBR1', [(3, 1), (1, 0), (1, 1), (3, 0), (2, 1)],),
                ('GC', [(2, 0), (2, 1), (3, 0), (3, 1), (1, 0), (1, 1)])]

    # coal_scheme_tuples = [ ('1B1SR0', [(3, 1), (1, 0)],) ,]

    off_scheme = {
        (4, 0): ((3, 0), (3, 1)),
        (3, 0): ((2, 0), (2, 1)),
        (3, 1): ((1, 0), (1, 1)),
    }
    in_and_out_values = {
        ((3, 0), (3, 1)): (3000000, 2000000),
        ((2, 0), (2, 1)): (2000000 / 2, 750000),
        ((1, 0), (1, 1)): (2000000 / 2, 750000)
    }

    ch, b, s = 0, 1, 2

    W = [0, 90000, 40000]
    S = [0, 500000, 125000]

    d = [131251, 86251, 45001]
    s1 = [1384616.385, 1304616.385, 80001]
    s2 = [3080001, 3000001, 80001]
    s3 = [8750000.976, 5750001, 3000000.976]

    z1 = (500000, 395000, 105001)
    z3 = (500000, 375001, 125000)

    B = d

    rules = (EQ_rule, SS_with_xi(1))
    # rules = (SS_with_xi(1),)
    # rules = (EQ_rule,)

    no_coal = [("None", [],)]

    # run_coals(off_scheme=off_scheme, in_and_out_values=in_and_out_values, funcs=(
        coverup_cost_func_def, reward_func_def),
    #             wages=[W[s], W[b]], bribes=B, coal_scheme_tuples=no_coal, rules=rules)

    a = [0, 0.5, 0.416666667, 0.333333333]
    a_eff = [0, (1 - a[3]) * (1 - a[2]) * a[1], (1 - a[3]) * a[2], a[3]]
    a_0_eff_i = [0, 0.041666667, 0.076388889, 0]
    print(a_eff)

    types_and_funcs = [("def", [coverup_cost_func_def, reward_func_def]),
                       ("s1", [coverup_cost_func_s1, reward_func_s1]),
                       ("s2", [coverup_cost_func_s2, reward_func_s2]),
                       ("s3", [coverup_cost_func_s3, reward_func_s3]),]

    types_and_funcs_z = [("z1", [coverup_cost_func_z1, reward_func_z1]),
                         ("z3", [coverup_cost_func_z3, reward_func_z3]),]

    analyze_sensitivity_B([S[b], S[s]], a_eff[3]/2 + min(a_eff[1], a_eff[2]),
        types_and_funcs, "Chain")
    analyze_sensitivity_B([S[b]], a_eff[3]/2, types_and_funcs, "Boss")
    analyze_sensitivity_B([S[s]], min(a_0_eff_i[1], a_0_eff_i[2]), types_and_funcs, "
        Subordinate only")


