Soft Play Detection in Shooter Games Using Hit Matrix Analysis

Jussi Laasonen and Jouni Smed
Department of Information Technology
University of Turku
Turku, Finland
juvila@utu.fi, jouni.smed@utu.fi

***Abstract*—Soft play is a form of cheating where players deliberately play easy against each other. We evaluate different methods for detecting the players engaging in soft play in shooter games using data generated with synthetic players. These methods are used when analysing the hit matrix of the game.**

***Keywords*—cheating, collusion, community detection, e-sports, graph clustering, multiplayer games, principal component analysis, soft play.**

I. Introduction

Soft play is a form of collusion—an attempt to co-operate covertly—where a group of players deliberately plays easy against each other to gain an advantage. For example, colluding players can save their resources by withholding attacks on one another and focus on the other players instead. When prohibited by game rules, collusion is considered cheating and it is especially harmful in e-sports tournaments—more so when real money prizes or bets are involved—but it can also ruin the game in a more casual environment.

Collusion in games has been studied mainly in card games [1], [2], [3], [4]. Van der Knyff et al. [5] try to find the colluding subset in a first-person-shooter game using principal component analysis [6]. Outside the field of multiplayer games, collusion has been addressed in tournaments [7], multiple choice examinations [8], covert communication channels [9], stock market trading [10], grid computing [11], social moderation [12] and spectrum auctions [13].

This paper continues our research to create swift and accurate collusion detection methods [14], [15]. In our previous work, we examined collusion features [16] and tried to detect area-of-interest, tabu-list sharing and blocking colluders using decision tree classifiers in a simple two-dimensional game [17]. In this paper, we try to detect players engaging in soft play with graph clustering algorithms. We build a directed graph from the hit matrix of a game using the players as vertices and calculating the edge weights based on the amount of hits. We then try to find clusters of nodes with low connecting edge weights.

In the next section, we define the concept of hit matrix and how it used to generate the graphs, followed by the clustering algorithms. In Section 3 we present effects of collusion in our test cases and results of the clustering algorithms. Finally, in Section 4 we provide a conclusion of the work and our plans for future research on the subject.

II. Methods

A hit matrix contains data on how much damage the players have inflicted to each other during the game. Each cell of the matrix contains the amount of hits the row player has inflicted on the column player. Table I provides an example hit matrix from a hypothetical game of three players. Player A has hit player B three times while B has not hit the A at all. B has inflicted one hit to himself due environmental damage (e.g. falling or fire) or splash damage from his weapon.

TABLE I. An Example Hit Matrix

	A	B	C
A	0	3	2
B	0	1	3
C	2	1	0

To measure how well a player did during the game different values can be calculated from the hit matrix. The simplest are the total number of hits the player got or inflicted (or deaths and kills). Games often use an aggregated value like kill-death-spread (KD-spread) or kill-death-ratio (KD-ratio) to measure the performance of a player. KD-spread is the difference of kills and deaths of a player, and KD-ratio is kills divided by deaths. If the player did not die a single death KD-ratio is the amount of kills. In Table I player A has five kills, two deaths, 3 KD-spread and 2.5 KD-ratio.

A. Principal Component Analysis

Van der Knyff et al. [5] use principal component analysis [6] to detect the colluding players. They argue that the hit matrix row of a colluder would correlate with different principal components than a row of a non-colluder. While van der Knyff et al. had successfully applied the method to team recognition, they failed to recognize the colluding players.

Principal component analysis is a method of dimensionality reduction where the original data set of correlated variables into to a set of values of principal components. The principal components are calculated from the data so that the first component will have the maximum variance possible, and each subsequent component will have maximum variance given it is uncorrelated with the other components.

We argue that, even if they failed to recognize the colluders, the assumption is theoretically sound. We repeat the experiment using Pakuhaku game and synthetic players. We start with a simple setup to see, if a correlation exists at all

INTETAIN 2015, June 10-12, Torino, Italy

DOI 10.4108/icst.intetain.2015.259565

and then add more complexity to see whether the correlation decreases.

