Data Preprocessing of eSport Game Records
Counter-Strike: Global Offensive

David Bednárek, Martin Kruliš, Jakub Yaghob and Filip Zavoral
Charles University, Prague, Czech Republic

Keywords: e-sport, Data Analysis, Data Integration, Data Quality, Player Rating.

Abstract: Electronic sports or pro gaming have become very popular in this millenium and the increased value of this new industry is attracting investors with various interests. One of these interest is game betting, which requires player and team rating, game result predictions, and fraud detection techniques. In our work, we focus on preprocessing data of Counter-Strike: Global Offensive game in order to employ subsequent data analysis methods for quantifying player performance. The data preprocessing is difficult since the data format is complex and undocumented, the data quality of available sources is low, and there is no direct way how to match players from the recorded files with players listed on public boards such as HLTV website. We have summarized our experience from the data preprocessing and provide a way how to establish a player matching based on their metadata.

1 INTRODUCTION

Competition between people has been one of the defining characteristics of the entire human race. In the digital era, one of the domains where people express their competition is computer gaming. In fact, game competitions have become much more than common leisure activities and professional players earn money for attending tournaments similarly to professional athletes. A new industry was founded around game competition which is commonly refered to as *electronic sports* (eSports) or *pro gaming*.

Similarly to traditional sports, additional activities have joined eSports domain, such as fan clubs, product advertisement (i.e., propagation of products/companies at game tournaments), or betting on game results. These activities depend on additional support from IT industry for data processing and analysis, especially providing publishing infrastructure (broadcasting the eSport events), player rating, game result prediction, and fraud detection.

In our work, we focus on analysing data from played games in order to rate the performance of individual players and teams. These data can be subsequently used for player rating, team rating, preciting game results, or even for fraud detection This paper addresses various aspects of processing the data of *Counter-Strike: Global Offensive* (CS:GO) recorded games (especially the data parsing and preprocessing) and to match these data with existing sources such as HLTV[1] player boards.

This paper is organized as follows. Section 2 explains the rules and technical details of the CS:GO game. Data parsing and preprocessing are summarized in Section 3 and Section 4 describes our player-matching technique which provides integration with HLTV data source. Section 5 summarizes our findings and concludes the paper.

2 COUNTER STRIKE: GLOBAL OFFENSIVE

The *Counter Strike: Global Offensive* (CS:GO) is a first-person shooter game where two teams of 5 players compete. The game has several scenarios, but the only one used in eSport is called *bomb defusal*. One team plays the *terrorists* who are attempting to plant a bomb at one of two possible targets (*planting sites*). The other team plays *counter-terrorists* who are trying to prevent the terrorists from planting and detonating the bomb.

The game is played on several[2] well known maps. Based on the characteristics of individual tourna-

[1] www.hltv.org

[2] In most tournaments, the map pool is limited to 7 standard maps.

Bednárek, D., Krulis, M., Yaghob, J. and Zavoral, F.
Data Preprocessing of eSport Game Records - Counter-Strike: Global Offensive.
DOI: 10.5220/0006475002690276
In *Proceedings of the 6th International Conference on Data Science, Technology and Applications (DATA 2017)*, pages 269-276
ISBN: 978-989-758-255-4

ments, the teams select one or several maps (typically 3) by a deterministic negotiation protocol. The game is played once for each map and a winner of each map is recognized. The team that wins certain amount of maps (e.g., 2 wins on 3 maps) wins the whole match. In the remaining text, we will use the term '*game*' instead of '*match*' since word '*match*' may get somewhat ambiguous when describing player matching. Each team may *buy* weapons and equipment for virtual money before each round. The virtual money are earned in the game for winning rounds and killing opponents. Furthermore, players who survive the round keep most of their equipment.

2.1 Demofiles and Data Sources

A game (one map) can be recorded into a DEM file called *demo file*. It is basically a serialization of the data transferred over the network between the server and the players. The demofile can be recorded by the server itself, but it can also be recorded by a spectator (player present in the game which is invisible to other players and cannot affect the game).