if __name__ == "__main__":
    main()
```

Appendix D. Table of coalitional payoffs in the example graph

Table 3.1: Values of all coalitions for Myerson/Theirson.

#	(3,0)	(3,1)	(2,0)	(2,1)	(1,0)	(1,1)	v(?)	Fully formable?
1	0	0	0	0	0	1	$\{(1,1)\}$	TRUE
2	0	0	0	0	1	0	$\{(1,0)\}$	TRUE
3	0	0	0	0	1	1	$\{(1,0),(1,1)\}$	TRUE
4	0	0	0	1	0	0	$\{(2,1)\}$	TRUE
5	0	0	0	1	0	1	$\{(2,1)\}+\{(1,1)\}$	FALSE
6	0	0	0	1	1	0	$\{(2,1)\}+\{(1,0)\}$	FALSE
7	0	0	0	1	1	1	$\{(2,1)\}+\{(1,0),(1,1)\}$	FALSE
8	0	0	1	0	0	0	$\{(2,0)\}$	TRUE
9	0	0	1	0	0	1	$\{(2,0)\}+\{(1,1)\}$	FALSE
10	0	0	1	0	1	0	$\{(2,0)\}+\{(1,0)\}$	FALSE
11	0	0	1	0	1	1	$\{(2,0)\}+\{(1,0),(1,1)\}$	FALSE
12	0	0	1	1	0	0	$\{(2,0),(2,1)\}$	TRUE
13	0	0	1	1	0	1	$\{(2,0),(2,1)\}+\{(1,1)\}$	FALSE
14	0	0	1	1	1	0	$\{(2,0),(2,1)\}+\{(1,0)\}$	FALSE
15	0	0	1	1	1	1	$\{(2,0),(2,1)\}+\{(1,0),(1,1)\}$	FALSE
16	0	1	0	0	0	0	$\{(3,1)\}$	TRUE
17	0	1	0	0	0	1	$\{(3,1),(1,1)\}$	TRUE
18	0	1	0	0	1	0	$\{(3,1),(1,0)\}$	TRUE
19	0	1	0	0	1	1	$\{(3,1),(1,0),(1,1)\}$	TRUE
20	0	1	0	1	0	0	$\{(3,1)\}+\{(2,1)\}$	FALSE
21	0	1	0	1	0	1	$\{(3,1),(1,1)\}+\{(2,1)\}$	FALSE
22	0	1	0	1	1	0	$\{(3,1),(1,0)\}+\{(2,1)\}$	FALSE
23	0	1	0	1	1	1	$\{(3,1),(1,0),(1,1)\}+\{(2,1)\}$	FALSE
24	0	1	1	0	0	0	$\{(3,1)\}+\{(2,0)\}$	FALSE
25	0	1	1	0	0	1	$\{(3,1),(1,1)\}+\{(2,0)\}$	FALSE
26	0	1	1	0	1	0	$\{(3,1),(1,0)\}+\{(2,0)\}$	FALSE
27	0	1	1	0	1	1	$\{(3,1),(1,0),(1,1)\}+\{(2,0)\}$	FALSE
28	0	1	1	1	0	0	$\{(3,1)\}+\{(2,0),(2,1)\}$	FALSE
29	0	1	1	1	0	1	$\{(3,1),(1,1)\}+\{(2,0),(2,1)\}$	FALSE
30	0	1	1	1	1	0	$\{(3,1),(1,0)\}+\{(2,0),(2,1)\}$	FALSE
31	0	1	1	1	1	1	$\{(3,1),(1,0),(1,1)\}+\{(2,0),(2,1)\}$	FALSE
32	1	0	0	0	0	0	$\{(3,0)\}$	TRUE
33	1	0	0	0	0	1	$\{(3,0)\}+\{(1,1)\}$	FALSE
34	1	0	0	0	1	0	$\{(3,0)\}+\{(1,0)\}$	FALSE
35	1	0	0	0	1	1	$\{(3,0)\}+\{(1,0),(1,1)\}$	FALSE
36	1	0	0	1	0	0	$\{(3,0),(2,1)\}$	TRUE
37	1	0	0	1	0	1	$\{(3,0),(2,1)\}+\{(1,1)\}$	FALSE
38	1	0	0	1	1	0	$\{(3,0),(2,1)\}+\{(1,0)\}$	FALSE
39	1	0	0	1	1	1	$\{(3,0),(2,1)\}+\{(1,0),(1,1)\}$	FALSE
40	1	0	1	0	0	0	$\{(3,0),(2,0)\}$	TRUE
41	1	0	1	0	0	1	$\{(3,0),(2,0)\}+\{(1,1)\}$	FALSE
42	1	0	1	0	1	0	$\{(3,0),(2,0)\}+\{(1,0)\}$	FALSE
43	1	0	1	0	1	1	$\{(3,0),(2,0)\}+\{(1,0),(1,1)\}$	FALSE
44	1	0	1	1	0	0	$\{(3,0),(2,0),(2,1)\}$	TRUE
45	1	0	1	1	0	1	$\{(3,0),(2,0),(2,1)\}+\{(1,1)\}$	FALSE
46	1	0	1	1	1	0	$\{(3,0),(2,0),(2,1)\}+\{(1,0)\}$	FALSE
47	1	0	1	1	1	1	$\{(3,0),(2,0),(2,1)\}+\{(1,0),(1,1)\}$	FALSE
48	1	1	0	0	0	0	$\{(3,0),(3,1)\}$	TRUE
49	1	1	0	0	0	1	$\{(3,0),(3,1),(1,1)\}$	TRUE
50	1	1	0	0	1	0	$\{(3,0),(3,1),(1,0)\}$	TRUE
51	1	1	0	0	1	1	$\{(3,0),(3,1),(1,0),(1,1)\}$	TRUE
52	1	1	0	1	0	0	$\{(3,0),(3,1),(2,1)\}$	TRUE
53	1	1	0	1	0	1	$\{(3,0),(3,1),(2,1),(1,1)\}$	TRUE
54	1	1	0	1	1	0	$\{(3,0),(3,1),(2,1),(1,0)\}$	TRUE
55	1	1	0	1	1	1	$\{(3,0),(3,1),(2,1),(1,0),(1,1)\}$	TRUE
56	1	1	1	0	0	0	$\{(3,0),(3,1),(2,0)\}$	TRUE
57	1	1	1	0	0	1	$\{(3,0),(3,1),(2,0),(1,1)\}$	TRUE
58	1	1	1	0	1	0	$\{(3,0),(3,1),(2,0),(1,0)\}$	TRUE
59	1	1	1	0	1	1	$\{(3,0),(3,1),(2,0),(1,0),(1,1)\}$	TRUE
60	1	1	1	1	0	0	$\{(3,0),(3,1),(2,0),(2,1)\}$	TRUE
61	1	1	1	1	0	1	$\{(3,0),(3,1),(2,0),(2,1),(1,1)\}$	TRUE
62	1	1	1	1	1	0	$\{(3,0),(3,1),(2,0),(2,1),(1,0)\}$	TRUE
63	1	1	1	1	1	1	GC	TRUE

Appendix E. Code listing for the Myerson value caluclation

