Jussi Laasonen | Jouni Smed

Detecting a colluding subset in a simple two-dimensional game

Turku Centre *for* Computer Science

TUCS Technical Report
No 1074, April 2013

Detecting a colluding subset in a simple two-dimensional game

Jussi Laasonen
University of Turku, Department of Information Technology

Jouni Smed
University of Turku, Department of Information Technology

TUCS Technical Report
No 1074, April 2013

Abstract

Collusion is covert co-operation between the participants of a game. Detecting the colluding players can require discerning and understanding the player's motivation, which is often difficult task even for humans. In this paper we analyse experimental data from a simple two-dimensional game using synthetic players. We calculate information gains of the features in the data to show how well they indicate collusion. Then we examine J4.8 decision tree classifiers learned from the data and use them to detect the colluding subsets.

Keywords: collusion, cheating detection, computer games, classification, decision tree

TUCS Laboratory
Software Development Laboratory

1. Introduction

Collusion means an attempt of covert co-operation. The players who are colluding are called colluders and the set of players who are colluding together is a colluding subset of all players. Detecting the colluding subset can require understanding the reasons behind player's motivation, which can be a difficult task even for humans.

Collusion can take many forms and appear in different contexts like tournaments [1], multiple choice examinations [2], covert communication channels [3], stock market trading [4], [5] social moderation [6], grid computing [7], spectrum auctions [8] and games. In games collusion has been addressed in poker [9–11] and bridge [12], but apart from them there is very little research.

In our earlier work [13] we proposed a set of features to be used in collusion detection in Pakuhaku game and compared briefly the information gains of the features with four colluders in dispenser and even distribution settings. This paper continues our research approach [14]:

1. Generate game data with different number of players, colluders, game types, and collusion strategies.
2. Devise detection methods.
3. Run the detection method against the data to get results.
4. Compare accuracy: How many (if any) of the colluders got detected.
5. Compare swiftness: How quickly the colluders were detected.

Our task is now to analyse how well the proposed features indicate collusion with different number of colluders. We create J4.8 decision tree classifiers from the datasets and analyse their performance in detecting the colluding subsets.

We present the Pakuhaku game in section and the generated data sets in section 2. Section 3 is divided to three parts. First, we analyse the information gains of the collusion features and then we present the results of collusion detection experiments using a decision tree classifier. Analysis of the swiftness of our collusion detection method concludes the section. Finally, in section 4 we present a summary of the results and our plans for future research.

2. Methods

As with our previous experiments, we use the Pakuhaku game [14] to generate the test data. In Pakuhaku, the players try to collect as many pills as possible before the game ends. The game world has the size of 800 times 800 units and players and pills are modelled using circles with the radii of 10 and 3 units, respectively. All players start at

the centre of the game world and can move freely to any direction at the speed of 5 units per turn. Players move towards the nearest or randomly selected location, if no pills are visible. Players have a fog-of-war with the radius of 75 units. Players keep a tabu list [15] of visited locations so they do not try to move to an area that is known to contain no pills. Players have a beam weapon, which can fire a beam of unlimited range every 20 turns. Players hit by a beam freeze for 10 turns. The beam passes through players and can hit multiple players at once.

The players and colluders have four different implementations of this basic behaviour to guide their movements:

- *Default player*: The player keeps its own tabu list (see Algorithm 1).
- *Tabu colluder*: The player plays like the default player, but informs all new additions to its tabu list to its co-colluders (i.e., the colluders have a shared tabu list). (see Algorithm 2)
- *Area-of-interest (AOI) colluder*: The colluders divide the game world into non-overlapping evenly sized rectangles or - if that is not possible - evenly sized horizontal slices (Figure 1). The player acts like a tabu colluder but does not leave its designated area-of-interest. (see Algorithm 3)
- *Blocking colluder*: One of the colluders is elected as a leader, who will play like a tabu colluder. The other colluders also play like the tabu colluder but try to keep near the non-colluding players so that they can prevent them from reaching pills by eating them first. (see Algorithm 4)

The players shoot at a random visible target immediately when their weapon is loaded. Colluding players engage in soft play and do not shoot at their co-colluders, although they might get hit by collateral fire.

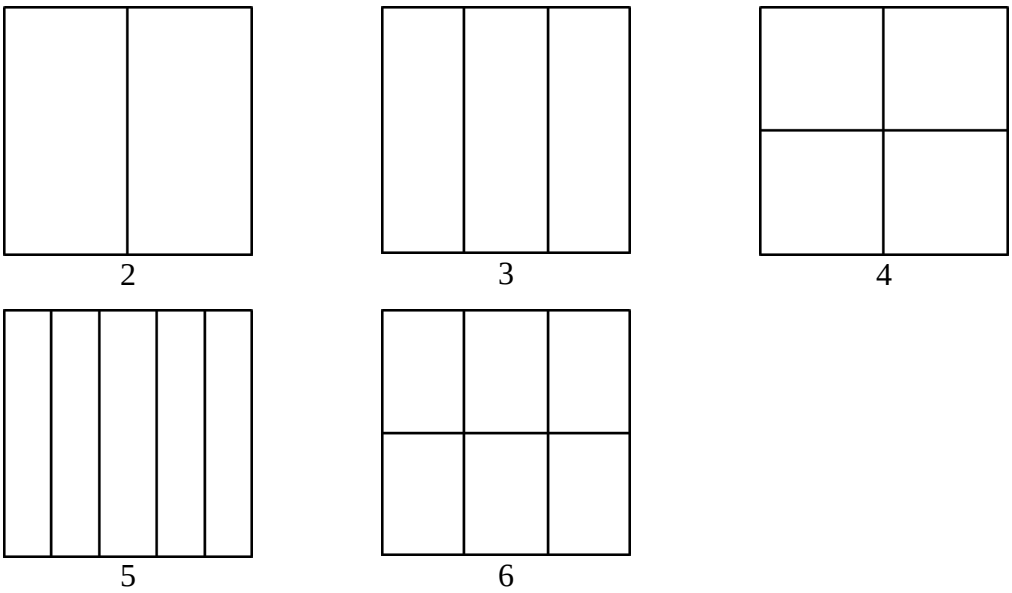

Figure 1 AOI partitions for 2 to 6 colluders

Algorithm 1: Decision making for the *default player*

UPDATE(*p*)
  in: the current player *p*

  if NEWPILLISDISPENSED() *tabu(p)* ← ∅
  if WANTSTOSHOOT(*p*)
    φ ← NORMALISE(GETTARGETPOSITION(*p*) – *position(p)*)
    SHOOT(*p*)
  else
    φ ← DECIDEDIRECTION(*p*)
    position(*p*) ← position(*p*) + *speed(p)*·φ

DECIDEDIRECTION(*p*)
  in: the current player *p*
  out: a new direction vector for the player

  if *destination(p)* = null
    destination(p) ← PICKNEWDESTINATION(*p*)
  else
    if ISDISPENSED()) *tabu(p)* ← ∅
    tabu(p) ← *tabu(p)* ∪ VISIBLEPLAYERPOSITIONS()
    nearestPill ← FINDNEARESTPILL()
    if *nearestPill* ≠ null
      destination(p) ← *nearestPill*
    else if $|position - destination| < 2.5 \cdot s$
      destination(p) ← PICKNEWDESTINATION(*p*)
  return NORMALISE(*destination* - *position*)

PICKNEWDESTINATION(*p*)
  in: the current player *p*
  out: a new destination location
  constant: the maximum number of tries before clearing the tabu list $trials_{max}$

  trials ← 0
  do
    trials ← *trials* + 1
    if *trials* > MAX($trials_{max}$, $|tabu(p)|$) *tabu(p)* ← ∅
    destination ← RANDOMPOSITION()
  while POSITIONISTABU(*destination, p*)
  return *destination*

POSITIONISTABU(*position, p*)
  in: a candidate position *position*; the current player *p*
  out: true if *p* is tabu position, false otherwise
  constant: fog of war radius r_{fow}

  return $\exists x \in tabu \left| |x - position| < r_{fow} \right.$

GETTARGETPOSITION(*p*)
in: the current player *p*
out: target player for *p*

FINDNEARESTPLAYER(*p*)

WANTSTOSHOOT(*p*)
in: the current player *p*
out: true if *p* is able and wants to shoot, false otherwise

WEAPONISREADY(*p*) **and** HASVISIBLEPLAYERS(*p*)

Algorithm 2: Decision making for the *tabu colluder* as different from Algorithm 1.

UPDATE(*p*)
in: the current player *p*

SENDMESSAGE(*position(p)*)
RECEIVEMESSAGES(*p*)
∇ Pick destination like the *default player*.
…

RECEIVEMESSAGES(*p*)
in: the current player *p*

for *message* **in** *messages(p)*
tabu(p) ← *tabu(p)* ∪ *message*
messages(p) ← ∅

SENDMESSAGE(*message*, *p*)
in: the message to send to co-colluders *message*, the current player *p*

for *colluder* **in** *colluders(p)*
messages(colluder) ← *messages(colluder)* ∪ *message*

GETTARGETPOSITION(*p*)
in: the current player *p*
out: target player for *p*

FINDNEARESTNONCOLLUDER(*p*)

WANTSTOSHOOT(*p*)
in: the current player *p*
out: true if *p* is able and wants to shoot, false otherwise

return WEAPONISREADY(*p*) **and** HASVISIBLENONCOLLUDERS(*p*)

Algorithm 3: Decision making for the *AOI colluder* as different from Algorithm 2.

POSITIONISTABU(*position*, *p*)

in: a candidate position *position*; the current player *p*

out: true if *position* is a tabu position or outside of the area-of-interest, false otherwise

constant: fog of war radius r_{fow}

return not($p \in$ *area-of-interest(p)*) **or** $\exists x \in tabu \left| |x - position| < r_{fow} \right.$

Algorithm 4: Decision making for the *blocking colluder* as different from Algorithm 2.

PICKNEWDESTINATION(*p*)

in: the current player *p*

out: a new destination location

if ISLEADER(*p*) **or** PILLSVISIBLE(*p*) **or** NOVISIBLENONCOLLUDERS(*p*)

∇ Pick destination like the *tabu colluder*.

…

else

return FUTUREPOSITION(FINDNEARESTNONCOLLUDER(*p*), *p*)

FUTUREPOSITION(*target*, *p*)

in: a target player *target*; the current player *p*

out: the predicted future position of the *target*

timeToTarget ← |*position(target) – position(p)*|/*speed(p)*

return *position(target) + speed(target)·direction(target)·timeToTarget*

In this paper, we concentrate on the dispenser competition: There is only one pill, which is repositioned to the game world into a random position after it gets eaten. The game ends immediately when the sum of pills eaten by all the players reaches 64.

Let A be the set of all players, $P \subseteq A$ be the set of non-colluding players and $Q \subseteq A$ the set of colluding players such that $P \cap Q = \emptyset$ and $P \cap Q = A$. Let $p \in P$ be a non-colluding player and $q \in Q$ be a colluding player. Moreover, let

- r_{max}: the number of rounds in a single game
- w, h: the width and height of the game world
- $wins(p)$: the number of games the player p has won
- $pills(p)$: the number of pills the player p has collected
- $hits(p)$: the number of hits the player p has given
- $hits(p, q)$: the number of hits the player p has given to the player q
- $deaths(p)$: the number of the hits the player p has taken

- $\ell_{p,r} = (x_{p,r}, y_{p,r})$: the location of the player p at the round $r (1 \leq r \leq r_{max})$
- $\overline{v}_{p,r}$: the velocity vector (from previous position to the current position) of the player p at the round $r (1 \leq r \leq r_{max})$
- $box(p,r)$: the bounding box of the player p (i.e., a square centered on the $\ell_{p,r}$ with the width and height of two times the radius of fog-of-war)
- $box(L)$: the bounding box of the set of locations L
- $area(b)$: the area of the bounding box b
- $S \subseteq A$: a subset of players
- $angle(r,S) = median_{\varphi_i \in \Phi, \varphi_i \leq \varphi_{i+1}}(\varphi_2 - \varphi_1, \cdots, \varphi_{|S|} - \varphi_{|S|-1}, 2\pi - \varphi_{|S|} + \varphi_1)$, where $\Phi = \left\{\varphi_s = \cos^{-1}\left(\frac{\bar{v}_{s,r} \cdot \bar{v}_{s,r}}{|\bar{v}_{s,r}|^2}\right) | s \in S\right\}$: median of the angles between the players in S
- $direction(r,S)\left|\sum_{s \in S} \bar{v}_{s,r}\right|$:length of the sum of the velocities of the players in S
- $correlation(r,S) = \left|\rho_{\{\ell_{s,r} | s \in S\}}\right|$: the absolute value of the Pearson correlation of the players' locations
- $distance(r,S) = max_{s,t \in S}(|\bar{v}_{s,r} - \bar{v}_{t,r}|)$: the maximum distance between a pair of players in S
- $intersection(r,S) = area(\bigcap_{s \in S} box(s))/wh$: the intersection area of the players' bounding boxes divided by the game world's area
- $union(r,S) = area\left(box(\{\ell_{s,r} | s \in S\})\right)/wh$: the union area of the players' bounding boxes divided by the game world's area

We created a test data for each colluder type using collusion features proposed by Laasonen et al. [13]. To make comparison of a large number of features easier, we arrange the features to four groups: direction, shooting, location and area, which base on the measure they represent. The features and their grouping are listed in Table 2. In addition the following meta-features were added to the data:

- *colluderRatio* [0,1]: fraction of colluders in the subset
- *colluders* $\mathbb{N}$: number of colluders for the game
- *collusion* {0,1}: 1 if $S = Q$, 0 otherwise

We performed 100 runs with each number of colluders between 2 and 6. For each test run one row was created for each subset $S \subseteq A, |S| = |Q|$. We have assumed that the number of the colluders is somehow known beforehand. While this is an unrealistic assumption, it makes the number subsets substantially smaller. The full data for each colluder type contains 23800 rows. The number of rows with specific number of colluders is listed in the Table 1. The data in ARFF format [16] and supplementary files are available for download from figshare [17].