The demofile for the analysis must be provided by the tournament organized, or a spectator access must be granted to the game server. The demofile can be also processed on the fly when the game is running (i.e., read whilst it is being written) to provide real-time game analysis. A very similar data stream is provided by *GOTV* – a broadcasting channel integrated in the game, which may be enabled on the server and it broadcasts game data to subscribed spectators. However, many tournaments delay this data (e.g., by 90 seconds), so the data cannot be feed back to the players via covert channels such as phone.

There is also a huge community interested in CS:GO which manage data about players and games. Perhaps the largest site dedicated to this game is HLTV. It registers all important events and tournaments and gather results. The site also gathers recorded demofiles and provide them for download. Unfortunately, HLTV administrators have little interest in sharing the data on a large scale; hence, there is no API and all data has to be scraped from web pages. Another issue with HLTV data is the player matching – i.e., interlinking existing player profiles with players in demofiles. Despite the fact that the site has identified the players internally, there is no direct linkage between the demofiles and the web.

HLTV player matching is a special case of a more general data matching problem. Although many commercial, open-source, or research data-matching systems have been developed, such as BigMatch, D-Dupe, R RecordLinkage, and many others (Christen, 2012), none of them is able to take into account the particular needs of HLTV matching. The problem is similar to nickname identification which is addressed especially in the domain of social networks. Some of the proposed methods use supervised learning methods (Peled et al., 2013), but they cannot be used in HLTV matching due to absence of the relevant labeled training data of sufficient size.

Other class of methods use their own specific models for matching individual accounts in particular social networks. These methods compute a similarity score from profile informations (Jamjuntra et al., 2017) or combine various identity search methods exploiting distinct profile attributes to match accounts across social networks (Jain et al., 2013). All of these methods utilize additional information available in user profiles to match the accounts. To our best knowledge, none of the published methods could be applicable to HLTV matching as additional information are not available in demofiles. Therefore, we propose our own method which is described in Section 4.

3 GAME DATA PARSING

The game recordings are saved in *demo files* – a proprietary format of Valve Corp which basically capture all network traffic (Breu, 2007) between the game server and clients. A demo file is fixed to one map, so if multiple maps are played in a game, multiple demo files are required. On the other hand, it captures a period of time in a game, so the game on one map may be (and sometimes is) divided into multiple demo files. Demo file uses three levels of encoding: network packets, messages encoded using Google's Protocol Buffers (Varda, 2008), and a proprietary Valve's data compression.

All encoding levels are bitwise-oriented. Parsing one demo file must be done sequentially and only maintaining the decoding state itself is rather complicated. Furthermore, the third layer of encoding is not very well documented (as it is proprietary) and changes with new version of the game.

The CS:GO server (called Source) uses a tick time unit as a logical time for the game simulation. All client inputs, actions, and interactions with each other and world objects are resolved periodically in these ticks. Typical tick-rate for tournament servers is 128 ticks per second (one tick lasts approximately 7.8ms).

Probably due to size reasons, the demo file stores information from 8 subsequent ticks together in a single *burst*. This burst has two fixed parts – list of *events*, and list of *delta changes*. Both parts are quite important, so we describe them in more detail.

3.1 Events and Delta Changes

Events register important player actions and interactions with the simulated world – for instance, when player fires a weapon, bomb is planted, or the round concludes. Events are structures that carry all event-related data, such as location of the event, player who caused the event etc. These structures are typically simple to parse, thus events are logical choice for basic data analysis of the game. On the other hand, we have discovered that events are not completely reliable and they differ significantly across the game versions. Some important events (e.g., player deaths) are sometimes missing or contain invalid information.

The second part of each burst are the delta changes. Each entity in the game is represented by its own structure (e.g., a player has a structure which contains coordinates on the map, pitch, health, etc.). Delta changes basically forms an update transaction of these structures – i.e., a list of game objects and their properties which should be inserted, updated, or removed from the game.