Let G bet the set of games in the test data. For each game $g_n \in G$, we calculate principal components $pc_{n,m}$ of the hit matrix. For each player $p_i \in P$, we calculate correlations $\rho_{n,m,i}$ of the hit matrix row of the player and the principal component. Then we build a linear model [18]

$$\rho_{m,n,i} = \beta_0 + \beta_1 collusion_{n,i} + \beta_2 strategy_{n,i} + \beta_3 team_{n,i} + \beta_4 skill_{n,i} + \epsilon \quad (1)$$

and calculate the p-value $\rho_{m,n,i}$ of the regression coefficient β_1 of the collusion $collusion_{n,i}$ term to see if collusion has a significant relationship with correlation.

B. Graph Clustering

Palshikar and Apte propose several graph clustering algorithms for finding a collusion set from stock market data [10]. The idea is to create a directed graph where the vertices represent the traders, and the edges represent the trades. The weight of an edge is the amount of trades made between the respective traders, and the direction denotes the buyer. Then the task of finding the collusion set reduces to finding clusters with high internal trade. We use the three clustering algorithms presented by Palshikar and Apte, and a community detection algorithm from igraph R package [19].

1) Shared nearest neighbour graph clustering: algorithm (shared_NN) starts with each vertex as a singleton cluster.It loops through all pairs of vertices and combines the respective clusters if the vertices are in the set of each others k-nearest neighbours and they share at least kt k-nearest neighbours.

2) Mutual nearest neighbour graph clustering: algorithm (mutual_NN_average) starts with each vertex in a singleton cluster and combines the clusters with minimum mutual neighbourhood value mnv, while more than m clusters remain or minimum mnv reaches the maximum. The mnv of two vertices is the sum of their rank in each others k-nearest neighbours and the mnv of two clusters is the average mnv of all pairs of vertices in the combined cluster.

3) Collusion clustering: algorithm is specially designed to detect collusion sets. It starts with each vertex as singleton cluster and merges two compatible clusters with the largest collusion value until no more compatible clusters exist. Two clusters are compatible if at least h per cent in each is compatible with the other cluster. A vertex is compatible with a set of vertices if its k nearest neighbours contain at least m or all vertices in the cluster.

4) Community detection: We use the optimal.community algorithm, which calculates the optimal partitioning of a graph based on the maximum the modularity measure over all possible partitions [20]. The algorithm requires an undirected graph, and we convert the graph to an undirected graph by collapsing edges.

5) Constructing the graph: Before we can apply the clustering algorithm, we need to construct the graph from the hit matrix. When using hit matrix values as weights, the problem is different from finding the colluding traders, and the colluding subset is loosely connected. To overcome this, we invert the definition of nearness. The nearest neighbour of a player is the player who has the fewest hits. To allow this definition to hold we add one to each hit matrix value. Otherwise, the relationship would be lost when players have zero mutual hits.

Before applying the clustering algorithm, the graph is pruned by retaining only k nearest neighbours of any vertex using the algorithm described in [10].

Instead of using hit matrix values directly, we can use collusion scores from Mazrooei et al. [4], [21] as edge weights. The collusion scores are based on a structure called collusion table, which shows the effect of each player to the utility of all players. A cell $C_g(A, B)$ in a collusion table contains the effect of A's actions to B's utility. We convert the hit matrix to a collusion table by negating the values and using the total hits scored by the players as a diagonal. Table II shows an example of a collusion table generated from the hit matrix of Table I.

TABLE II. AN EXAMPLE COLLUSION TABLE BASED ON THE HIT MATRIX FROM TABLE I

	A	B	C
A	5	-3	-2
B	0	2	-3
C	-2	-1	3

To calculate the total impact we let

$$\vartheta_g^{TI}(a,b) = \sum_{i\in\{a,b\}} \sum_{j\in\{a,b\}} C_g(i,j), \quad (2)$$

the marginal impact

$$\vartheta_g^{MaI}(a,b) = \left(C_g(b,a) - \frac{1}{|N|-2} \sum_{\substack{i\in P_g \\ i\notin\{a,b\}}} C_g(i,a) \right) + \left(C_g(a,b) - \frac{1}{|N|-2} \sum_{\substack{j\in P_g \\ j\notin\{a,b\}}} C_g(j,b) \right), \quad (3)$$