Table 1 Number of instances with different number of colluders

# of colluders	2	3	4	5	6	Full dataset
# instances	2800	5600	7000	5600	2800	23800

We calculate the information gain to analyse how the number of colluders affects the different feature. Because the value class information is maximum for information gain and it is different in the different subsets based on the number of colluders we use $gain(feature)/info(collusion)$ to compare the features.

We used the *R* programming language [18] and the *Weka 3* software suite [19] to create J4.8 decision tree classifiers from the datasets and perform tenfold cross validation. J4.8 is Weka's implementation of C4.5 decision tree learner [20] and it was chosen because it is available in Weka suite out of the box and decision trees are easy for humans to comprehend.

Table 2 Collusion features

Group	Feature	Function
Direction	angleMean	$mean_{1\le r\le r_{max}}(angle(r,S))$
	angleMax	$max_{1\le r\le r_{max}}(angle(r,S))$
	angleMin	$min_{1\le r\le r_{max}}(angle(r,S))$
	angleStdDev	$stddev_{1\le r\le r_{max}}(angle(r,S))$
	directionsMean	$mean_{1\le r\le r_{max}}(direction(r,S))$
	directionsMax	$max_{1\le r\le r_{max}}(direction(r,S))$
	directionsMin	$min_{1\le r\le r_{max}}(direction(r,S))$
	directionsStdDev	$stddev_{1\le r\le r_{max}}(direction(r,S))$
Shooting	deathsTotal	$\sum_{s\in S} deaths(s)/\lvert S\rvert$
	deathsTotalDelta	$stddev_{s\in S}(deaths(s))$
	hitsAll	$\sum_{s\in S} hits(s)/\lvert S\rvert$
	hitsAllDelta	$stddev_{s\in S}(hits(s))$
	hitsMutual	$\sum_{p\in S}\sum_{q\in S,p\neq q} hits(p,q)$
	hitsMutualDelta	$stddev(\{hits(p,q)\lvert p,q\in S,p\neq q\})$
Location	correlationMean	$mean_{1\le r\le r_{max}}(correlation(r,S))$
	correlationMax	$max_{1\le r\le r_{max}}(correlation(r,S))$
	correlationMin	$min_{1\le r\le r_{max}}(correlation(r,S))$
	correlationStdDev	$stddev_{1\le r\le r_{max}}(correlation(r,S))$
	meanXMean	$mean_{1\le r\le r_{max}}\left(mean_{s\in S}(x_{s,r})\right)$
	meanXMax	$max_{1\le r\le r_{max}}\left(mean_{s\in S}(x_{s,r})\right)$
	meanXMin	$min_{1\le r\le r_{max}}\left(mean_{s\in S}(x_{s,r})\right)$
	meanXStdDev	$stddev_{1\le r\le r_{max}}\left(mean_{s\in S}(x_{s,r})\right)$
	meanYMean	$mean_{1\le r\le r_{max}}\left(mean_{s\in S}(y_{s,r})\right)$
	meanYMax	$max_{1\le r\le r_{max}}\left(mean_{s\in S}(y_{s,r})\right)$
	meanYMin	$min_{1\le r\le r_{max}}\left(mean_{s\in S}(y_{s,r})\right)$
	meanYStdDev	$stddev_{1\le r\le r_{max}}\left(mean_{s\in S}(y_{s,r})\right)$

(Continues on the next page.)

Table 2 (continues)

Group	Feature	Function
Area	distanceMean	$mean_{1\le r\le r_{max}}\big(distance(r,S)\big)$
	distanceMax	$max_{1\le r\le r_{max}}\big(distance(r,S)\big)$
	distanceMin	$min_{1\le r\le r_{max}}\big(distance(r,S)\big)$
	distanceStdDev	$stddev_{1\le r\le r_{max}}\big(distance(r,S)\big)$
	intersectionMean	$mean_{1\le r\le r_{max}}\big(intersection(r,S)\big)$
	intersectionMax	$max_{1\le r\le r_{max}}\big(intersection(r,S)\big)$
	intersectionMin	$min_{1\le r\le r_{max}}\big(intersection(r,S)\big)$
	intersectionStdDev	$stddev_{1\le r\le r_{max}}\big(intersection(r,S)\big)$
	totalIntersetion	$area\left(\bigcup_{s\in S} box\big(\{\ell_{s,r}\vert 1\le r\le r_{max}\}\big)\right)/wh$
	totalUnion	$area\left(\bigcap_{s\in S} box\big(\{\ell_{s,r}\vert 1\le r\le r_{max}\}\big)\right)/wh$
	unionMean	$mean_{1\le r\le r_{max}}\big(union(r,S)\big)$
	unionMax	$max_{1\le r\le r_{max}}\big(union(r,S)\big)$
	unionMin	$min_{1\le r\le r_{max}}\big(union(r,S)\big)$
	unionStdDev	$stddev_{1\le r\le r_{max}}\big(union(r,S)\big)$
Score	scoreTotal	$\sum_{s\in S} pills(s)/\vert S\vert$
	scoreTotalDelta	$stddev_{s\in S}\big(pills(s)\big)$

3. Results

3.1. Analysis of the collusion features

A common pattern for gain as a function of the number of the colluders is that gain starts to grow when the number of colluders increases and reaches the maximum around 4 or 5 colluders and then starts to decrease again. With AOI colluders there is more fluctuation on the information gain given the number colluders. This is due to a different partitioning of the game world.

3.1.1. Score

Score-based features have a low information gain (see Figure 2). Even if they would have a high gain, it would be problematic to use them to recognize colluders because it could also discriminate good players.

In previous work [13] we have shown that the colluders are able to collect more pills than the non-colluders. However, the difference is not large enough to separate the classes (see Figure 3). Distribution of values reflects the utility function (see Figure 4) [13]:

$$u_p = \sum_{q \in Q} pills(q) \Big/ |Q| - \sum_{p \in P} pills(p) / |P|$$

The non-colluding subsets have larger *scoreTotal* when number of colluders and the subset size is large. This could indicate that the few non-colluders are able to perform better than they would among more non-colluders, but it is more likely duo mixed subsets containing mostly colluders. For further analysis pure non-colluders subsets need to be separated from mixed subsets.

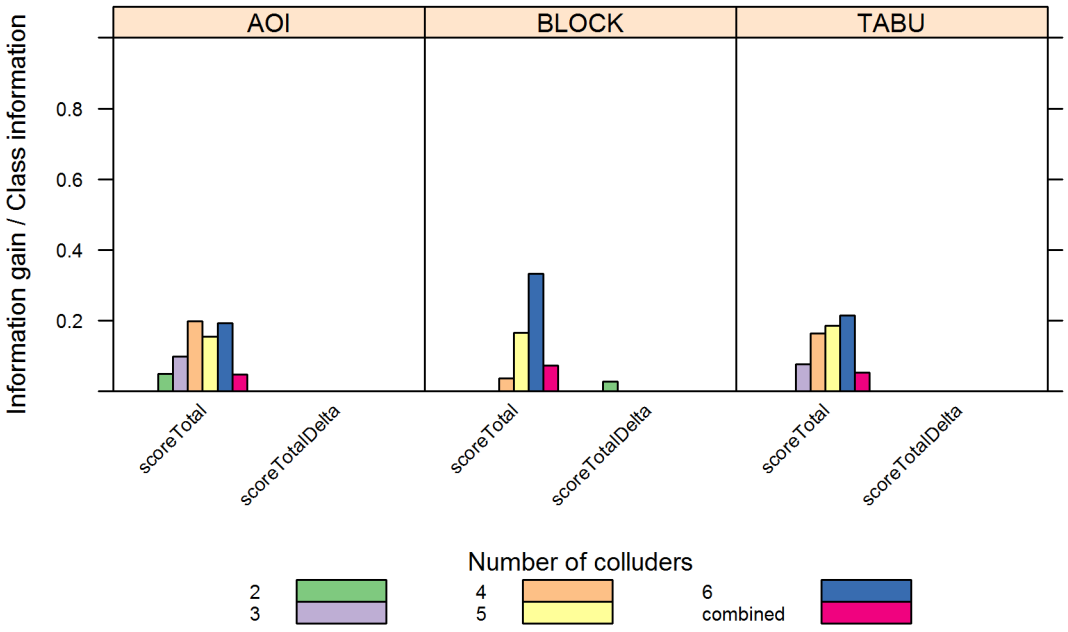

Figure 2 Information gain / class information of the score features

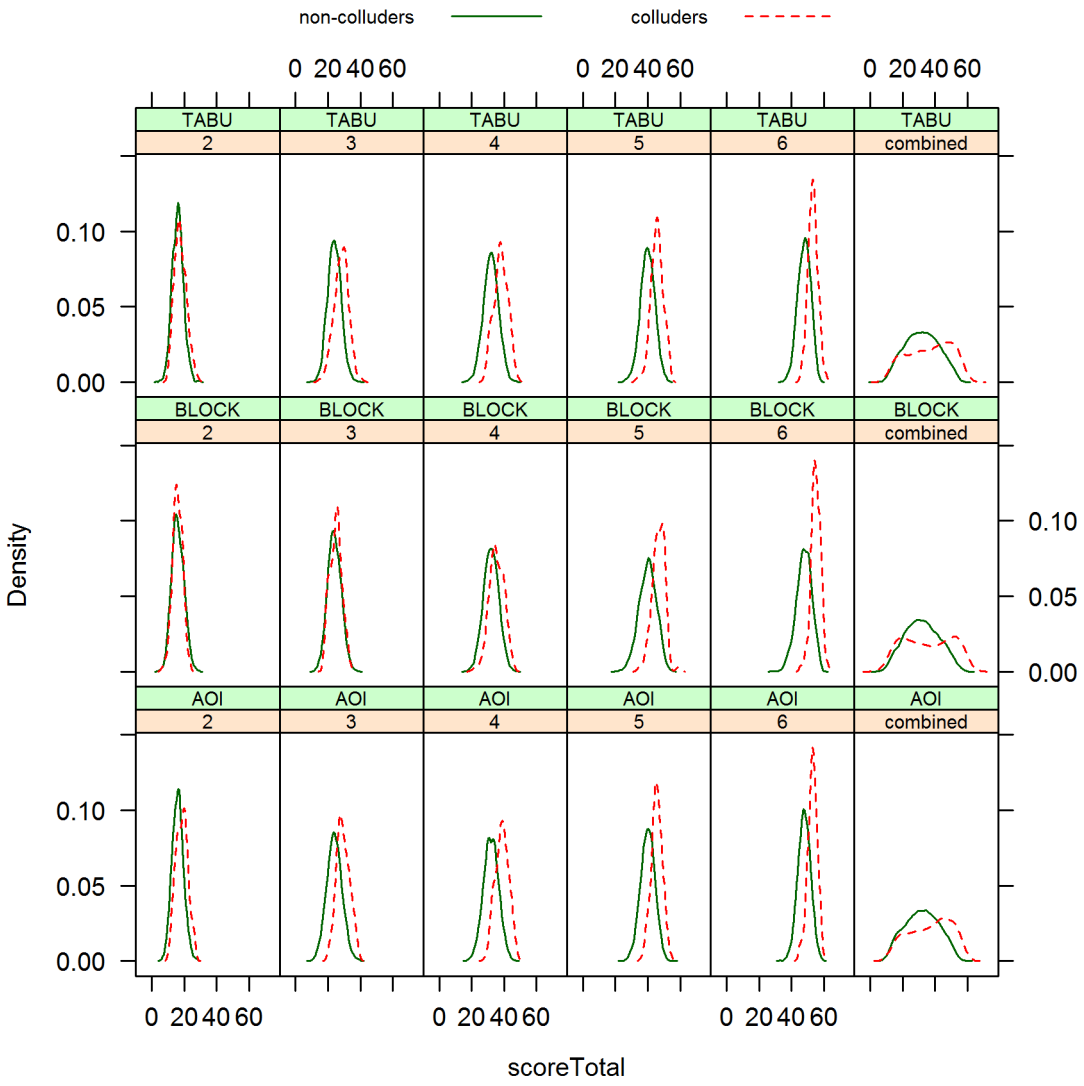

Figure 3 Density estimates of the *scoreTotal* for different number and type of colluders.

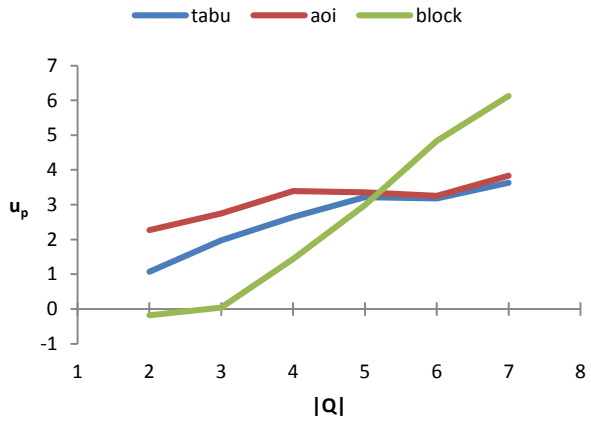

Figure 4 Collusion pay-off in the dispenser setting [13]

3.1.2. Shooting

Colluding players engage strongly in soft play by refusing to shoot each other at all, which can be seen in that the shooting related features indicate collusion very clearly (especially *hitsAllDelta, hitsMutual* and *hitsMutualDelta*) (see Figure 5). Because the colluding players do not shoot each other, there are very few mutual hits. This also affects the total number of hits, because there are fewer targets to fire at.

The information gain with the full datasets is somewhat worse than for a specific number of colluders. The distribution of values varies with the number of colluders and while the classes are separate for specific number of colluders they overlap in the full datasets (Figure 6).

3.1.3. Direction

Most of the direction-based features have very small gain, but some stand out with a large gain, especially for AOI colluders (see Figure 7).