Delta changes are much more complex to process as they do not carry a complete information, but only a change from the previous state. Therefore, to process the state of the game completely, we have developed a simplified game simulator which holds all game objects and apply the delta change lists on them. This way we can determine the complete state of the game at any time.

3.2 Data Quality

There are several important issues we have observed when processing demo files. Perhaps the most important issue is the fact that the demo files may be corrupted. Unfortunately, the lack of documentation prevent us to detect whether this is actually a problem of file corruption on the data level, a matter of protocol errors or old protocol constructions, or simply an insufficient knowledge of the format. In order to deal with this problem, we have defined a *trustworthiness* measure which is assigned to each parsed demo file. It is basically a maximum of parsing error severity levels from the file processing. Based on trustworthiness, the files were divided into three categories: files without errors, files which were parsed correctly but contain unexpected data values (e.g., players without identifiers), and files which cannot be parsed correctly. The first category can be fully processed while the last one cannot be processed at all. The files with unexpected values can be still partially processed for some statistical purposes (e.g., when computing global precision of weapon fire).

Even if the file can be parsed correctly, the fact that the data are aggregated into bursts which represent 8 subsequent ticks may still cause minor processing issue. For instance, if a player is spawned in tick T_1 and immediately shoots in a tick T_2 whilst both ticks T_1 and T_2 are in the same burst, it might happen that the information about weapon fire would preceded the information about player appearance. Sequential parser would fail in such case as it would encounter a weapon fire caused by nonexisting player. Fortunately, this seeming violation of causality can be easily rectified by deferring processing of events and delta changes that involve nonexisting objects after the entire burst is processed.

The whole demo file contains all data from the period of time when the recording was enabled (similarly to a camera). Typically, such recording is initiated well before the actual game is started. Therefore, it also contains warmup rounds and sessions when the players are connecting and waiting. These parts are often very disruptive for data processing as they contain similar events and delta changes as a regular game, but they should not be used for data analysis of the players' performance. Furthermore, in these time periods, the players typically connect and disconnect which makes it more difficult to determine, which players are actually participating in the game.

Fortunately, a reset is typically performed just before the actual game begins. In some cases, the game is reset multiple times in one demo file. It is imperative to found the last reset just before the game starts and then process only the data after this reset.

Finally, we have mentioned that the game events are not reliable, especially when dealing with older versions of demo files. On the other hand, delta changes are more tedious for processing. In order to maintain the simplicity of events but provide better reliability, we have place detectors on certain game object properties, and when these properties are changed, we generate our own events. Our simulated events mimic the structures of the parsed game events to simplify their subsequent processing.

4 HLTV INTEGRATION AND PLAYER MATCHING

Our research included an interesting case of data integration – our data originated from the following sources:

The database of Steam users, maintained by the Valve Corporation. Although there is a public API for this database, many players have set their profiles to private and, therefore, we did not use this data di-

rectly. Nevertheless, the database was manifested in our data indirectly, since the game recordings contained Steam user identifiers for all players and observers. Since the Steam system associates the user identifiers with valuable assets like credit cards and purchased software, it is reasonable to assume that the mapping of Steam user identifiers to physical persons is sufficiently reliable.

The game recordings (demo files), created by the organizers of the tournaments. The demo files are created by the game server or by an on-site spectator client; although they are not protected by cryptographic means, their alteration would require considerable effort – thus, their contents may be considered sufficiently reliable. However, there is no 1:1 correspondence between demo files and games; therefore, some games may be covered incompletely and some demo files may contain more recordings than the game itself (e.g., a warmup phase).

The HLTV database, created by the community of tournament organizers, players, and fans. Due to the community origin, the reliability of data is variable, depending on the author of particular record and the amount of effort invested, leaving aside the possibility of intentionally entered false data. Links to the demo files are part of the HLTV database; therefore, the relation between demo files and games is unreliable as well, containing frequent omissions and a number of mismatched entries.