the mutual impact

$$\vartheta_g^{MuI}(a,b) = C_g(a,b) + c_g(b,a), \quad (4)$$

the minimum impact

$$\vartheta_g^{MiI}(a,b) = min_{i\in\{a,b\}} \left\{ \sum_{j\in\{a,b\}} C_g(j,i)) \right\}, \quad (5)$$

and the differential total impact

$$\vartheta_g^{DI}(a,b) = \vartheta^{TI}(a,b) - max\left\{ \underset{\substack{d\in P_g \\ d\notin\{a,b\}}}{max} \left\{\vartheta^{TI}(a,d)\right\}, \underset{\substack{d\in P_g \\ d\notin\{a,b\}}}{max} \left\{\vartheta^{TI}(b,d)\right\} \right\}. \quad (6)$$

The collusion scores have higher values for a colluder, and the algorithms need not to be altered. However, the values can be negative, and we can add $min(C_g)+1$ to all values when converting them to edge weights.

C. Detecting the Colluding Set

Having a clustering algorithm to find out the subset consisting of the colluding players is not enough. We need to select a correct subset from all the subsets produced by the algorithm. Depending on the algorithm this can be the largest or the smallest cluster, but using such a simple criterion will miss some of the correct subsets. Options for selecting the colluding subset can be based in-degree/out-degree [12] or the collusion index $\Phi(C) = I(C)/E(C)$, where $I(C)$ is the sum of edges where both vertices are in C, and $E(C)$ is the sum of edges where only other vertex is in C [10]. When using the plain hit matrix values to generate a graph, the inverse of the collusion index is used. Also, we need a method to decide whether a colluding subset exists at all. Selecting the correct subset or testing if collusion exists are, however, outside the scope of this paper.

III. Results

We use Pakuhaku game (Fig. 1) that we have presented in our earlier papers [15], [17], and the R programming language [22] to run our experiment. In Pakuhaku, eight players try to collect as many pills as possible from the playing field.The game ends when the total of collected pills reaches 64, and the winner is the player with most pills. The players have a limited field of view and can shoot each other with a ray that freezes the target for a short time. The game will output the hit matrix of each run and attributes of the players in JSON format [23]. The following attributes are used to create the synthetic players:

- Aggression $[0,1]$: When deciding whom to shoot, the colluders have an aggression probability to shoot another colluder.
- Skill $[0,1] \times \mathbb{N}$: When creating the players, the skill of a player is randomly selected from a normal distribution with given mean and standard deviation. When deciding an action, the player has a skill probability to go after a pill or shoot an available target.
- Strategy $\mathcal{P}_{\geq 1}(\{random, scanning\})$: When creating the synthetic players, the play style of a player is randomly selected from the given set. The players have two strategies: random or scanning. Random strategy picks a random target point, and the player will move there and then pick a new target. Scanning starts at one of the corners and systematically moves to the opposite side of the game world. If the player sees a pill, it will move towards it regardless of the selected strategy.
- Teams $\mathbb{N}$: The players will be distributed to teams and team members will not shoot each other.

In this experiment, we do not use the other collusion methods studied in our previous work, but concentrate solely on the soft play collusion. We generate 50 test runs with each of the following settings presented in Table III.