DirectionsMin has a high gain for all collusion types with 2 colluders. For a random set of players, we might expect that value for *directionsMin* would be zero, but colluders do not move freely in relation to each other and a common direction can be found (Figure 8). However, this effect is very small and diminishes when the number of colluders increases. Also, the AOI colluders may have this effect when the partitions do not have uniform dimensions, because players have more room to move in one direction than another. For two AOI colluders also *directionsStdDev* has a high gain. Two AOI colluders have more deviation in mean direction than non-colluders (Figure 9), but the difference diminishes with larger number of colluders or more uniform division

AngleStdDev has a high gain for three AOI colluders and they have larger standard deviation than non-colluders (Figure 10). *AngleMean* has a high gain for five AOI colluders and a moderate gain for three AOI colluders. For three colluders *angleMean* is larger than non-colluders and for five colluders *angleMean* is smaller than for non-colluders (Figure 11).

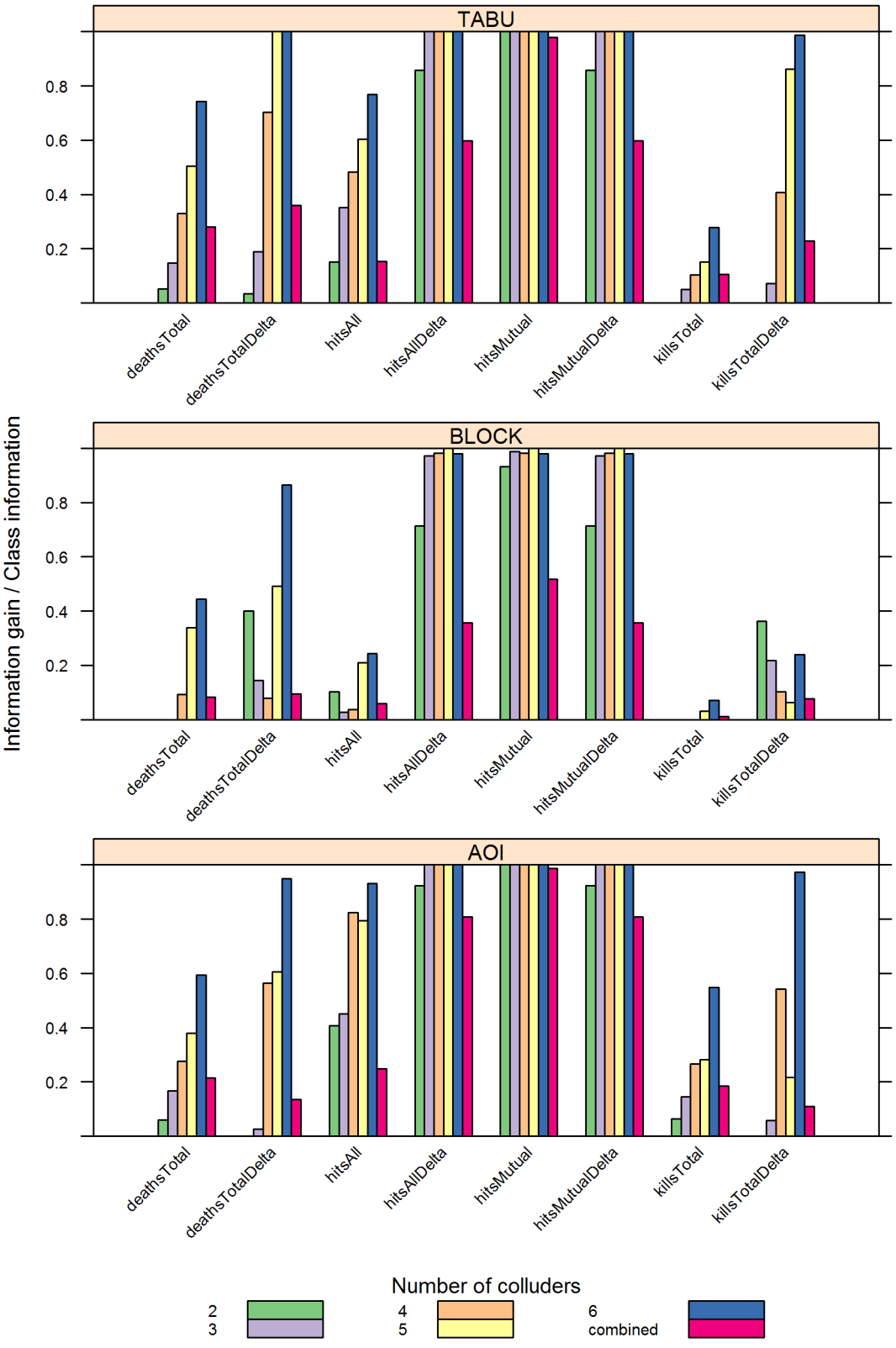

Figure 5 Information gain / class information of the shooting features

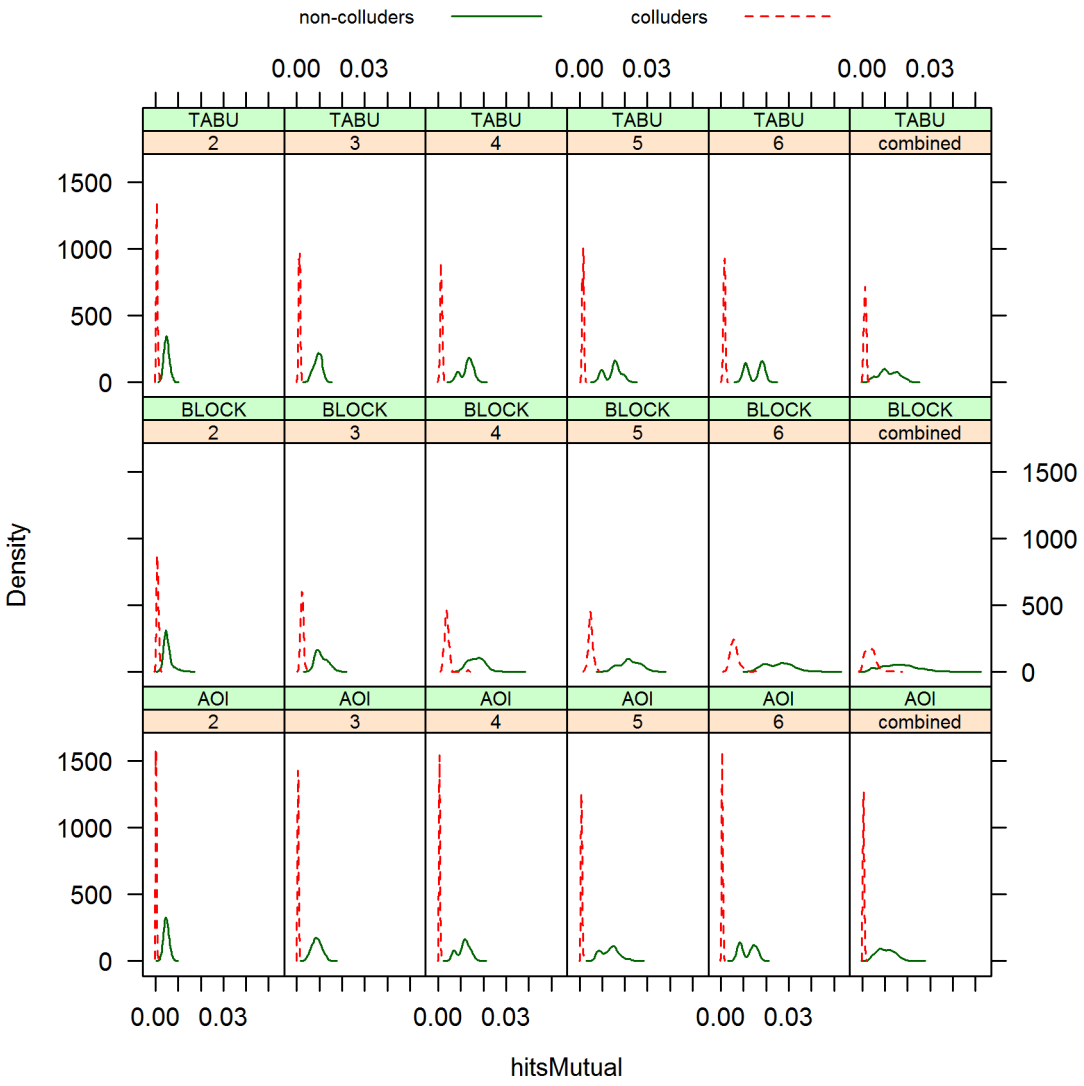

Figure 6 Density estimates of *hitsMitual* with different number and type of colluders.