In the dataset extracted from the HLTV website in November 2016, there were 18 513 game entries; however, usable game recordings were available for only 8474 of them. There were 6566 HLTV player entries, but only 4888 of them were attached to at least one usable game entry. The 8474 recordings contained 6563 different Steam user identifiers.

4.1 Steam Users and HLTV Player Entries

The main problem in our case of data integration was matching player entries in game recordings to player entries in the HLTV database. For each game, there is a set of Steam users present in the recording and a set of HLTV player references entered by the community. The rules of game do not recognize particular roles of the players in the game, except for team allegiance. Consequently, there is no reliable mechanism which could pair the Steam users involved in a recording to the player entries present in the HLTV database for the same game.

In addition, while the recordings reliably identify the users (identified by their Steam ids), their role in the game, including the team allegiance is not easily recognized as users often make mistakes (or intentional disturbances) during the heat phases and observers may be present too. Figure 3 shows the histogram of the number of Steam users found in individual recordings. Ideally, the number of players should be 10; however, less than a third of all recordings satisfy this condition – there are many recordings containing more than 10 players, either due to observers or mistakenly connected users. In addition, there are also several recordings which have less than the required number of players.

On the HLTV side, there are no games with more than 10 players due to technical restrictions; on the other hand, a significant percentage of games have less than 10 players entered, either due to incompleteness of the game entry or due to presence of a player without a HLTV entry.

4.2 Names and Nicks

When Steam users connect to the game server, they may select a name which is displayed in the messages and statistics produced by the game server. Users often choose fancy names to attract attention There are some loosely followed conventions like including the team name in brackets; however, most of the names are created as a form of free art which may include a play on characters, sounds, or meanings of the name. On the other hand, there are names chosen in haste or neglect like 'asdf' or similar semi-random texts. In many cases, the same user uses different names in different games.

In the HLTV database, the visible identifier of a player is called a *nick*. Unlike in-game names, most nicks consist only of plain alphabetical words, i.e. no decorations are present. Nicks are rarely changed (and there is no record of previous nicks in the database).

Consequently, matching the Steam users to HLTV nicks requires a string-matching algorithm capable to compare the fancy in-game name to the plain nick; the system must also cope with the fact that in-game names change while the nicks are stable. It is also apparent that the string matching could not be the sole mechanism for player matching, since it is possible that in-game name would bear no similarity to the nick of the same physical person.

The name-nick matching requires a similarity measure – among the possible candidates for such a measure, the length of *Longest Common Subsequence* (Hirschberg, 1975) was selected as a trade-off between quality and implementation complexity. The LCS measure is based solely on matching characters (unlike, for instance, the Levenstein distance), i.e.

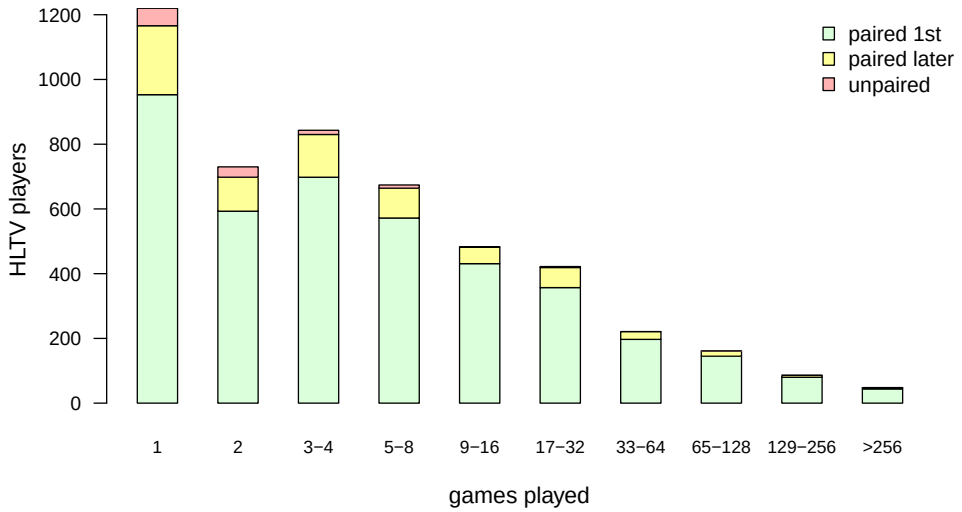

Figure 1: HLTV players distributed by the number of games played.