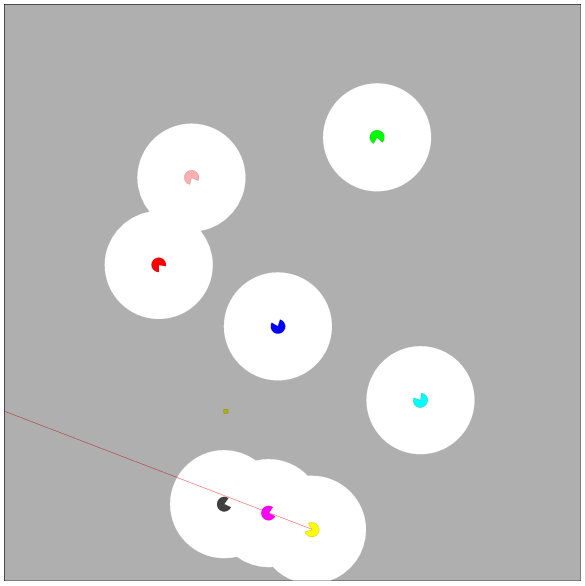

Fig. 1. In the Pakuhaku game, eight players try to collect as many pills as possible from the playing field. Pills are dispensed to the game world one pill at a time. A new pill is added to a random location every time a pill is collected. The game ends when the players have collected 64 pills in total, and the winner is the player with most pills. The game world has a size of 800×800 units and all players start at the centre of the world. The players are modelled as circles with a radius of 10, units and they can move 5 units per turn to any direction. The pills are modelled as circles with a radius of 3 units. The field of view has a radius of 75 units. The players can shoot each other once every 20 turns with a ray that freezes the target for 10 turns.

TABLE III. Test Scenarios

	Colluders	Skill	Strategy	Aggression	Teams
2 Plain	2	1, 0	random	1	0
2 Full	2	0.6, 0.15	random, scanning	1	0
2 Aggro	2	0.6, 0.15	random, scanning	0.5	0
3 Plain	3	1, 0	random	1	0
3 Full	3	0.6, 0.15	random, scanning	1	0
3 Aggro	3	0.6, 0.15	random, scanning	0.5	0
2 Teams	0	0.6, 0.15	random, scanning	—	2

Except when using three colluders with reduced aggression, the collusion has a statistically significant effect on at least two of the values. The colluders have fewer kills and deaths than non-colluders but with the ratio and difference are better than the non-colluders. When aggression was not reduced, the colluders were able to win more rounds than non-colluders. The effects are presented in Table IV where each cell (except

the u_w column) contains mean values of the corresponding metric. When a cell contains two values, the top value is for the non-colluders and the bottom value is for the colluders. If the difference between means is significant based on one-way analysis [24], the values are presented in bold. The u_w column contains the utility of collusion as the difference of average wins per colluder and average wins per non-colluder [16]:

$$u_w = \sum_{q \in Q} wins(q)/\left|Q\right| - \sum_{p \in P} wins(p)/\left|P\right| \qquad (7)$$

where Q is the set of colluding players and P is the set of non-colluding players. The player with the most pills at the end of the game is considered to be the winner.

TABLE IV. THE EFFECTS OF COLLUSION

	Kills	Deaths	KD-spread	KD-ratio	Pills	u_w
2 Plain	**91.95** **86.43**	**93.73** **81.09**	**-1.78** **5.34**	**0.9856** **1.0748**	7.933 8.200	6.333
2 Full	**74.07** **68.30**	**75.04** **66.40**	**-0.9667** **2.9000**	**0.9953** **1.0501**	7.89 8.33	1.667
2 Aggro	100.21 99.95	100.8 98.1	**-0.6167** **1.8500**	**0.9991** **1.0247**	8.03 7.91	-1
3 Plain	**96.68** **82.10**	**102.1** **73.1**	**-5.4** **9.0**	**0.9524** **1.1334**	**7.564** **8.733**	4.933
3 Full	**70.98** **59.69**	**74.67** **53.53**	**-3.692** **6.153**	**0.9568** **1.1256**	**7.736** **8.440**	2.267
3 Aggro	97.47 96.73	98.06 95.64	-0.5920 0.9867	0.9986 1.0186	8.032 7.947	-0.4

A. Principal Component Analysis

For reference, we first made the experiment with the 2 Teams scenario to get a baseline expectation for the p-values of the model and variables. The relationship between a team and the correlation with the first principal component is significant ($p \leq 2 \cdot 10^{-4}$) but different from Van der Knyff et al. results. In their experiments, different teams correlated with different principal components whereas in our experiment both teams correlated with the first principal component but with a different sign.

To see how adding more variation to the players affects the significance on the collusion on correlation we use the settings from Table II. The collusion and the correlation with first principal have a significant relationship in all scenarios except when aggression is 0.5. However, the significance and the quality of the models decreases when the players have more variation. Also, the principal component having the most significant relationship starts to be more evenly distributed. Like the Van der Knyff et al. experiment it is impossible to know beforehand which values of the correlation indicate colluders and non-colluders. In our experiment colluders and non-colluders are distinguished by the sign of the correlation instead of which principal component has the largest correlation. Also, when combining collusion with teams, the first principal components contained information about both. In conclusion, the method proposed Van der Knyff et al. is theoretically sound and works in simple scenarios. It is not practically usable due to real player setup being too complex for the method to work.