```
from math import factorial

coals = [0, frozenset([(1, 1)]), frozenset([(1, 0)]), frozenset([(1, 0), (1, 1)]), frozenset([(2, 1)]),
            frozenset([(2, 1), (1, 1)]), frozenset([(2, 1), (1, 0)]), frozenset([(2, 1), (1, 0), (1, 1)]),
            frozenset([(2, 0)]), frozenset([(2, 0), (1, 1)]), frozenset([(2, 0), (1, 0)]),
            frozenset([(2, 0), (1, 0), (1, 1)]), frozenset([(2, 0), (2, 1)]), frozenset([(2, 0), (2, 1), (1, 1)]),
            frozenset([(2, 0), (2, 1), (1, 0)]), frozenset([(2, 0), (2, 1), (1, 0), (1, 1)]), frozenset([(3, 1)]),
            frozenset([(3, 1), (1, 1)]), frozenset([(3, 1), (1, 0)]), frozenset([(3, 1), (1, 0), (1, 1)]),
            frozenset([(3, 1), (2, 1)]), frozenset([(3, 1), (1, 1), (2, 1)]), frozenset([(3, 1), (1, 0), (2, 1)]),
            frozenset([(3, 1), (1, 0), (1, 1), (2, 1)]), frozenset([(3, 1), (2, 0)]),
            frozenset([(3, 1), (1, 1), (2, 0)]),
            frozenset([(3, 1), (1, 0), (2, 0)]), frozenset([(3, 1), (1, 0), (1, 1), (2, 0)]),
            frozenset([(3, 1), (2, 0), (2, 1)]), frozenset([(3, 1), (1, 1), (2, 0), (2, 1)]),
            frozenset([(3, 1), (1, 0), (2, 0), (2, 1)]), frozenset([(3, 1), (1, 0), (1, 1), (2, 0), (2, 1)]),
            frozenset([(3, 0)]), frozenset([(3, 0), (1, 1)]), frozenset([(3, 0), (1, 0)]),
            frozenset([(3, 0), (1, 0), (1, 1)]), frozenset([(3, 0), (2, 1)]), frozenset([(3, 0), (2, 1), (1, 1)]),
            frozenset([(3, 0), (2, 1), (1, 0)]), frozenset([(3, 0), (2, 1), (1, 0), (1, 1)]),
            frozenset([(3, 0), (2, 0)]),
            frozenset([(3, 0), (2, 0), (1, 1)]), frozenset([(3, 0), (2, 0), (1, 0)]),
            frozenset([(3, 0), (2, 0), (1, 0), (1, 1)]), frozenset([(3, 0), (2, 0), (2, 1)]),
            frozenset([(3, 0), (2, 0), (2, 1), (1, 1)]), frozenset([(3, 0), (2, 0), (2, 1), (1, 0)]),
            frozenset([(3, 0), (2, 0), (2, 1), (1, 0), (1, 1)]), frozenset([(3, 0), (3, 1)]),
            frozenset([(3, 0), (3, 1), (1, 1)]), frozenset([(3, 0), (3, 1), (1, 0)]),
            frozenset([(3, 0), (3, 1), (1, 0), (1, 1)]), frozenset([(3, 0), (3, 1), (2, 1)]),
            frozenset([(3, 0), (3, 1), (2, 1), (1, 1)]), frozenset([(3, 0), (3, 1), (2, 1), (1, 0)]),
            frozenset([(3, 0), (3, 1), (2, 1), (1, 0), (1, 1)]), frozenset([(3, 0), (3, 1), (2, 0)]),
            frozenset([(3, 0), (3, 1), (2, 0), (1, 1)]), frozenset([(3, 0), (3, 1), (2, 0), (1, 0)]),
            frozenset([(3, 0), (3, 1), (2, 0), (1, 0), (1, 1)]), frozenset([(3, 0), (3, 1), (2, 0), (2, 1)]),
             frozenset([(3, 0), (3, 1), (2, 0), (2, 1), (1, 1)]), frozenset([(3, 0), (3, 1), (2, 0), (2, 1), (1, 0)]),
             frozenset([(3, 0), (3, 1), (2, 0), (2, 1), (1, 0), (1, 1)])]