it does not penalize the presence of additional non-matching characters which frequently occur in fancy in-game names.

4.3 Formal Model

The available input data may be formalized as a tripartite *game-presence graph*

$$G = (V_u, V_h, V_m, E_{um}, E_{hm})$$

where V_u, V_h, and V_m are sets of vertices corresponding to Steam users, HLTV player entries, and games, respectively. $E_{um} \subseteq V_u \times V_m$ are edges representing the presence of a Steam user in a game recording, $E_{hm} \subseteq V_h \times V_m$ are edges representing the presence of an HLTV player in a game.

Names and nicks are formalized as mappings $n_u : E_{um} \to \mathcal{S}$ and $n_h : V_h \to \mathcal{S}$ where $\mathcal{S}$ denotes the domain of strings. The in-game names are associated to edges because players may select a different name in each game while the HLTV nicks are bound to vertices representing the player entries. The LCS measure is denoted as $s_{LCS} : \mathcal{S} \times \mathcal{S} \to \mathcal{N}$.

The player matching process starts with determining *candidate pairs*:

$$C = \{\langle u,h\rangle \,;\, (\exists m \in V_m)\langle u,m\rangle \in E_{um} \wedge \langle h,m\rangle \in E_{hm}\}$$

A pair of a Steam user and a HLTV player becomes a candidate if and only if there is a game where both participated. Each candidate pair receives a *weight* corresponding to the number of such games:

$$w(u,h) = |\{m \in V_m \,;\, \langle u,m\rangle \in E_{um} \wedge \langle h,m\rangle \in E_{hm}\}|$$

Then, for each candidate pair, the LCS measure is computed:

$$s(u,h) = max\{s_{LCS}(n_u(u,m), n_h(h)) \,;\, \langle u,m\rangle \in E_{um}\}$$

Since a Steam user u may use more than one name $n_u(u,m)$, all the names are compared with a HLTV player nick $n_h(h)$ and the maximum similarity is considered. Note that names which u have used in all its games are considered, including those games where the HLTV player h was not present – this arrangement improves the pairing process for users (incorrectly) represented by more than one HLTV entry.

4.4 Player-matching Algorithm

The player matching algorithm makes use of both the game-presence graph (transformed into the weights $w(u,h)$) and the name-nick similarity $s(u,h)$. The weights are given priority, i.e. the name-nick similarity is used only to break ties in the weights.

Using the game-presence graph, subsets C_u, C_h, and C_b of the candidate-pair set C are named *u-best pairs*, *h-best pairs*, and *best pairs*, respectively, and defined by the following criteria:

$$\begin{aligned}&\langle u,h\rangle \in C_u \;\Leftrightarrow\; \langle u,h\rangle \in C \;\wedge\\ &(\forall\langle u,h'\rangle \in C)\;(h' \neq h \Rightarrow (w(u,h') < w(u,h) \;\vee\\ &\qquad w(u,h') = w(u,h) \;\wedge\; s(u,h') < s(u,h)))\end{aligned}$$

$$\begin{aligned}&\langle u,h\rangle \in C_h \;\Leftrightarrow\; \langle u,h\rangle \in C \;\wedge\\ &(\forall\langle u',h\rangle \in C)\;(u' \neq u \Rightarrow (w(u',h) < w(u,h) \;\vee\\ &\qquad w(u',h) = w(u,h) \;\wedge\; s(u',h) < s(u,h)))\end{aligned}$$

$$C_b = C_u \cap C_h$$

In other words, the candidate-pair set C is ordered by the lexicographical ordering on their $\langle w(u,h), s(u,h)\rangle$ values. Then, $\langle u,h\rangle$ is a u-best pair if