B. Graph Clustering

We apply the shared_NN, mutual_NN, collusion clustering, and community detection algorithms to the graphs generated from the hit matrix using hits and collusion scores as edge weights. The algorithms have different parameters, and we use the values recommended by the authors except for the number of nearest neighbours used k, for which we use values from one to three (Table V).

TABLE V. PARAMETERS USED FOR THE ALGORITHMS

Algorithm	Parameters
shared_NN	$k \in [1,3]\,, kt = 2$
mutual_NN	$k \in [1,3]$
collusion clustering	$k \in [1,3]\,, m = 2, h = 0.7$
community detection	$k \in [1,3]$

All of the algorithms, except the shared_NN, were able to separate the colluding subset in most cases in the two colluder settings without aggression. Mutual_NN and community detection also worked well with three colluders and no aggression, but collusion clustering failed in these settings. There is a small reduction in the accuracy when the players have more variation. When collusion is made less prominent by making the colluders shoot each other, the algorithms are unable to detect the colluding subset. In these cases, the colluders were able to win fewer games than non-colluders but were able to boost kill-related statistics (see Table IV). Fig. 2 shows accuracy of detection. The loss of accuracy is consistent with the decreasing p-values from principal component analysis.

When using the mutual_NN algorithm, the colluding subset was often the largest detected subset, and when using community detection, the colluding subset was the smallest detected community. In some cases, the colluding subset is found by the algorithm, but it is not the smallest or the largest subset. Selecting the cluster with the largest collusion index provides better results than selecting the smallest or the largest cluster. The results depend on the value of k and collusion clustering performs better when k large while mutual_NN and community detection work better when k one less than the number of colluders. Because larger values of k make the algorithms slower it is important to choose the value carefully based on the optimal size of the colluding subset.

C. Computational performance

The time required to process a single game is on average less than one second, which is negligible compared to the length of a typical game. On a personal computer with AMD FX-8320 processor at 3.51 GHz and 16 GB of RAM clustering a batch of 50 games took between 0.5 seconds and 19.8 seconds. The type of algorithm and the k-parameter had most impact on the execution time (see Fig. 3). Because the performance is lower for larger values of k and k should be at least the expected number of colluders minus one, the performance can become a problem when the number of players is large.

The complexities of the algorithms are: shared_NN $O(kN^2)$, mutual_NN $O(kN^2)$ and collusion clustering $O(kN^3)$ [10]. The optimal.community algorithm is based on modularity optimization, which is an NP-complete problem,