whole_coals_ids = [1, 2, 3, 4, 8, 12, 16, 17, 18, 19, 32, 36, 40, 44, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
                       59, 60, 61, 62, 63]

def calc_vals(whole_val):
    coal_vals = {}

    for i in whole_coals_ids:
        coal_vals[coals[i]] = whole_val[i]

    coal_vals[coals[5]] = coal_vals[coals[4]] + coal_vals[coals[1]]
    coal_vals[coals[6]] = coal_vals[coals[4]] + coal_vals[coals[2]]
    coal_vals[coals[7]] = coal_vals[coals[4]] + coal_vals[coals[3]]
    coal_vals[coals[9]] = coal_vals[coals[8]] + coal_vals[coals[1]]
    coal_vals[coals[10]] = coal_vals[coals[8]] + coal_vals[coals[2]]
    coal_vals[coals[11]] = coal_vals[coals[8]] + coal_vals[coals[3]]
    coal_vals[coals[13]] = coal_vals[coals[12]] + coal_vals[coals[1]]
    coal_vals[coals[14]] = coal_vals[coals[12]] + coal_vals[coals[2]]
    coal_vals[coals[15]] = coal_vals[coals[12]] + coal_vals[coals[3]]
    coal_vals[coals[20]] = coal_vals[coals[16]] + coal_vals[coals[4]]
    coal_vals[coals[21]] = coal_vals[coals[17]] + coal_vals[coals[4]]
    coal_vals[coals[22]] = coal_vals[coals[18]] + coal_vals[coals[4]]
    coal_vals[coals[23]] = coal_vals[coals[19]] + coal_vals[coals[4]]
    coal_vals[coals[24]] = coal_vals[coals[16]] + coal_vals[coals[8]]
    coal_vals[coals[25]] = coal_vals[coals[17]] + coal_vals[coals[8]]
    coal_vals[coals[26]] = coal_vals[coals[18]] + coal_vals[coals[8]]
    coal_vals[coals[27]] = coal_vals[coals[19]] + coal_vals[coals[8]]
    coal_vals[coals[28]] = coal_vals[coals[16]] + coal_vals[coals[12]]
    coal_vals[coals[29]] = coal_vals[coals[17]] + coal_vals[coals[12]]
    coal_vals[coals[30]] = coal_vals[coals[18]] + coal_vals[coals[12]]
    coal_vals[coals[31]] = coal_vals[coals[19]] + coal_vals[coals[12]]
    coal_vals[coals[33]] = coal_vals[coals[32]] + coal_vals[coals[1]]
    coal_vals[coals[34]] = coal_vals[coals[32]] + coal_vals[coals[2]]
    coal_vals[coals[35]] = coal_vals[coals[32]] + coal_vals[coals[3]]
    coal_vals[coals[37]] = coal_vals[coals[36]] + coal_vals[coals[1]]
    coal_vals[coals[38]] = coal_vals[coals[36]] + coal_vals[coals[2]]
    coal_vals[coals[39]] = coal_vals[coals[36]] + coal_vals[coals[3]]
    coal_vals[coals[41]] = coal_vals[coals[40]] + coal_vals[coals[1]]
    coal_vals[coals[42]] = coal_vals[coals[40]] + coal_vals[coals[2]]
    coal_vals[coals[43]] = coal_vals[coals[40]] + coal_vals[coals[3]]
    coal_vals[coals[45]] = coal_vals[coals[44]] + coal_vals[coals[1]]
    coal_vals[coals[46]] = coal_vals[coals[44]] + coal_vals[coals[2]]
    coal_vals[coals[47]] = coal_vals[coals[44]] + coal_vals[coals[3]]
    # todo chains or whatnot