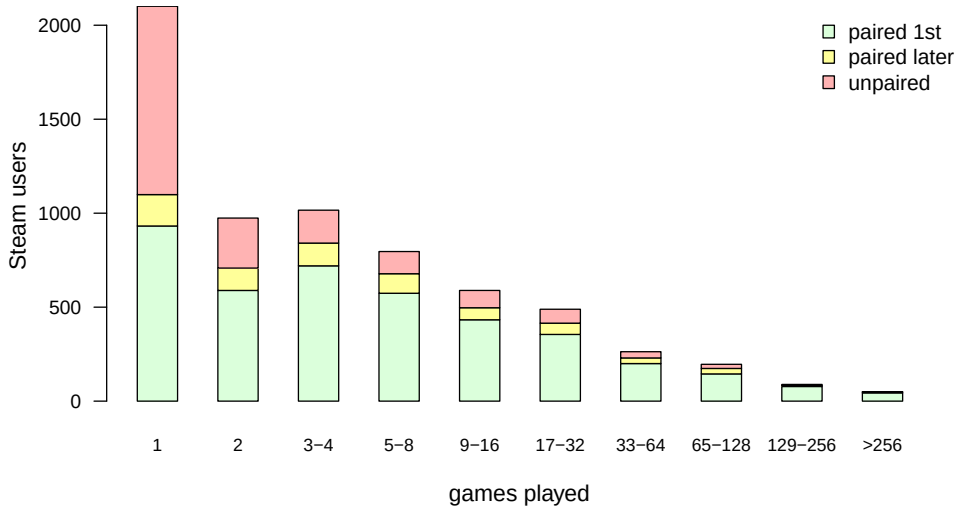

Figure 2: Steam users distributed by the number of games played.

all other $\langle u,h'\rangle$ pairs are positioned lower in the lexicographical ordering. Similarly, it is a h-best pair if all other $\langle u',h\rangle$ pairs are positioned strictly lower.

Our problem is similar to maximum weighted bipartite matching (Kuhn, 1955); however, we are interested only in matchings which are strictly better than any other. If there is a tie among several maximum matchings, our problem requires that the uncertain part of the matching be removed, i.e. not paired at all. The Hungarian algorithm for maximum weighted bipartite matching cannot reliably detect the existence of alternative matchings during its score-improving phases; therefore, it is unusable in our settings.

Note that C_u is a partial mapping from V_u to V_h and C_h^{-1} is a partial mapping from V_h to V_u. If $C_h^{-1}(C_u(u)) = u$ then $\langle u, C_u(u)\rangle$ is considered a best pair. Note that the definition of C_u and C_h implies that the values of $\langle w(u,h), s(u,h)\rangle$ must be strictly increasing along any directed acyclic path formed by C_u and C_h^{-1} edges; therefore no cycle longer than two may exist and the best pairs are the only cycles in the oriented graph formed by C_u and C_h^{-1}.

The C_b relation does not necessarily cover all vertices in V_u or V_h, leaving a *residual pair set* C_r defined as

$$\langle u,h\rangle \in C_r \;\Leftrightarrow\; (\forall h')\ \langle u,h'\rangle \notin C_b \;\wedge\; (\forall u')\ \langle u',h\rangle \notin C_b$$

The residual pair set is then used as a new candidate-pair set input for the search for best pairs and the process is repeated until no new best pairs are found. This produces a sequence of candidate-pair sets $\{C^{(i)}\}$ defined by

$$C^{(1)} = C \qquad C^{(i+1)} = C_r^{(i)}$$

Table 1: Name similarity (LCS) for 5 most-frequent players.

	h165	h161	h5386	h2553	h317
u5441	3	1	3	2	6
u5439	3	3	2	3	6
u5897	2	2	5	2	5
u5446	2	2	3	4	5
u5898	2	3	3	2	11
	3	3	5	4	11

and terminated when $C_b^{(i)} = \emptyset$.