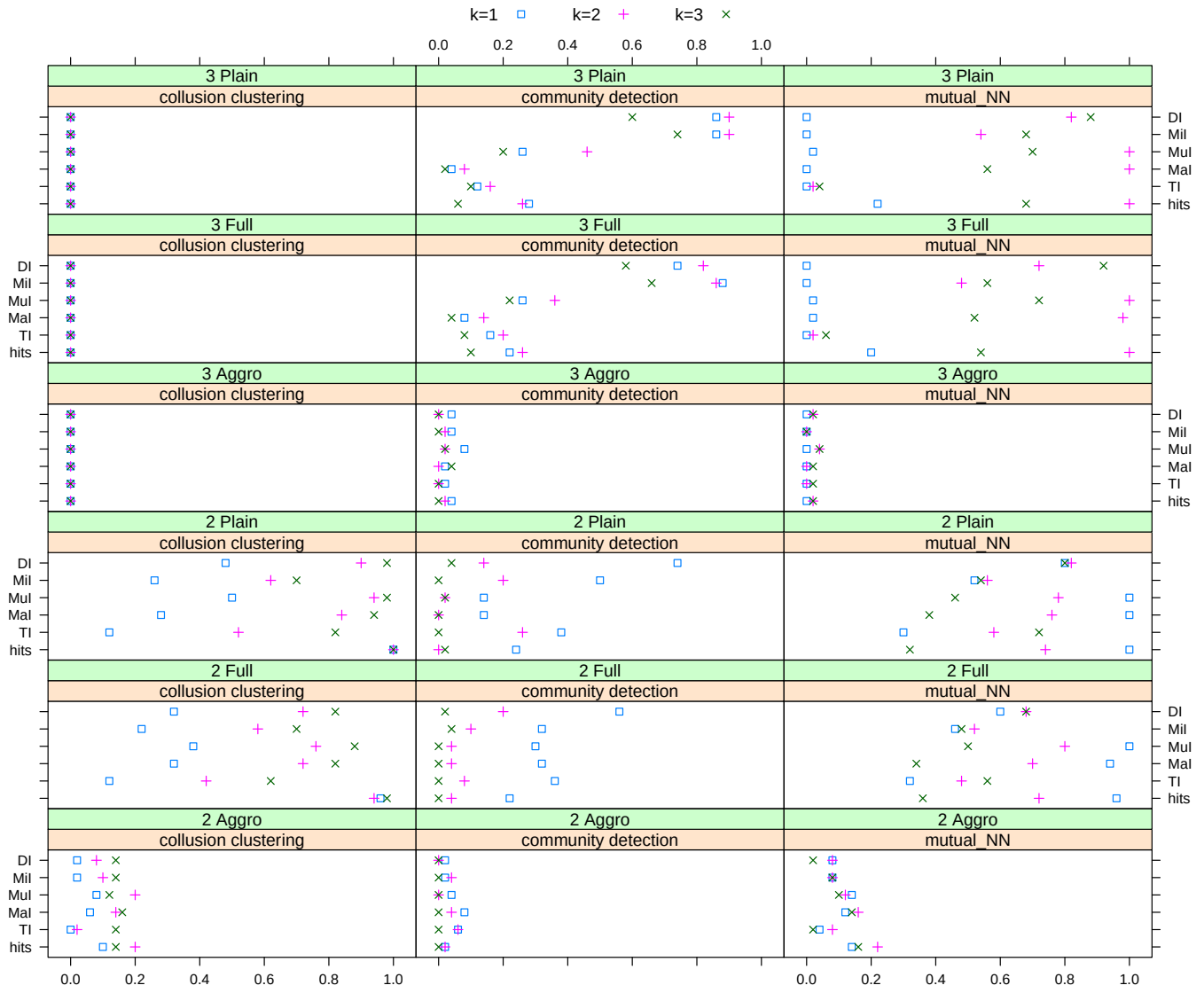

Fig. 2. The amount of colluding subsets found in 50 test games using different edge weights (vertical titles) and clustering algorithms (orange titles). Shared_NN has been omitted due to its poor performance. Games without colluders are not included.

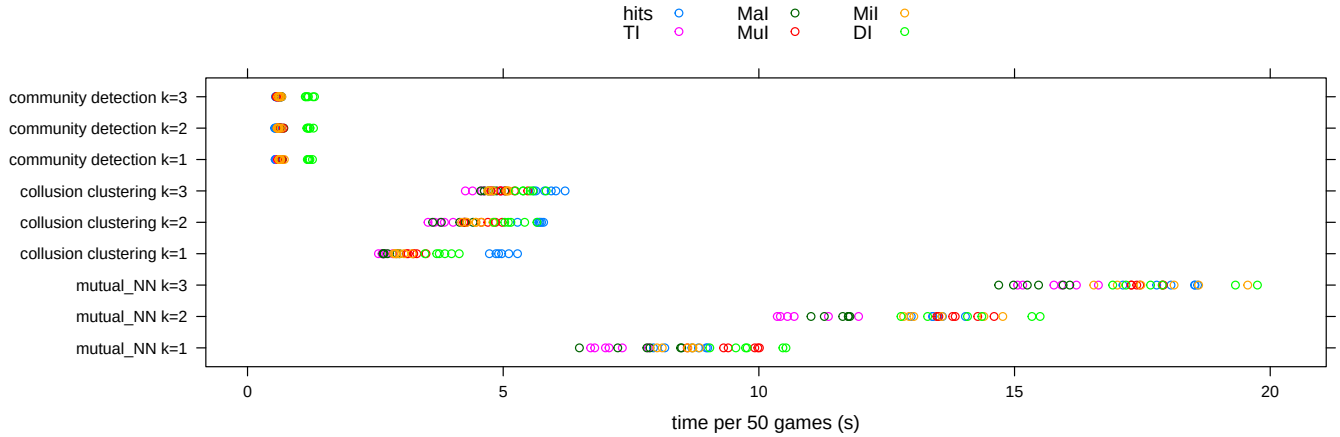

Fig. 3. Time required to get the clustering result from a batch of 50 games. For community detection the k parameter is only used for the pruning step and as minimal effect on execution time.