    offs = [(3, 0), (3, 1), (2, 0), (2, 1), (1, 0), (1, 1)]
    H = len(offs)

    myerson_vec = {}
    for off in offs:
        myerson_vec[off] = 0
        for coal in coal_vals.keys():
            if off not in coal and off != coal:
                S = len(coal)
                myerson_vec[off] += factorial(S) * factorial(H - 1 - S) / factorial(H) * (
                            coal_vals[coal.union(frozenset([off]))] - coal_vals[coal])

    return myerson_vec


def check_conv(whole_val):
    coal_vals = {}

    for i in whole_coals_ids:
        coal_vals[coals[i]] = whole_val[i]

    coal_vals[coals[5]] = coal_vals[coals[4]] + coal_vals[coals[1]]
    coal_vals[coals[6]] = coal_vals[coals[4]] + coal_vals[coals[2]]
    coal_vals[coals[7]] = coal_vals[coals[4]] + coal_vals[coals[3]]
    coal_vals[coals[9]] = coal_vals[coals[8]] + coal_vals[coals[1]]
    coal_vals[coals[10]] = coal_vals[coals[8]] + coal_vals[coals[2]]
    coal_vals[coals[11]] = coal_vals[coals[8]] + coal_vals[coals[3]]
    coal_vals[coals[13]] = coal_vals[coals[12]] + coal_vals[coals[1]]
    coal_vals[coals[14]] = coal_vals[coals[12]] + coal_vals[coals[2]]
    coal_vals[coals[15]] = coal_vals[coals[12]] + coal_vals[coals[3]]
    coal_vals[coals[20]] = coal_vals[coals[16]] + coal_vals[coals[4]]
    coal_vals[coals[21]] = coal_vals[coals[17]] + coal_vals[coals[4]]
    coal_vals[coals[22]] = coal_vals[coals[18]] + coal_vals[coals[4]]
    coal_vals[coals[23]] = coal_vals[coals[19]] + coal_vals[coals[4]]
    coal_vals[coals[24]] = coal_vals[coals[16]] + coal_vals[coals[8]]
    coal_vals[coals[25]] = coal_vals[coals[17]] + coal_vals[coals[8]]
    coal_vals[coals[26]] = coal_vals[coals[18]] + coal_vals[coals[8]]
    coal_vals[coals[27]] = coal_vals[coals[19]] + coal_vals[coals[8]]
    coal_vals[coals[28]] = coal_vals[coals[16]] + coal_vals[coals[12]]
    coal_vals[coals[29]] = coal_vals[coals[17]] + coal_vals[coals[12]]
    coal_vals[coals[30]] = coal_vals[coals[18]] + coal_vals[coals[12]]
    coal_vals[coals[31]] = coal_vals[coals[19]] + coal_vals[coals[12]]
    coal_vals[coals[33]] = coal_vals[coals[32]] + coal_vals[coals[1]]
    coal_vals[coals[34]] = coal_vals[coals[32]] + coal_vals[coals[2]]
    coal_vals[coals[35]] = coal_vals[coals[32]] + coal_vals[coals[3]]
    coal_vals[coals[37]] = coal_vals[coals[36]] + coal_vals[coals[1]]
    coal_vals[coals[38]] = coal_vals[coals[36]] + coal_vals[coals[2]]
    coal_vals[coals[39]] = coal_vals[coals[36]] + coal_vals[coals[3]]
    coal_vals[coals[41]] = coal_vals[coals[40]] + coal_vals[coals[1]]
    coal_vals[coals[42]] = coal_vals[coals[40]] + coal_vals[coals[2]]
    coal_vals[coals[43]] = coal_vals[coals[40]] + coal_vals[coals[3]]
    coal_vals[coals[45]] = coal_vals[coals[44]] + coal_vals[coals[1]]
    coal_vals[coals[46]] = coal_vals[coals[44]] + coal_vals[coals[2]]
    coal_vals[coals[47]] = coal_vals[coals[44]] + coal_vals[coals[3]]