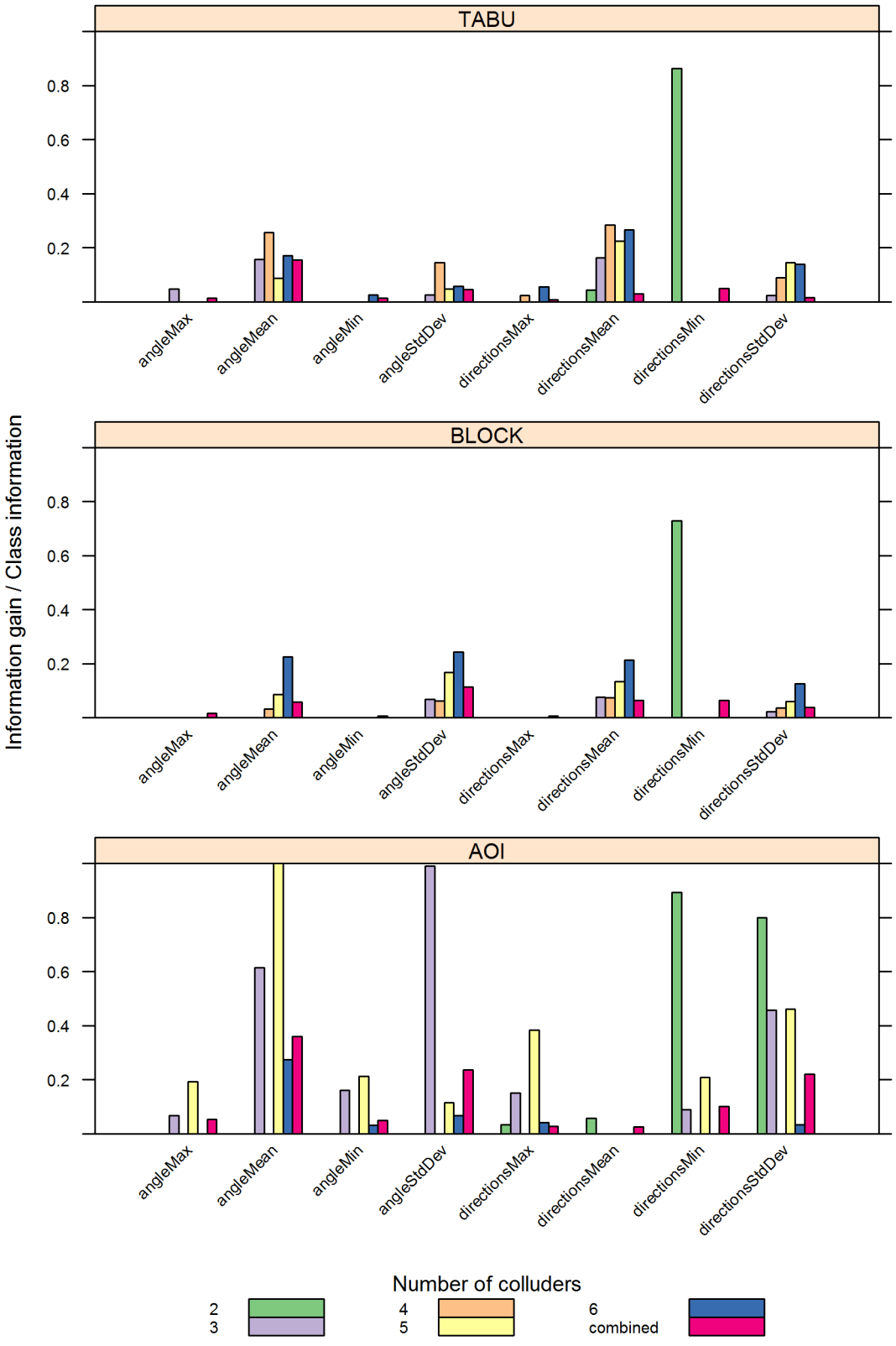

Figure 7 Direction features

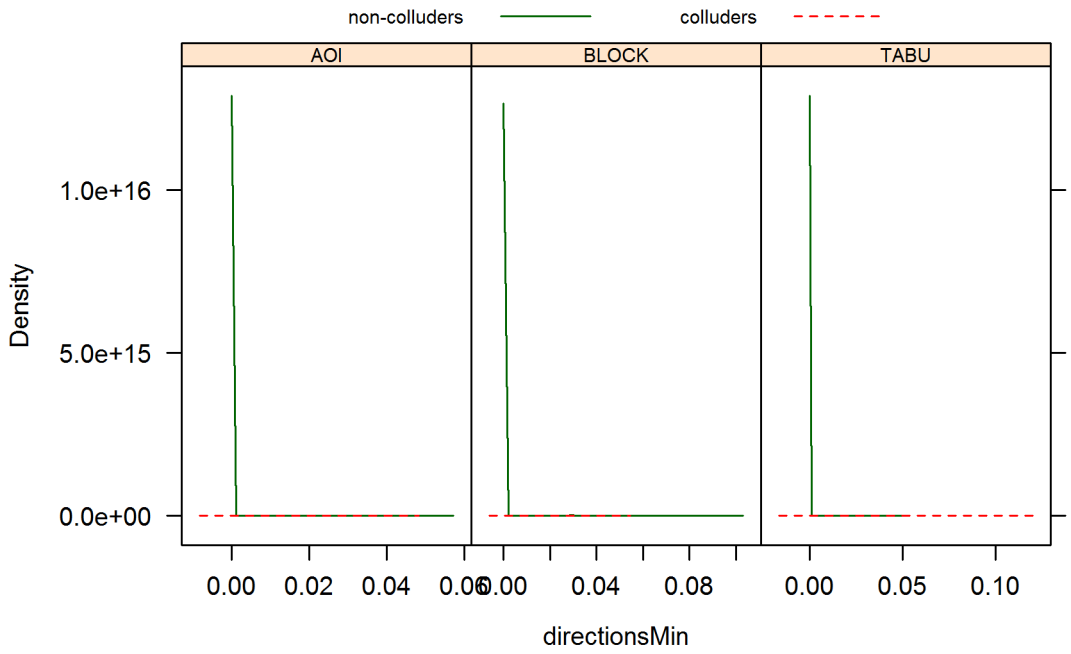

Figure 8 Density estimates of directionsMin for 2 colluders

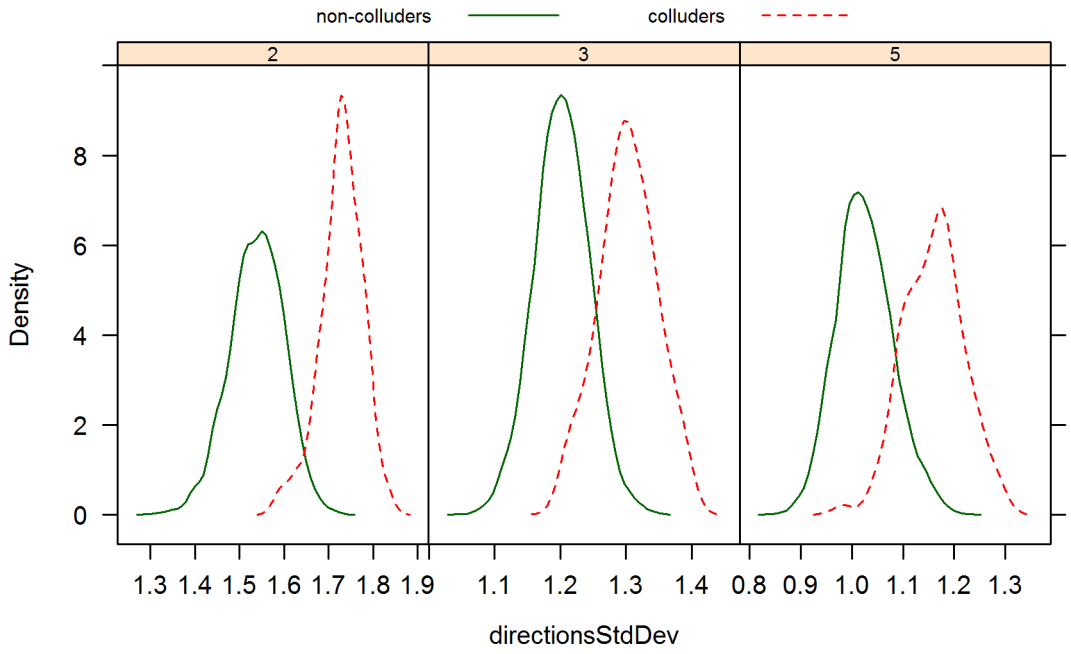

Figure 9 directionStdDev density of 4 and 6 AOI colluders

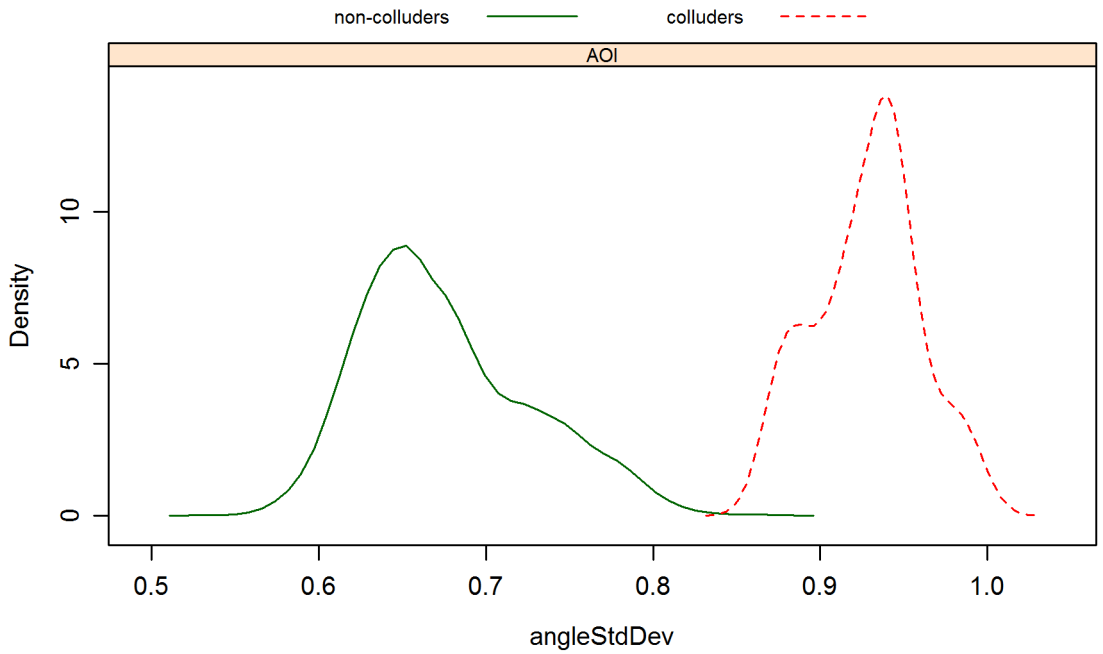

Figure 10 *angleStdv* density estimate for 3 AOI colluders

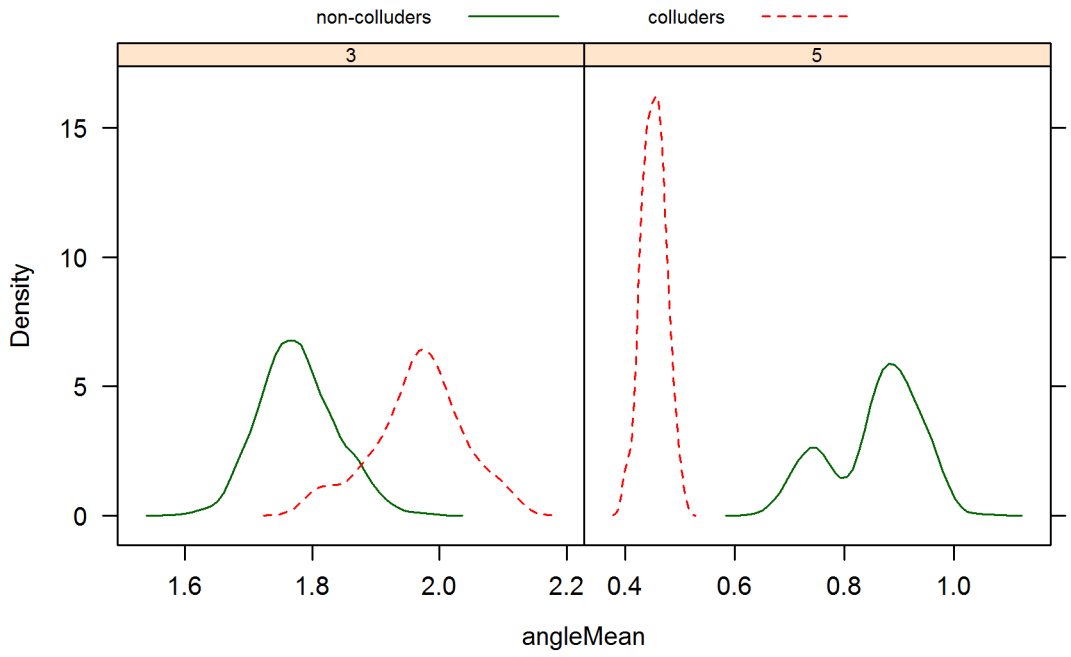

Figure 11 *angleMean* density estimate for 3 and 5 AOI colluders

3.1.4. Location

Location-based features have almost a zero gain for blocking and tabu colluders, but very high gain on some features for AOI colluder (see Figure 13). *MeanXStdDev* has a high

gain with all numbers of AOI colluders while *meanYStdDev* has a high gain only for 4 and 6 AOI colluders. The game world is always divided in the x-direction, and because of that the agents have more even distribution in the x-direction (see Figure 12). For the y-direction, division happens only with 4 and 6 colluders. These features work well with our division, but, for example, rotating the division would make them less useful.

Correlation and c*orrelationStdDev* have a high gain with 4 colluders and moderate with 6 colluders. The division used with 4 and 6 colluders forces the agents to be distributed more uniformly and fixates them to a lattice-like structure which has a low correlation between the agent locations (see Figure 15 and Figure 16). This leads also lower standard deviation in *correlation* (see Figure 14). When a division occurs only in one dimension, it allows the agents also to have highly correlating positions.

Coordinate and correlation features indicate AOI collusion strongly, but changing the division or making it vary over different games would make them less useful.