and has exponential complexity [20]. The faster execution time of the algorithm is most likely explained by the use of a native library for calculations.

IV. Conclusions and Future Work

We were successful in finding the colluding subset in simple scenarios, but finding the colluding subset gets more difficult when collusion is not the only source of variation among the players. In addition, we tested if there is a significant relationship between collusion and correlation of the rows and the principal components of a hit matrix. We found significant correlation only when collusion was the only source of variation among the players. Both of the results demonstrate the overall difficulty of collusion detection; even when our game and synthetic players are simplistic, the problem is not easy to solve.

From the studied algorithms, mutual_NN and community detection worked best in our test scenarios. Shared_NN worked poorly on all tests and collusion clustering failed on three colluder tests. We confirmed our expectations that the correlation of principal components of hit matrix rows is significant only in very simple scenarios where collusion is the strongest source of variation. Therefore, the method has little practical use in this area.

In this paper, we examined only death match style scenarios where all players play against each other. However, team-based modes are more popular than pure all-versus-all modes. Finding a colluding subset is not limited to shooter games and single rounds of a game. For example, a similar method could be used on team level to detect match fixing or the methods could be adjusted to find other unwanted behaviour like griefing [25] or camping. To evaluate the methods in these situations requires further research.

We did not take the player's skill into account, which may make it difficult to discern a bad player from a colluder. The method could be improved by integrating a skill measure into the graph. The real value of a player's skill is unknown in reality, but metrics like TrueSkill™ [26] can be used in place to provide an estimate of the player's skill level.

We studied only a small subset of algorithms, and there are several possibilities for improving soft play detection. Islam et al. [27] use a Markov clustering algorithm [28], [29] to improve results of Palshikar and Apte [10] and it could also work for soft play detection. The players engaging soft play have fewer mutual hits, which leads to a set of vertices with few internal connections.This kind of structure is called an anti-community, and algorithms have been developed to detect this type of structure [30], [31], [32].

Admittedly, the Pakuhaku game is too simple for realistic experiments, but it is useful for testing and trying out methods in a more controlled environment before moving to more realistic scenarios. To overcome its limitations and allow human participants to take part in the experiments, we are currently working on a new test framework.

References

[1] C. Vallve-Guionnet, "Finding colluders in card games," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II - Volume 02*, ITCC '05, (Washington, DC, USA), pp. 774–775, IEEE Computer Society, 2005.

[2] R. Yampolskiy, "Detecting and controlling cheating in online poker," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pp. 848–853, Jan 2008.

[3] J. Yan, "Collusion detection in online bridge," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pp. 1510–1515, 2010.

[4] P. Mazrooei, C. Archibald, and M. Bowling, "Automating collusion detection in sequential games," in *AAAI Conference on Artificial Intelligence*, 2013.

[5] C. M. VanderKnyff, D. J. Bethea, M. M. K. Reiter, and M. C. Whitton, "Statistical methods for user and team identification in multiplayer games," tech. rep., University of North Carolina, 2009.

[6] I. T. Joliffe, *Principal Component Analysis*. Springer-Verlag New York, 2002.

[7] S. J. Murdoch and P. Zieliński, "Covert channels for collusion in online computer games," in *Information Hiding* (J. Fridrich, ed.), vol. 3200 of *Lecture Notes in Computer Science*, pp. 355–369, Springer Berlin Heidelberg, 2005.

[8] A. Ercole, K. D. Whittlestone, D. G. Melvin, and J. Rashbass, "Collusion detection in multiple choice examinations," *Medical Education*, vol. 36, no. 2, pp. 166–172, 2002.