    C = True
    (y, n) = (0, 0)
    for S in coal_vals.keys():
        for T in coal_vals.keys():
            if S.intersection(T) == frozenset():
                inter = 0
            else:
                inter = coal_vals[S.intersection(T)]

            test = (coal_vals[S] + coal_vals[T] <= coal_vals[S.union(T)] + inter)
            if test:
                y = y + 1
            else:
                n = n + 1

            C = C & test

    # There are len(coal_vals.keys()) tests for S=T that return True.
    return C, y-len(coal_vals.keys()), n


def main():
    ch, b, s = 0, 1, 2

    W = [0, 90000, 40000]
    S = [0, 500000, 125000]

    d = [131251, 86251, 45001]
    s1 = [1384616.385, 1304616.385, 80001]
    s2 = [3080001, 3000001, 80001]
    s3 = [8750000.976, 5750001, 3000000.976]

    z1 = (500000, 395000, 105001)
    z3 = (500000, 375001, 125000)

    B = z3

    a = [0, 0.5, 0.416666667, 0.333333333]
    a_eff = [0, (1 - a[3]) * (1 - a[2]) * a[1], (1 - a[3]) * a[2], a[3]]

    # [0, 0.19444444443055556, 0.2777777781388889, 0.333333333]

    m_1B1SR = S[b] + S[s] - ((a[3] / 2 + a_eff[1] / 2) * B[ch] + a_eff[1] / 2 * B[s])
    m_1B1SL = S[b] + S[s] - ((a[3] / 2 + a_eff[2] / 2) * B[ch] + a_eff[2] / 2 * B[s])
    m_BB1SR = 2 * S[b] + S[s] - ((a[3] + a_eff[2] + a_eff[1] / 2) * B[ch] + a_eff[1] / 2
            * B[s])
    m_BB1SL = 2 * S[b] + S[s] - ((a[3] + a_eff[2] / 2 + a_eff[1]) * B[ch] + a_eff[2] / 2
            * B[s])
    m_1SBB1S = 2 * S[b] + 2 * S[s] - (
                (a[3] + a_eff[2] / 2 + a_eff[1] / 2) * B[ch] + (a_eff[2] / 2 + a_eff[1] /
                    2) * B[s])
    m_2SBB1SR = 2 * S[b] + 3 * S[s] - (
            (a[3] / 2 + a_eff[1] / 2) * B[ch] + a[3] / 2 * B[b] + (a_eff[2] + a_eff[1] /
                2) * B[s])
    m_2SBB1SL = 2 * S[b] + 3 * S[s] - (
            (a[3] / 2 + a_eff[2] / 2) * B[ch] + a[3] / 2 * B[b] + (a_eff[2] / 2 + a_eff
                [1]) * B[s])