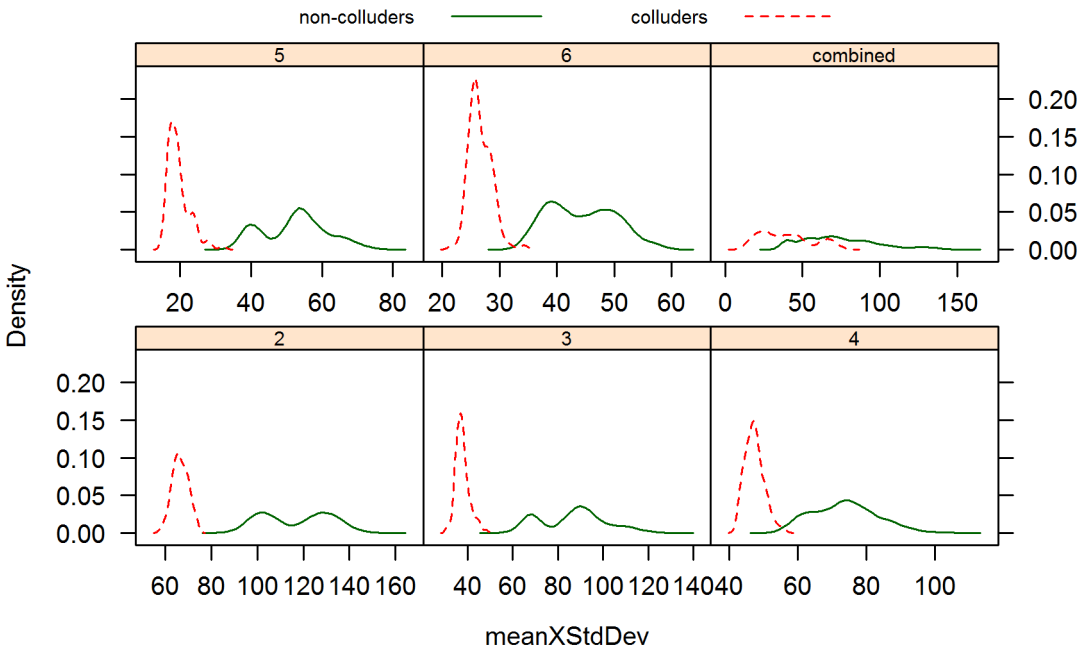

Figure 12 Density estimates of meanXStdDev of AOI colluders

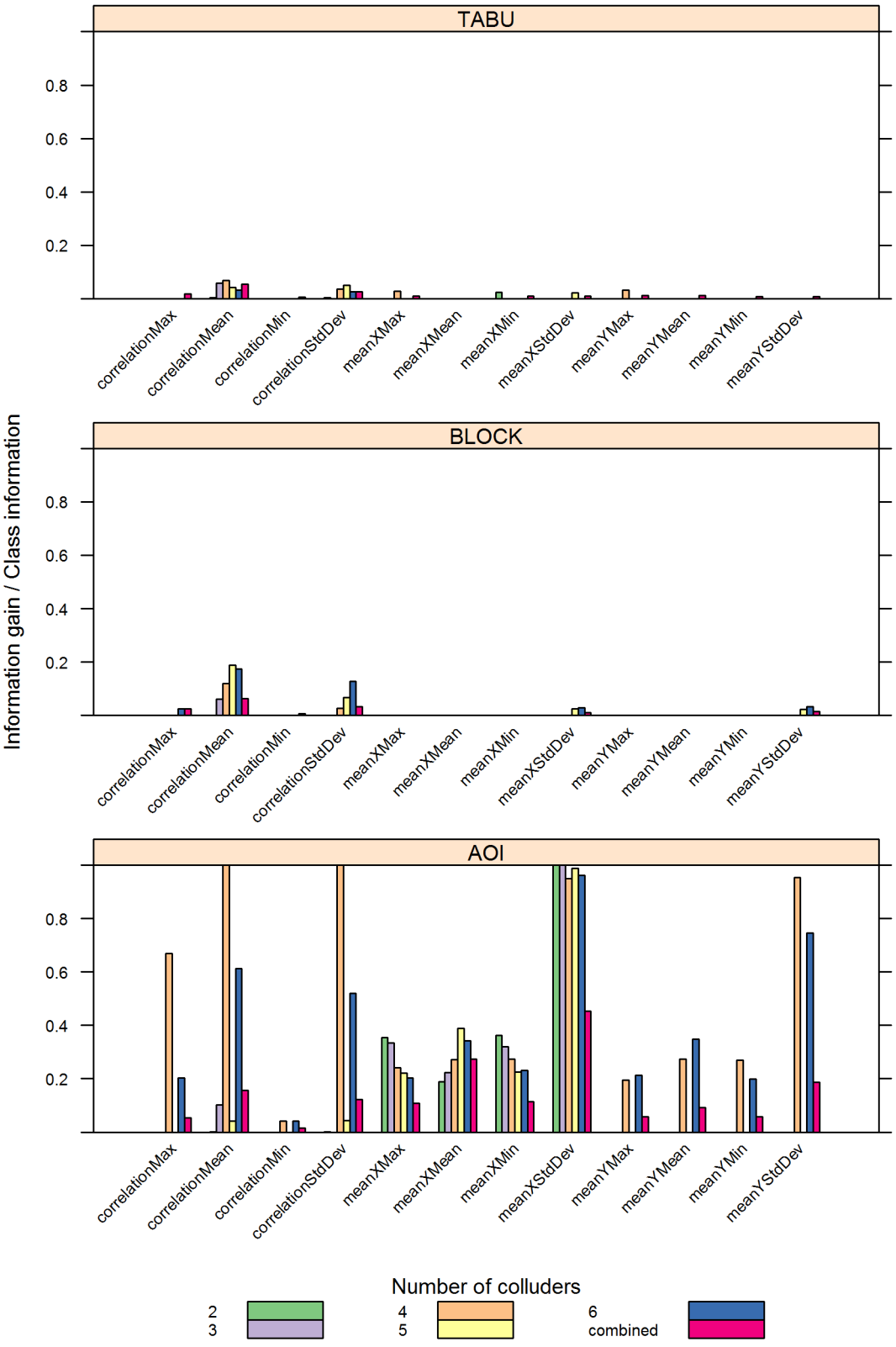

Figure 13 Location features

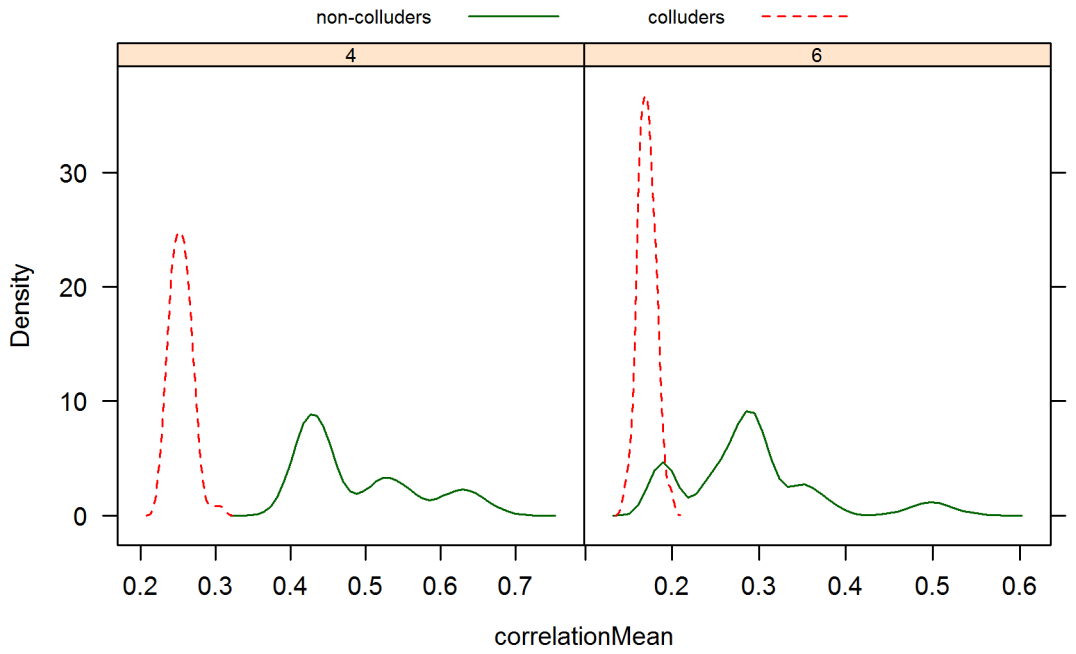

Figure 14 Density estimates of correlationMean for 4 and 6 AOI colluders

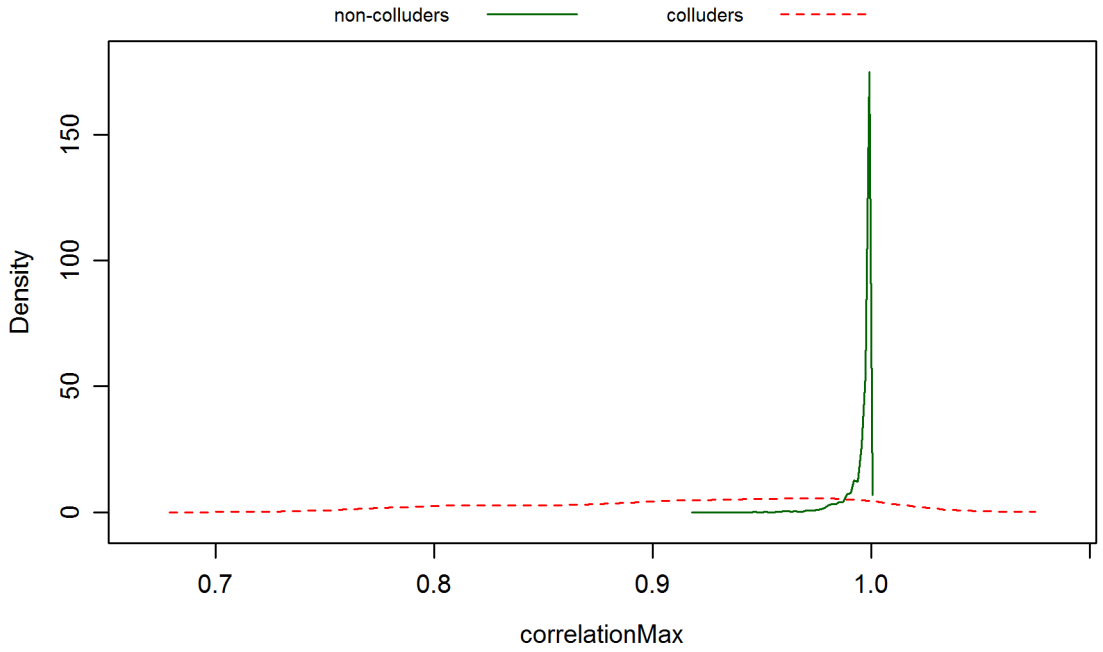

Figure 15 Density estimation of correlationMax for 4 AOI colluders

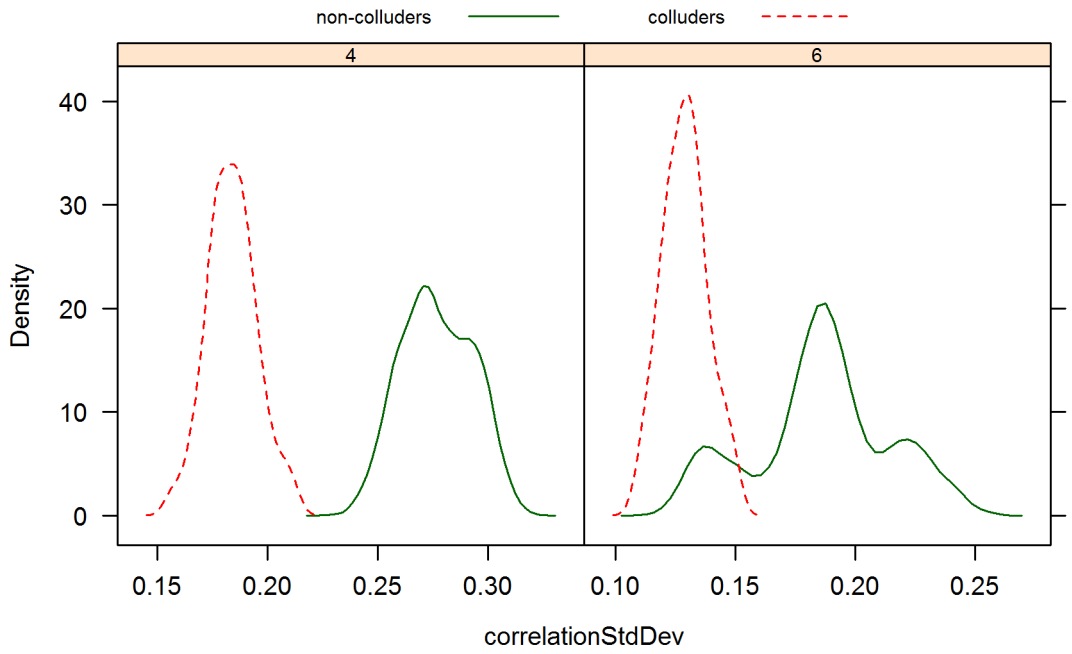

Figure 16 Density estimates of correlationStdDev for 4 and 6 AOI colluders

3.1.5. Area

The area-based features have small gains for blocking and tabu colluders. *DistanceMean* and *unionMean* have high gains for two and four AOI colluders. *intersectionMean* and *intrsectionStdDev* have high gain for two AOI colluders. These features have a larger value when colluders use grid-like division of the game world (Figures 13 and 14). With a small number of colluders there is naturally less variation in the distances between the colluders. When the number of colluders increases and the areas of interest get smaller, the bounding boxes of the colluders tend to overlap more often.