The rationale behind the iteration is the informal definition "a pairing is best if all better candidates were already paired". Such a definition is recursive with negation – the absence of rigorous semantics for such a definition is solved by the stratification introduced by the iterative algorithm.

The sequence of best-pair sets $\{C_b^{(i)}\}$ produces the final *pairing* (by disjoint union)

$$P = \bigcup \{C_b^{(i)}\}$$

together with the following *confidence attributes* assigned to each $\langle u,h\rangle \in C_b^{(i)}$:

$$a(u,h) = \langle w(u,h), s(u,h), i\rangle$$

4.5 Example

As an illustration of the difficulty of name-nick matching, Table 1 shows the similarity between the names of five most active Steam users and the nicks of five most active HLTV players. The columns of the table correspond to HLTV player entries, ordered in the decreasing order of games played. Their nicks

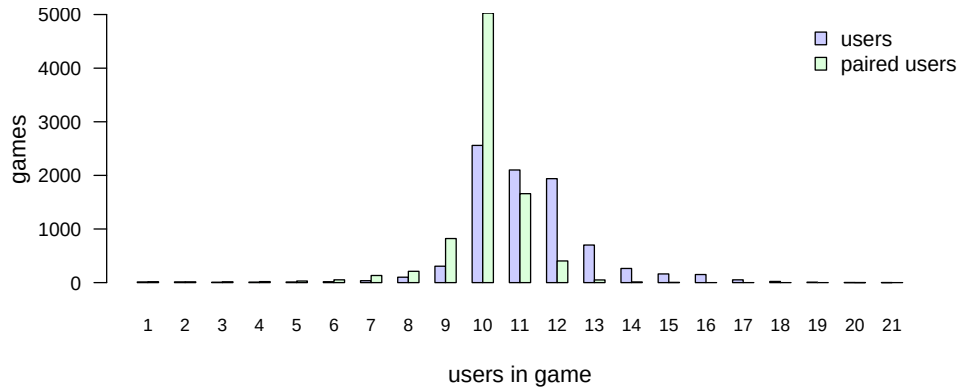

Figure 3: Superimposed histograms of users in games.

Table 2: Presence in games for 5 most-frequent players.

	h165	h161	h5386	h2553	h317	
u5441	495	490	487	478	471	499
u5439	489	493	484	475	471	494
u5897	476	475	480	461	461	481
u5446	457	455	453	459	447	462
u5898	449	451	449	444	453	453
	497	496	492	483	476	

in the HLTV databases were 3, 3, 5, 4, and 11 characters long, as shown in the bottom row of the table. The rows correspond to Steam users which may use a number of different names; therefore, the table does not show their lengths. Each cell of the table shows the similarity (longest common subsequence) between the nick and the best matching among the names of the corresponding user.

While the similarity between u5898 and h317 is prominent, the rest of the table is inconclusive. For instance, u5441 (using the name ”GO4BALKAN neo”) has the same similarity 3 to h165 (nick ”NEO”) and h5386 (”byali”) because of subsequences ”NEO” and ”BAL”, respectively. This could mean that LCS is not the ideal similarity measure here; unfortunately, this behavior is a natural consequence of the required ability to handle interleaving (like ”*N*E*O*”). In another example of false similarity, u5441 alternatively uses the name ”neoBiceps” which has similarity 6 to h317 (”pashaBiceps”), because of the inclusion of the name of clan or team.

The same five users and five players are shown in Table 2 where the cells denote the number of games where both the corresponding user and the corresponding HLTV player participated. The rightmost column shows the total number of games played by each user; the bottom row presents the total number of games played by each HLTV player.