    myerson_vals = {1: S[s],
                    2: S[s],
                    3: 2 * S[s] - a_eff[1] * B[s],
                    4: S[s],
                    8: S[s],
                    12: 2 * S[s] - a_eff[2] * B[s],
                    16: S[b] - (a[3] / 2 + a_eff[1]) * B[ch],
                    17: m_1B1SR,
                    18: m_1B1SR,
                    19: S[b] + 2 * S[s] - (a[3] / 2 * B[b] + a_eff[1] * B[s]),
                    32: S[b] - (a[3] / 2 + a_eff[2]) * B[ch],
                    36: m_1B1SL,
                    40: m_1B1SL,
                    44: S[b] + 2 * S[s] - (a[3] / 2 * B[b] + a_eff[2] * B[s]),
                    48: 2 * S[b] - (a[3] + a_eff[2] + a_eff[1]) * B[ch],
                    49: m_BB1SR,
                    50: m_BB1SR,
                    51: 2 * S[b] + 2 * S[s] - ((a[3] / 2 + a_eff[2]) * B[ch] + a[3] / 2 *
                        B[b] + a_eff[1] * B[s]),
                    52: m_BB1SL,
                    53: m_1SBB1S,
                    54: m_1SBB1S,
                    55: m_2SBB1SL,
                    56: m_BB1SL,
                    57: m_1SBB1S,
                    58: m_1SBB1S,
                    59: m_2SBB1SL,
                    60: 2 * S[b] + 2 * S[s] - ((a[3] / 2 + a_eff[1]) * B[ch] + a[3] / 2 *
                        B[b] + a_eff[2] * B[s]),
                    61: m_2SBB1SR,
                    62: m_2SBB1SR,
                    63: 2 * S[b] + 4 * S[s] - (a[3] * B[b] + (a_eff[2] + a_eff[1]) * B[s
                        ])}

    t_R = S[s] - a_eff[1] / 2 * B[s]
    t_L = S[s] - a_eff[2] / 2 * B[s]
    t_1B1SR = S[b] + S[s] - (a[3] / 2 * B[b] + a_eff[1] / 2 * B[s])
    t_1B1SL = S[b] + S[s] - (a[3] / 2 * B[b] + a_eff[2] / 2 * B[s])
    t_BB1SR = 2 * S[b] + S[s] - (a[3] * B[b] + a_eff[1] / 2 * B[s])
    t_BB1SL = 2 * S[b] + S[s] - (a[3] * B[b] + a_eff[2] / 2 * B[s])
    t_1SBB1S = 2 * S[b] + 2 * S[s] - (a[3] * B[b] + (a_eff[2] / 2 + a_eff[1] / 2) * B[s])
    t_2SBB1SL = 2 * S[b] + 3 * S[s] - (a[3] * B[b] + (a_eff[1] / 2 + a_eff[2]) * B[s])
    t_2SBB1SR = 2 * S[b] + 3 * S[s] - (a[3] * B[b] + (a_eff[1] + a_eff[2] / 2) * B[s])

    theirson_vals = {1: t_R,
                     2: t_R,
                     3: myerson_vals[3],
                     4: t_L,
                     8: t_L,
                     12: myerson_vals[12],
                     16: S[b] - a[3] / 2 * B[s],
                     17: t_1B1SR,
                     18: t_1B1SR,
                     19: myerson_vals[19],
                     32: S[b] - a[3] / 2 * B[s],
                     36: t_1B1SL,
                     40: t_1B1SL,
                     44: myerson_vals[44],
                     48: 2 * S[b] - a[3] * B[b],
                     49: t_BB1SR,
                     50: t_BB1SR,
                     51: 2 * S[b] + 2 * S[s] - (a[3] * B[b] + a_eff[1] * B[s]),
                     52: t_BB1SL,
                     53: t_1SBB1S,
                     54: t_1SBB1S,
                     55: t_2SBB1SL,
                     56: t_BB1SL,
                     57: t_1SBB1S,
                     58: t_1SBB1S,
                     59: t_2SBB1SL,
                     60: 2 * S[b] + 2 * S[s] - (a[3] * B[b] + a_eff[2] * B[s]),
                     61: t_2SBB1SR,
                     62: t_2SBB1SR,
                     63: myerson_vals[63]}

    def print_it(vec):
        offs = [(3, 0), (3, 1), (2, 0), (2, 1), (1, 0), (1, 1)]
        for off in offs:
            print("{}\t{}".format(off, vec[off]))

    my = calc_vals(myerson_vals)
    print_it(my)
    print(check_conv(myerson_vals))
    print("\n")
    th = calc_vals(theirson_vals)
    print_it(th)
    print(check_conv(theirson_vals))
```

```python


if __name__ == "__main__":
    main()
```