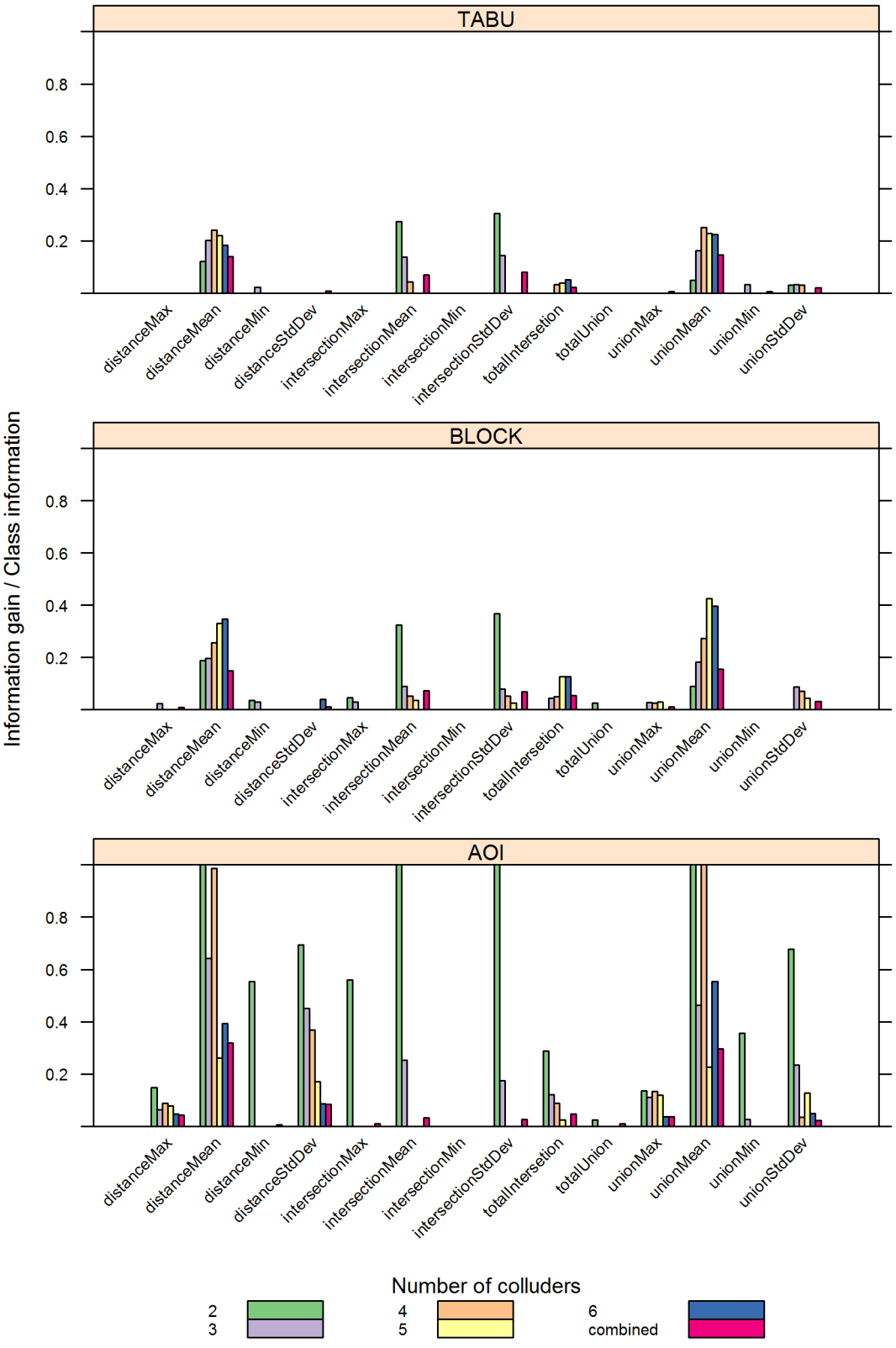

Figure 17 Area features

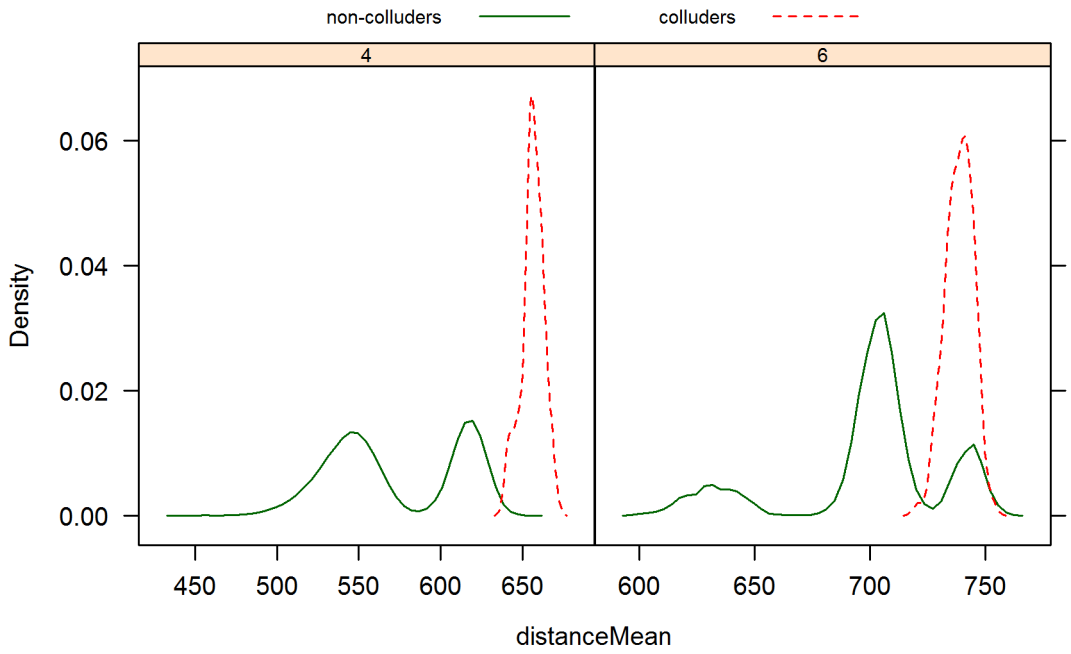

Figure 18 distanceMean density of 2 and 4 AOI colluders

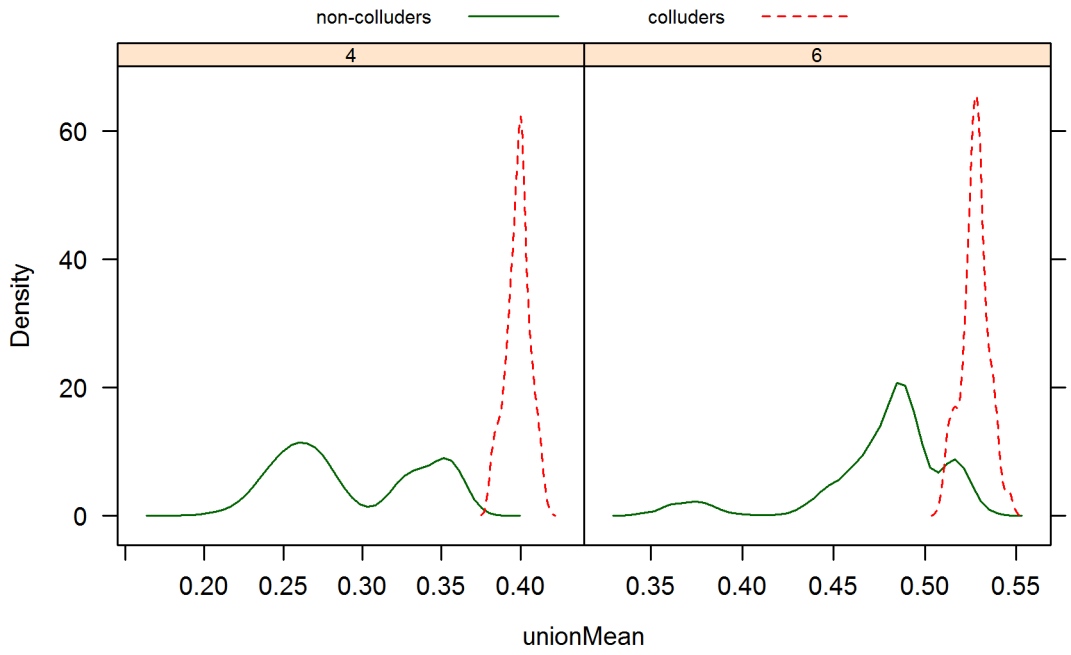

Figure 19 unionMean density estimate for 2 and 4 AOI colluders

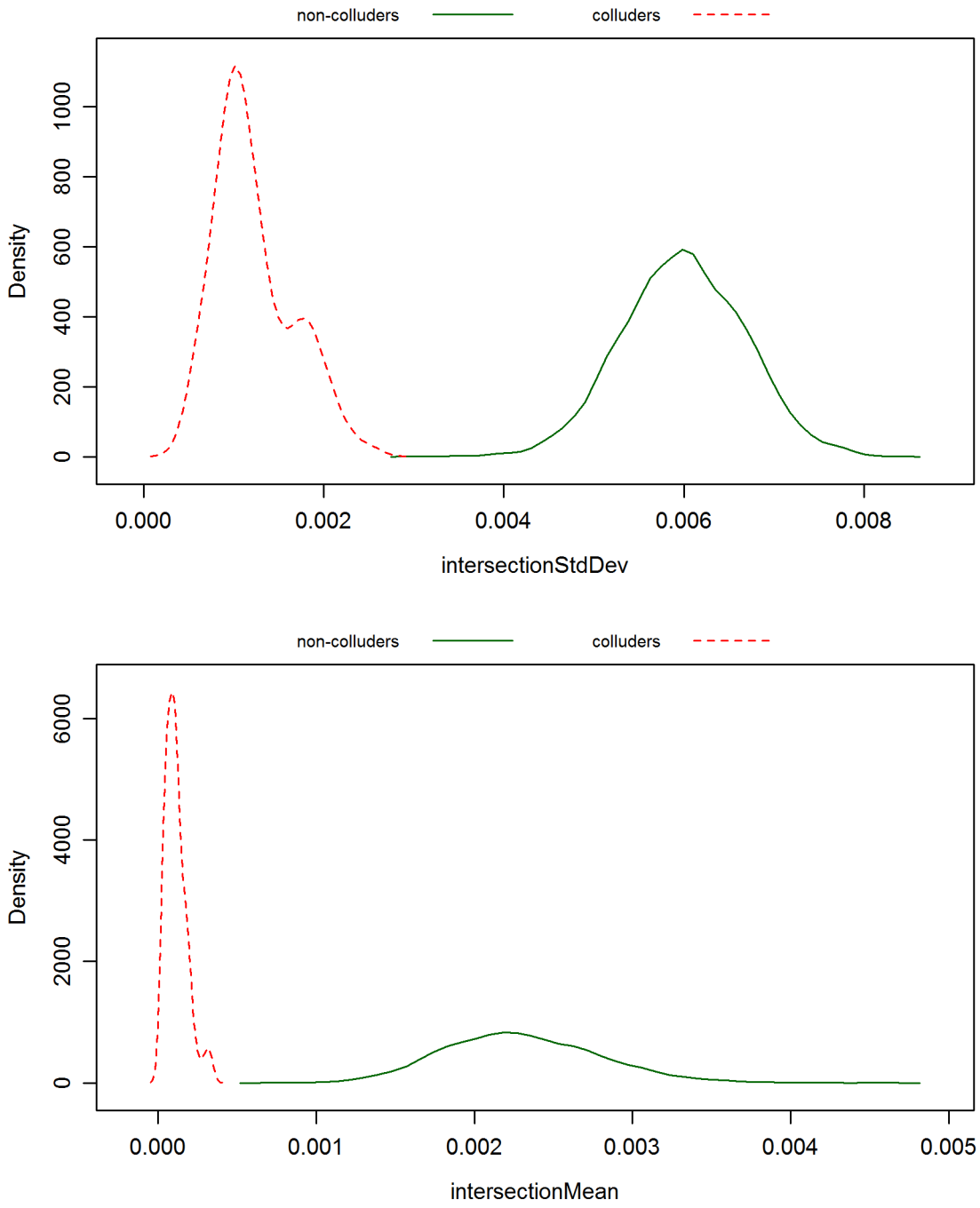

Figure 20 Density estimates of intersection features for 2 AOI colluders

3.2. Detecting the colluding subset

For colluding subset detection we omit the shooting related features from all of the datasets, because soft play is very easy to detect. We created J4.8 decision tree classifiers from the data. We generated one classifier for each colluder type and number of colluders between 2 and 6, and also for the full data. The sizes of the resulting decision trees are presented in Figure 21.

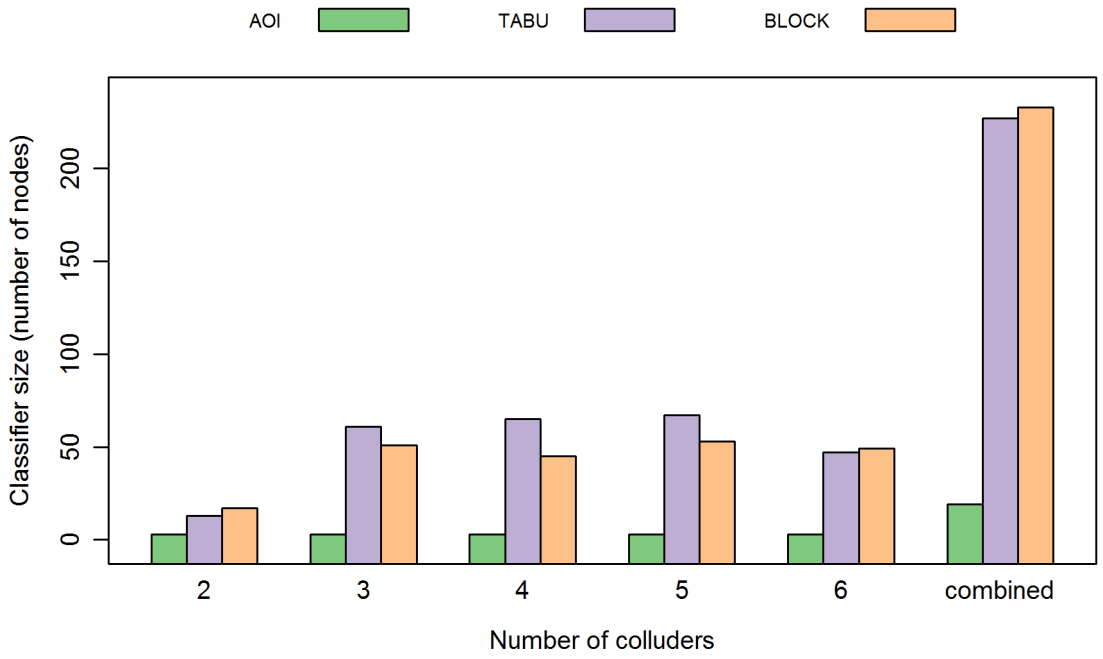

Figure 21 Classifier size (nodes)

The overall performance seems good, because most of the instances are non-colluding subsets. They are classified correctly but colluders have a large error in most cases. Only the classifiers for AOI colluders show good classification performance across every number of colluders. Recall and precision of the colluders are plotted in Figure 22. The two-colluder case was the only one with good results for all colluders

The classifiers for the tabu and blocking colluders have precision and recall around 80% for the two-colluder case. For real applications this might still be acceptable level of accuracy compared to no detection at all, but it will require human screening to filter out false positives.

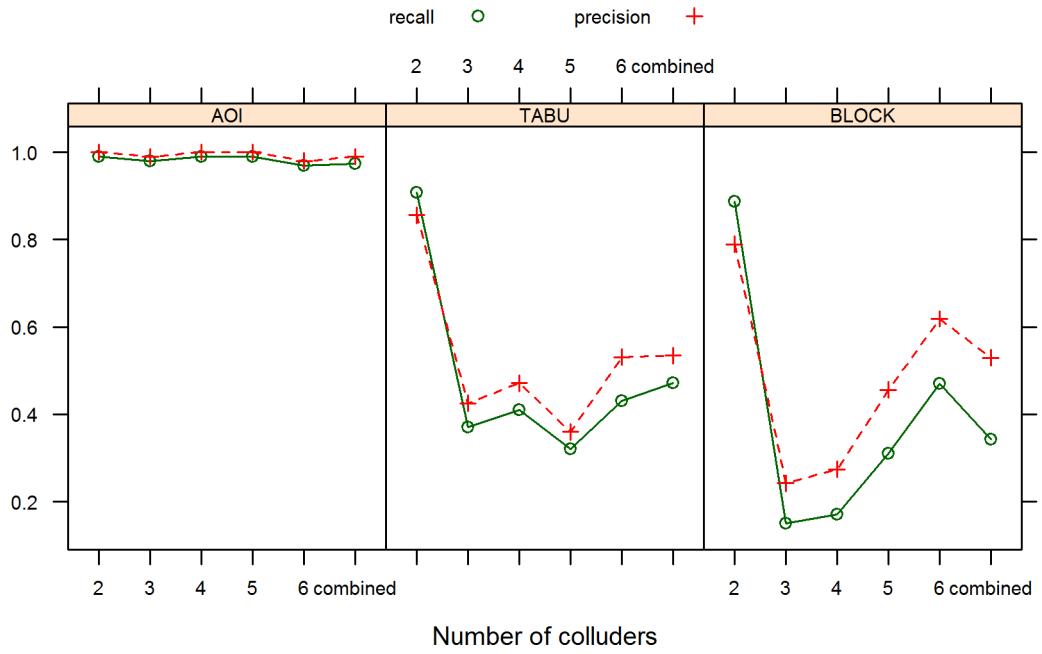

Figure 22 Colluders precision and recall

Example decision trees for AOI colluder are presented in figures 27 to 30, for tabu colluder in Figure 29 and for blocking colluder in Figure 30. The nodes contain the name of the feature and the lines connecting the nodes have the value of the feature for that branch. Leafs are represented as grey rectangles and contain the value of classification: 1 for colluders and 0 for non-colluders. Beneath the class value is the number of training instances for the leaf. In only one number is presented all of the instances are of the predicted class. If two numbers are separated by a slash first number is the number of instances with the predicted class and the second number is the number of instances of the other class. Different decision trees are possible, due learning algorithm parameters. Also, in some datasets there are multiple features with enough information to separate the classes completely.

Leaf nodes are displayed as rectangles and they contain the following classification: 0 for non-colluders and 1 for colluders and the number of training instances for the leaf. In case of misclassified instances their number is displayed after slash. For example, in the decision tree for four AOI colluders (Figure 23) the root node has the feature *intersectionMean* and all instances with value larger than 0 go to the right sub tree and are classified as non-colluders. Instances with *intersectionMean* equal to or smaller than 0 go to the left sub tree and are classified as colluders. In this case both leaf nodes are pure.

For three AOI colluders classification is based on *maxXStdDev*. Because the colluders stay on their respective areas and the areas are large compared to the whole game world there is little overlap in the player's bounding boxes. So the value for *union* is often two times the bounding box area.