[9] S. Zander, G. Armitage, and P. Branch, "Covert channels in multiplayer first person shooter online games," in *33rd IEEE Conference on Local Computer Networks, 2008. LCN 2008.*, pp. 215–222, Oct 2008.

[10] G. K. Palshikar and M. M. Apte, "Collusion set detection using graph clustering," *Data Mining and Knowledge Discovery*, vol. 16, no. 2, pp. 135–164, 2008.

[11] E. Staab and T. Engel, "Collusion detection for grid computing," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09.*, pp. 412–419, May 2009.

[12] J.-K. Lou, K.-T. Chen, and C.-L. Lei, "A collusion-resistant automation scheme for social moderation systems," in *6th IEEE Consumer Communications and Networking Conference, 2009. CCNC 2009.*, pp. 1–5, Jan 2009.

[13] X. Zhou and H. Zheng, "Breaking bidder collusion in large-scale spectrum auctions," in *Proceedings of the eleventh ACM international symposium on Mobile ad hoc networking and computing - MobiHoc '10*, p. 121, ACM Press, 2010.

[14] J. Smed, T. Knuutila, and H. Hakonen, "Can we prevent collusion in multiplayer online games?," in *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)* (Honkela, Raiko, Kortela, and Valpola, eds.), pp. 168–175, 2006.

[15] J. Smed, T. Knuutila, and H. Hakonen, "Towards swift and accurate collusion detection," in *8th International Conference on Intelligent Games and Simulation (Game-On 2007)*, pp. 103–107, 2007.

[16] J. Laasonen, T. Knuutila, and J. Smed, "Eliciting collusion features," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, (ICST, Brussels, Belgium, Belgium), pp. 296–303, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[17] J. Laasonen and J. Smed, "Detecting a colluding subset in a simple two-dimensional game," Tech. Rep. 1074, Turku Centre for Computer Science, 2013.

[18] T. Hesterberg, J. M. Chambers, and T. J. Hastie, *Statistical Models in S*, vol. 35. 1993.

[19] "igraph R package." Software. http://igraph.org/r/.

[20] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 2, 2008.

[21] P. Mazrooei, "Collusion detection in sequential games," Master's thesis, University of Alberta, Edmonton, Alberta, Sept. 2012.

[22] "The R Project for Statistical Computing." Software. http://www.r-project.org.

[23] "ECMA-404 The JSON Data Interchange Format." http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, October 2013.

[24] B. Welch, “On the comparison of several mean values: an alternative approach,” *Biometrika*, vol. 38, no. 3/4, pp. 330–336, 1951.

[25] H. Lin and C.-T. Sun, “"white-eyed" and "griefer" player culture: Deviance construction in mmorpgs,” in *Worlds in Play: International Perspectives on Digital Games Research* (S. d. Castell and J. Jenson, eds.), ch. 8., pp. 103–114, Peter Lang Publishing Inc, 2007.

[26] “Trueskill™ ranking system.” Webpage. http://research.microsoft.com/en-us/projects/trueskill/.

[27] M. N. Islam, S. M. R. Haque, K. M. Alam, and M. Tarikuzzaman, “An approach to improve collusion set detection using mcl algorithm,” in *2009 12th International Conference on Computers and Information Technology*, pp. 237–242, IEEE, 12 2009.

[28] S. V. Dongen, “Graph clustering by flow simulation,” *Structure*, vol. 1, p. 27–64, 2000.

[29] S. v. Dongen, “A cluster algorithm for graphs,” *Information Systems [INS]*, no. R 0010, pp. 1–40, 2000.

[30] B. L. Chen, L. Chen, S. R. Zou, and X. L. Xu, “Quantitative measurement and method for detecting anti-community structures in complex networks,” *International Journal of Wireless and Mobile Computing*, vol. 6, p. 431, 10 2013.

[31] L. Chen, Q. Yu, and B. Chen, “Anti-modularity and anti-community detecting in complex networks,” *Information Sciences*, vol. 275, pp. 293–313, 8 2014.

[32] Q. Yu and L. Chen, “A new method for detecting anti-community structures in complex networks,” *Journal of Physics: Conference Series*, vol. 410, p. 012103, 2 2013.