The framed cells denote the pairing selected by our player-matching algorithm – the two thickest frames were paired in the first iteration of the algorithm, the pair u5897-h5386 was selected in the second iteration, and the two thinnest frames correspond to the third iteration. The fact that the pairs appear on the diagonal is a coincidence, but not completely random, since both the rows and the columns are ordered by the number of games played.

In the case of these five players, the pairing was based purely on the game counts. The name-nick similarity (i.e. the Table 1) is used only to break the tie when equal number of games is encountered.

4.6 Evaluation

At the input of our player matching algorithm, there were 6563 Steam users and 4888 HLTV player entries which together participated in 8474 recorded games. We worked with the idea that each HLTV player corresponds to one Steam user, i.e. we searched only for 1:1 matchings.

Our algorithm was able to find 4775 pairs – it means that 97.69% of HLTV player entries were matched to Steam users. Furthermore, the success ratio improves to 99.70% when frequent players with 10 or more games are considered. The situation is detailed in Figure 1 which shows the number of HLTV player entries depending on the number of games played. It also shows how many of them were paired in the first and in the subsequent iterations of the algorithm – the iterative character of the algorithm improved the total success ratio from 83.27% in the first iteration to the final 97.69%.

Figure 2 illustrates the situation from the other side – the Steam users. Among the 6563 Steam users, 72.76% were paired. Furthermore, 86.82% of frequent users (with 10 or more games) were paired. This corresponds to the fact that the average number of games played is 5.76 among unpaired users and 17.78 among paired users, which suggests that the un-

paired Steam users are less frequent players ignored by the HLTV database.

Finally, Figure 3 shows two superimposed histograms: The distributions of the total number of Steam users per game and of the number of paired Steam users per game. The first histogram shows that less than a third of games contain the ideal number of 10 users – majority of games include more than 10 users, probably as observers. The second histogram is restricted to paired users and shows a clear peak at 10 paired users per game, contributing to the hypothesis that most superfluous users are observers not registered in HLTV database. It also means that in 84.46% of games we can identify at least 10 users, i.e. the standard number of players.

5 CONCLUSIONS

In this paper, we have summarized our experience from preprocessing gaming data for subsequent analysis. It has been establish that selected game has a data format which is particularly difficult for processing and the parsing itself gave rise to many issues. The most important ones have been described and solution has been proposed for them.

The game recordings themselves were downloaded from the most important community portal. Unfortunately, this portal does not provide any reliable mapping between player ID from the demo files (which are actually Steam IDs) and their community profiles. We have developed a player matching algorithm which combines tripartite graph matching with string similarity measure applied on player's nick names. This algorithm was able to pair most of the players which is very important for establishing ground truth in any player rating algorithm.

In our future work, we will use the preprocessed data to perform data analysis and establish a player rating and team rating algorithm. Such rating could be used for predicting outcome of future matches (and calculating better fixed-odds for betting) or to detect possible frauds in eSport.

ACKNOWLEDGEMENTS

This work was supported by project PROGRES Q48.

REFERENCES

Breu, L. (2007). Online-games: Traffic analysis of popular game servers (counter strike: Source).

Christen, P. (2012). *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343.

Jain, P., Kumaraguru, P., and Joshi, A. (2013). @i seek 'fb.me': Identifying users across multiple online social networks. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1259–1268, New York, NY, USA. ACM.

Jamjuntra, L., Chartsuwan, P., Wonglimsamut, P., Porkaew, K., and Supasitthimethee, U. (2017). Social network user identification. In *2017 9th International Conference on Knowledge and Smart Technology (KST)*, pages 132–137.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Peled, O., Fire, M., Rokach, L., and Elovici, Y. (2013). Entity matching in online social networks. In *2013 International Conference on Social Computing*, pages 339–344.

Varda, K. (2008). Protocol buffers: Googles data interchange format. *Google Open Source Blog, Available at least as early as Jul*.