For two tabu and blocking colluders the most of the non-colluder instances are decided by the root node. The *directionsMin* feature was the only one with high information gain and separates most of non-colluders.

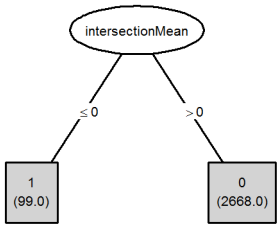

Figure 23 Decision tree for two AOI colluders

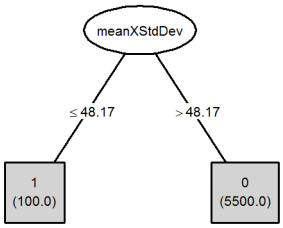

Figure 24 Decision tree for three AOI colluders

Figure 25 Decision tree for four AOI colluders

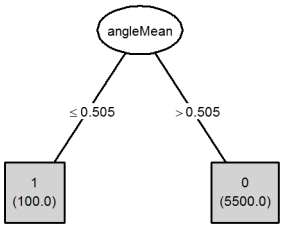

Figure 26 Decision tree for five AOI colluders

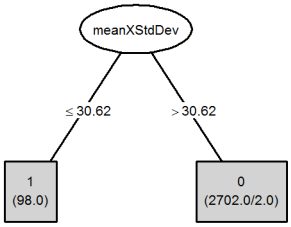

Figure 27 Decision tree for six AOI colluders

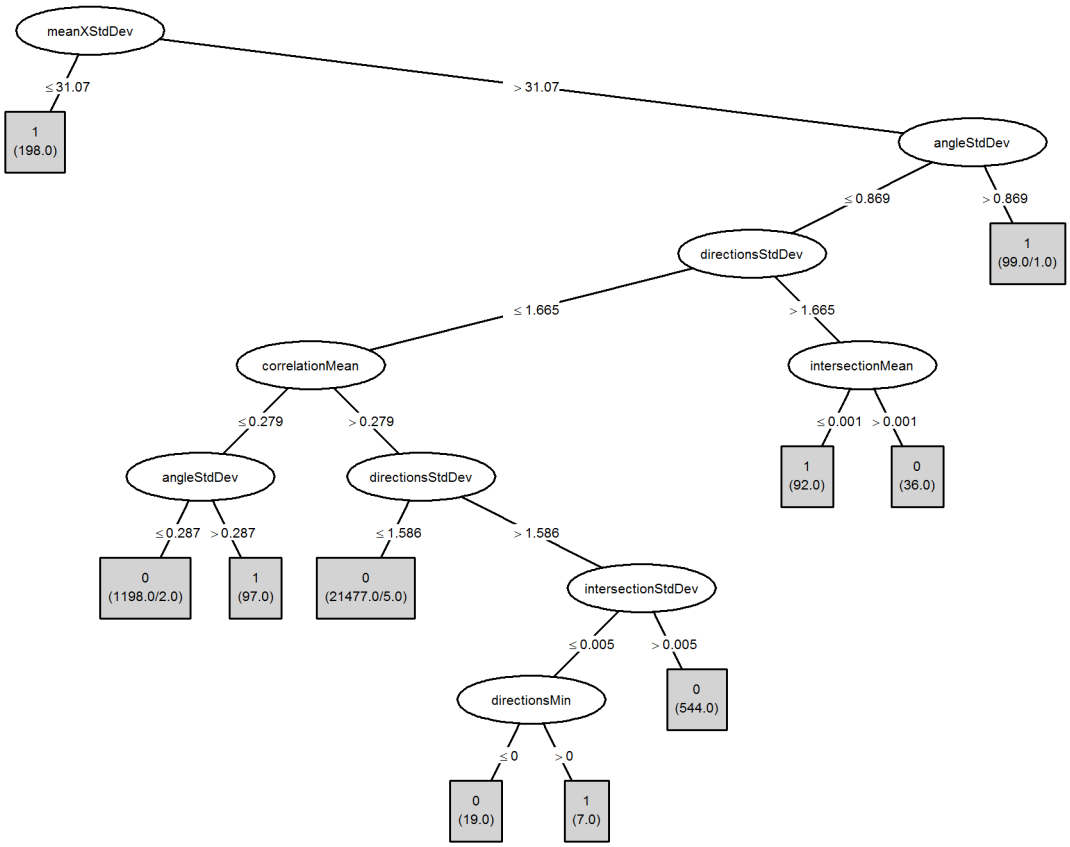

Figure 28 Decision tree for any number of AOI colluders

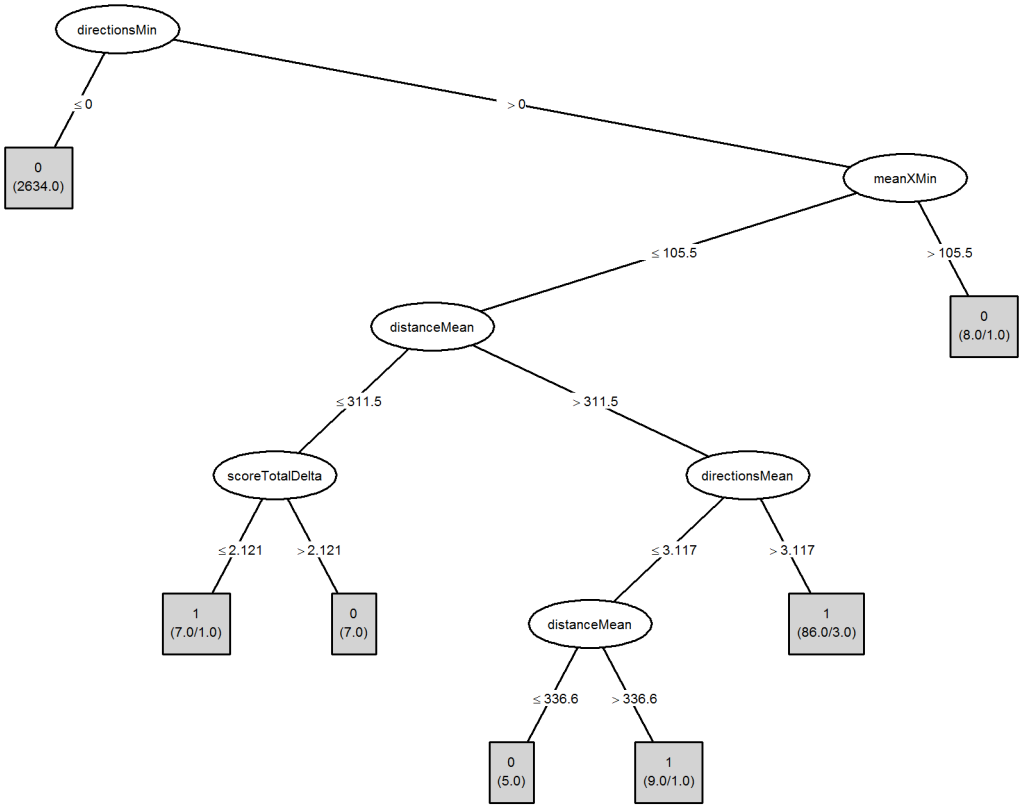

Figure 29 Decision tree for two tabu colluders

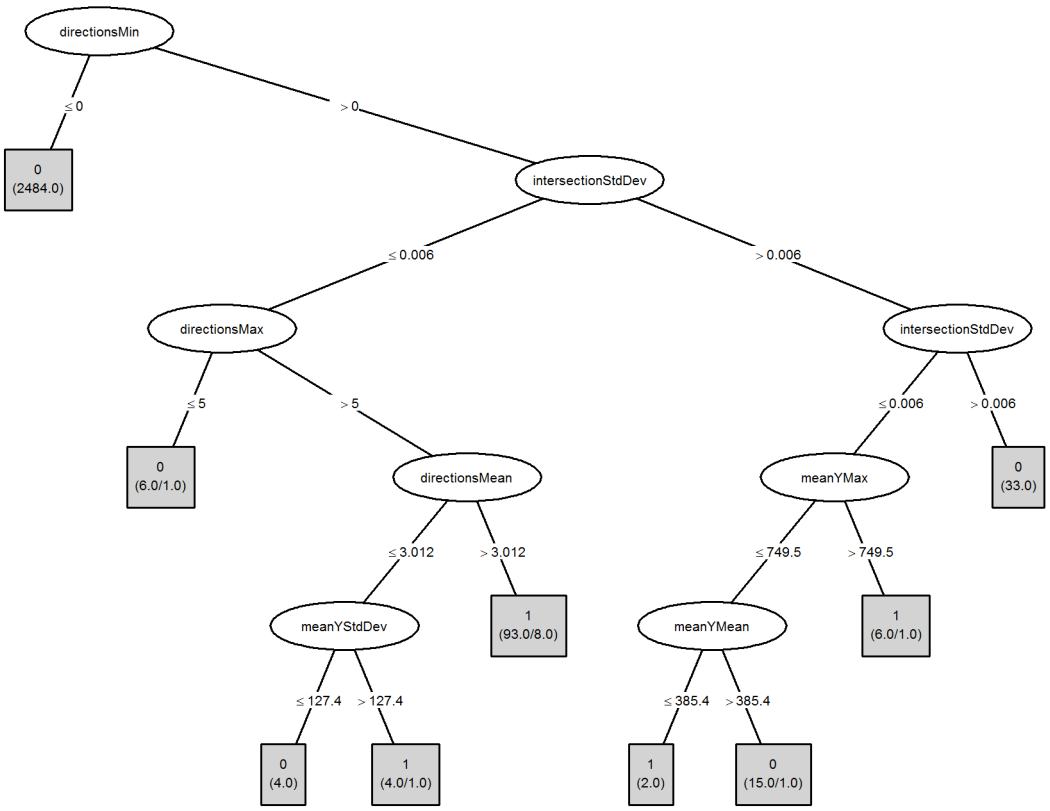

Figure 30 Decision tree for two blocking colluders

When we compare the collusion pay-off [13]

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

for different number of colluders (see Figure 31) to the classification performance in Figure 22. We observe that the maximum and minimum of the collusion payoff and classification accuracy are almost co-located. Classification seems to correlate negatively with the collusion pay-off. It is somewhat surprising that it is more difficult to recognize colluders when the gain from collusion is at its largest. Figure 32 shows the relationship between the gain from collusion and precision and recall of the learned classifiers. For the tabu colluder the relationship is statistically significant (P_{recall} = 0.0117, $P_{precision}$ = 0.0197).

One explanation for this is that collusion pay-off has the maximum around 4 colluders which is even distribution between colluders and non-colluders and this could be the reason for classification minimum. To verify this we need more experiments with larger number of players. We expect the maximum pay-off to appear with fairly small number of colluders so with a large number of players the pay-off should peak when $|Q|$ is

relatively small and the classification performance should have minimum when colluders are evenly distributed, but more experiments are needed to confirm that.

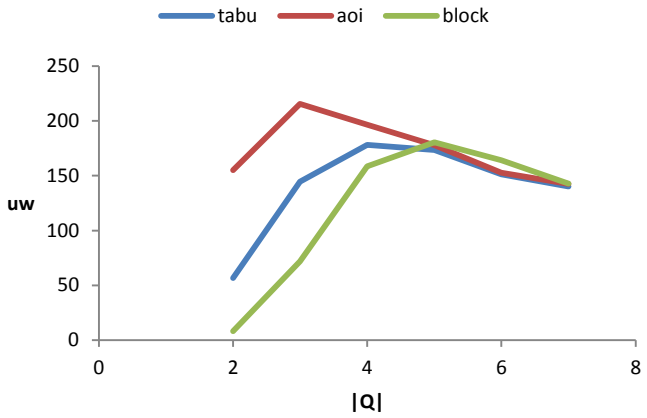

Figure 31 Collusion payoff in the dispenser setting [13]

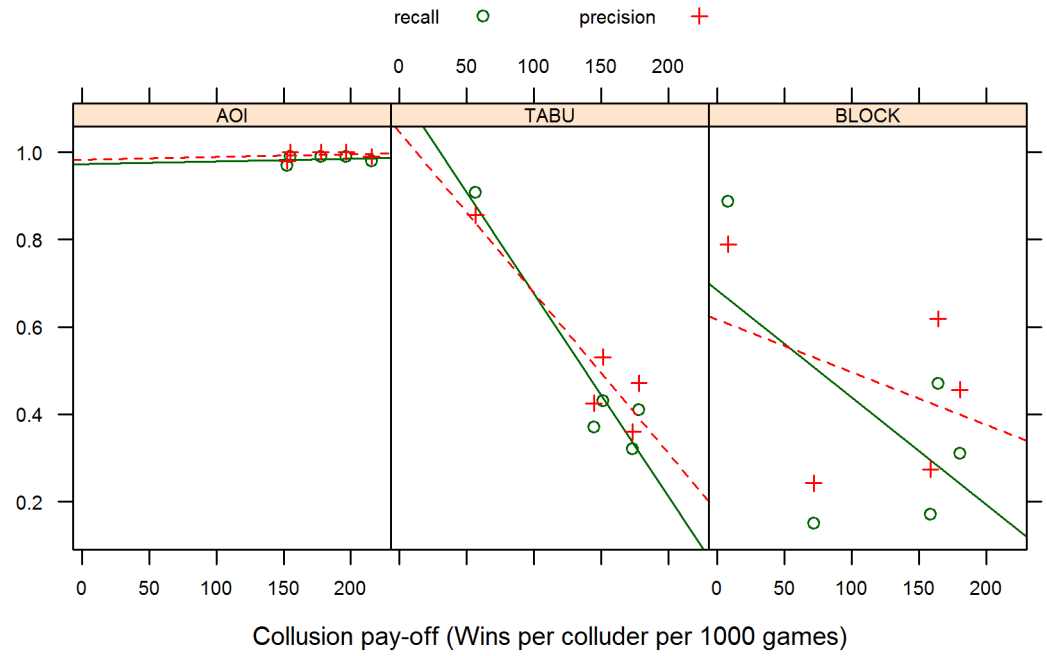

Figure 32 Correlation between collusion pay off and classification performance

When the number of colluders increases this kind of detection becomes more difficult because more subsets with both colluders and non-colluders appear. When subset size is large, some of the subsets contain mostly colluders and the feature values for these mixed sets overlap easily with the colluding subsets.

Instead of using binary classifier, a numerical classifier could be used to get a collusion rating. Subsets with sufficiently high rating would be tagged suspicious and selected for

further analysis. For example, we could check which players appear in a large number of suspicious sets.

3.3. Swiftness of the proposed method

The feature generation were performed on a computer with Intel Core 2 Duo CPU at 2.66 GHz and 4 GB of memory running the Windows 7 operating system. The test application was implemented with Java 7. The classification experiments were performed on a computer with AMD FX 8320 CPU at 3.5 GHz and 16 GB of memory running the Windows 8 operating system.

Calculating the features for eight players and a limited subset size appears to be fast. The time required to calculate the features for a single time step is between 0.6 and 1.7 milliseconds depending on the subset size (Figure 33). However, in reality the features for multiple subset sizes are required because the number of the colluders is unknown. In this case, it would increase the required time from under 1.8 milliseconds to 6.0 milliseconds. Compared to the 0.3 milliseconds required to perform the simulation step the time required to calculate features is large.

Considering that to play modern first person shooters smoothly requires a frame rate of at least 40 frames per second (25 milliseconds per frame) contributing several milliseconds to collusion detection is not feasible. This places constraints on how the feature generation is implemented. If a dedicated server is used, this can be generated on the server side, but if features are generated on the client side, the frequency of the feature collection needs to be lower than the simulation or graphics update frequency or the task needs to be distributed between the clients.

Learning a decision tree classifier from the features took under 1 second for any number of colluders and around 5 seconds for a full dataset (Figure 34). The number of instances in the full datasets and with a given number of colluders are presented in Table 1. The time required for applying the classifier is not given by Weka, but using decision trees should be fast.

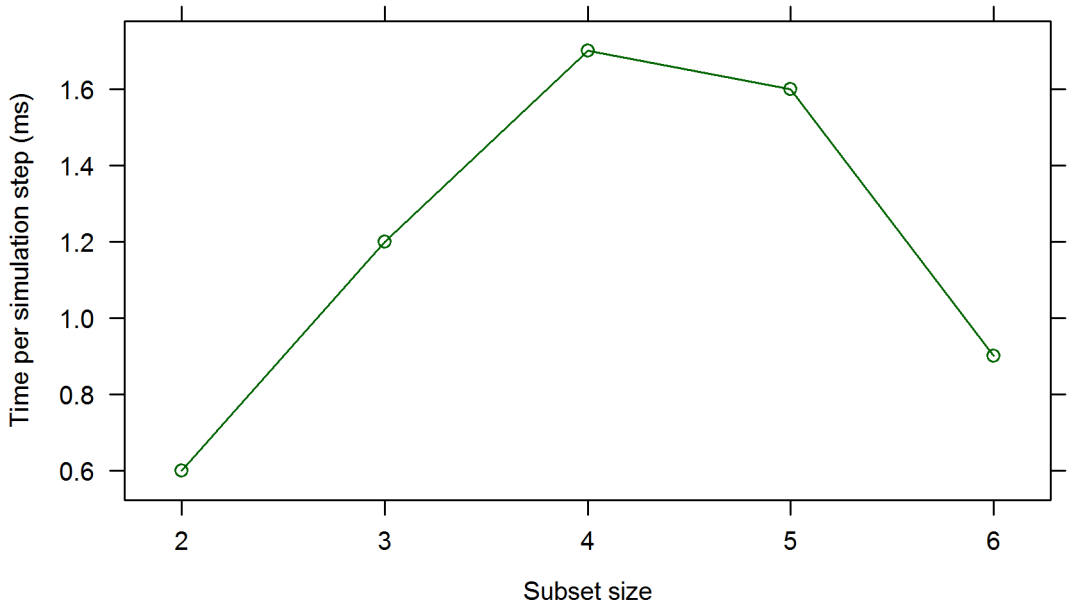

Figure 33 Time required for calculating the features.

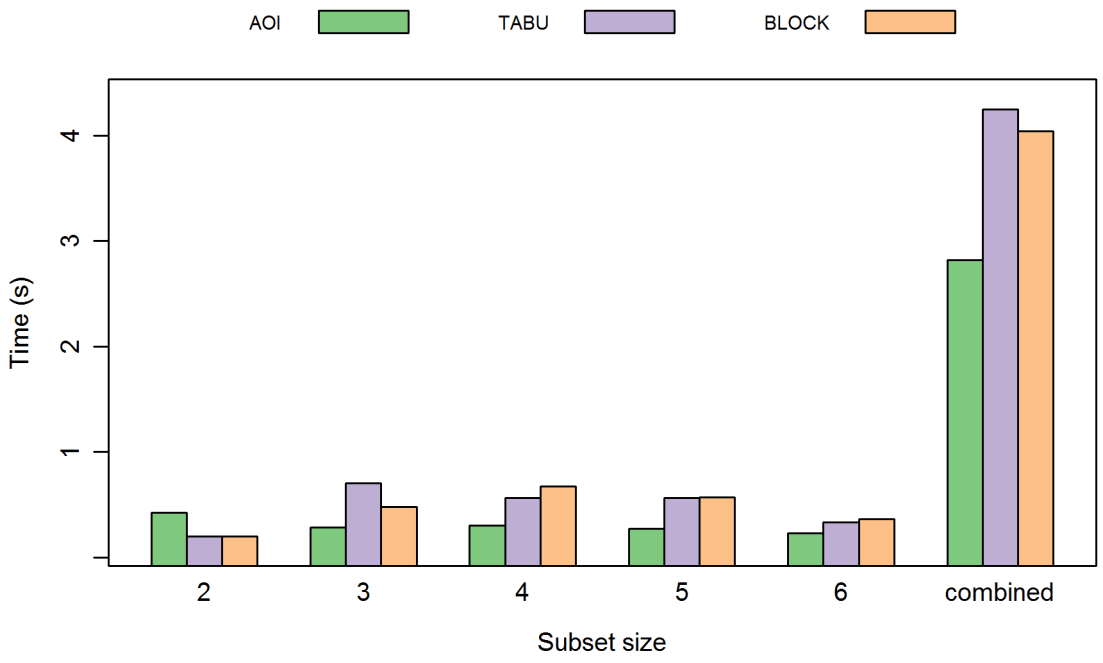

Figure 34 Time required to build the classifier model

3.4. Summary

Shooting-based features, especially the ones related to mutual hits, have very high gains. Majority of the proposed features are not good for tabu or blocking collusion. Only *directionsMin* has significant gain in 2 colluder case. Several of the features have high gains for AOI colluders, but gains depend on the number of colluders and division in the game world. It is expectable that these features would not generalize well to other divisions.

We were able to detect area-of-interest colluders in call test cases and tabu and blocking in two-colluder case with J4.8 decision tree classifier built based on the proposed features. We found that there is a statistically significant inverse relationship with collusion pay-off and classifier performance. However, we expect this to be an artifact of our test setup.

Our method was reasonably fast. Building the decision trees and classification requires negligible time, considering that they can be performed off-line. Feature generation on – while fast with small number of subsets – can cause performance problems in resource tight environments or when all features are required for all possible subsets and may require that feature generation is skipped for some time-steps.

4. Conclusions

In this paper, we analysed information gains of the proposed collusion features [13] in Pakuhaku game [14] using different number of colluders. We have demonstrated that for simple cases it is possible to devise a collusion detection method by defining a suitable set of features and using J4.8 decision tree classifiers to detect a colluding subset. We also analysed the performance of the feature generation and collusion detection by using the datasets from [17].

Soft play is easy to detect with decision tree classifiers and the proposed features. Features *hitsAllDelta*, *hitsMutual* and *hitsMutualDelta* have a very high information gain with all colluder types. This was expected because the colluders do not shoot each other at all. While the proposed features indicate collusion for all cases, the distribution of feature values is different with different number or type of colluders. Also, further experiments are needed for more subtle forms of soft play.

Several of the proposed features had high information for area-if-interest collusion with pre-set non-overlapping rectangular partitioning and it is easy to detect. The J4.8 decision tree classifier learnt from proposed features had nearly 100% precission and recall for colluding subset. Tabu and blocking colluders had low gains with the exception of *directionsMin* with two colluders and were difficult to detect except the two-colluder case, which had around 80% precision and recall for colluding subset.

The most promising features are: *angleMean*, *angleStdDev*, *directionsMean*, *directionsStdDev*, *directionsMin*, *directionsMax*, *distanceMin*, *distanceMax*, *unionMean*, *unionMin*, *totalUnion*, *totalIntersection*, *correlationMean*, *correlationStdDev*. Again, the classifier learnt from certain configuration works poorly on others. Our proposed method do not generalize well over different number of colluders. Even if the feature has high information gain, several number of colluders the distribution of values can be different and it is difficult to separate the colluders without knowing their exact number.

It could be possible to learn a more general classifier by using union of data from different number of colluders, but the size of the decision tree is much larger and it is expected that this classifier would perform poorly on situations not present in the original dataset. It could also be possible to achieve similar results by aggregating setup specific classifiers. This would allow updating the classifier incrementally when classifiers for new setups become available.

We detected an inverse correlation between collusion pay-off and the accuracy of colluding subset accuracy with tabu and blocking colluders. The pay-off per colluder peaks around 4 colluders, which is also minimum for detection accuracy. We surmise that accuracy minimum is due uniform distribution of colluders and non-colluders and just coincides with pay-off maximum in the used setting, but further research is required to verify that.

The learning and using decision tree classifiers is reasonably fast and can be done offline if required. Feature generation for games with a large number of players can easily become infeasible in practice, because features need to be calculated for $|\mathcal{P}(S)| \in O(c^n)$ subsets. Also the mixed subsets with both colluding and non-colluding players make the detection more difficult when the number of colluders is large. To reduce overhead of feature generation we could use only subset sizes with large predicted collusion pay-off or small subsets in general or calculate feature values with smaller frequency than the game state updates.

4.1. Future work

The scenarios analysed in this paper are very simplistic and we seek to move towards more realistic settings. However, the problem of collusion detection is difficult even in these simple cases. We aim at making our test settings and synthetic players incrementally more complex. Eventually we want to perform experiments with human players. As we move towards more realistic scenarios, we also need to update our test platform used to generate the test data. Our current *Java* implementation is still quite far from a real multiplayer game and we plan to create new test platform using a real 3D game engine.

In this paper, we demonstrated colluding subset detection using features calculated from for each subset. With larger number of players and unknown number of colluders the number of possible subsets becomes too large for this kind of approach to be feasible. We plan to apply collusion clustering algorithms [4], [5]. Then it is required to calculate collusion index only for candidate subsets found be the algorithm. Before we can apply

it to our problem, a collusion index and a pairwise metric for the weights in the collusion graph have to be prepared. The collusion features analysed in this paper are will be our starting point for the collusion index and a numeric model tree from our data sets might be used to calculate the collusion index.

In this paper, we have analysed features for collusion detection but with suitable features similar approach could be used in detecting different kinds of player behaviour for example this could be other unwanted activity like griefing [21], encouraged to activity to distinguish good players or player modelling in general.

References

[1] S. J. Murdoch and P. Zielinski, "Covert channels for collusion in online computer games," in Information Hiding, vol. 3200, J. Fridrich, Ed. Springer-Verlag Berlin, 2004, pp. 355–369.

[2] 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.

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

[4] G. Keshav and P. Manoj, "Collusion set detection using graph clustering," Data Mining and Knowledge Discovery, vol. 16, no. 2, pp. 135–164, 2008.

[5] 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, 2009, pp. 237–242.

[6] J.-K. Lou, K.-T. Chen, and C.-L. Lei, "A Collusion-Resistant Automation Scheme for Social Moderation Systems," 2009 6th IEEE Consumer Communications and Networking Conference, pp. 1–5, Jan. 2009.

[7] E. Staab and T. Engel, "Collusion Detection for Grid Computing," in 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 412–419.

[8] 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, 2010, p. 121.

[9] C. Vallve-Guionnet, “Finding colluders in card games,” in International Conference on Information Technology: Coding and Computing (ITCC’05) - Volume II, 2005, pp. 774–775 Vol. 2.

[10] J. J. Yan and H.-J. Choi, “Security issues in online games,” The Electronic Library, vol. 20, no. 2, pp. 125–133, 2002.

[11] R. V. Yampolskiy, “Detecting and Controlling Cheating in Online Poker,” in 2008 5th IEEE Consumer Communications and Networking Conference, 2008, pp. 848–853.

[12] J. Yan, “Collusion Detection in Online Bridge,” in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), 2010, pp. 1510–1515.

[13] J. Laasonen, T. Knuutila, and J. Smed, “Eliciting collusion features,” in Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, 2011, pp. 296–303.

[14] 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), 2007, pp. 103–107.

[15] F. Glover, “Tabu search-part I,” ORSA Journal on Computing, vol. 1, no. 3, pp. 190–206, 1989.

[16] “ARFF (stable version),” weka.wikispaces.com, 2012. [Online]. Available: http://weka.wikispaces.com/ARFF+(stable+version). [Accessed: 25.5.2012].

[17] J. Laasonen, “Collusion features from a simple two-dimensional game,” figshare. [Online]. Available: http://dx.doi.org/10.6084/m9.figshare.97804. [Accessed: 21.4.2013].

[18] “The R Project for Statistical Computing.” [Online]. Available: http://www.r-project.org. [Accessed: 16.3.2013].

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software,” ACM SIGKDD Explorations Newsletter, vol. 11, no. 1, p. 10, 2009.

[20] J. R. Quinlan, C4.5: Programs for Machine Learning, vol. 1, no. 3. Morgan Kaufmann, 1993, p. 302.

[21] 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. de Castell and J. Jenson, Eds. Peter Lang Publishing Inc, 2007, pp. 103–114.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN 978-952-12-2874-2
ISSN 1239-1891