MASWS – Report 2

Razvan Ranca – 0954584

The data I used consists a history of poker matches played between bots as part of an artificial intelligence contest(http://www.computerpokercompetition.org)
One of my main goals was to transform this data into an RDF format from which exploratory data analysis could be performed with ease. Essentially I wanted a large variety of queries to be easily made on the data thus revealing various interesting properties such as if one player is more aggressive than the other, or what percentage of times a player tends to win from a certain situation.

Another important goal was to create the dataset in such a way that it could be improved and combined with any other poker datasets, even if the amount of information contained in these datasets differs from mine. One example of such a difference comes from the contrast between the type of dataset I used, which is recorded by an omniscient observer which knows every detail about the game, and the type of dataset that is recorded by one of the actual players. The latter will have less information because, in the case of a "fold", the player will never find out what the opponent's cards were.

Given these constraints I decided it would be best to avoid any 3rd party converters and instead maintain complete control over the data by writing my own script to handle the RDF conversion. (this script is made available on my homepage). Since I could find no other poker datasets, I had to engineer the format of the data from scratch, which proved to be somewhat tricky.

One particular difficulty was deciding how to represent the temporal dimension of the data. Any useful design would have to permit queries to access temporal information regarding, at the very least, the play within one hand (i.e. you should be able to ask: "if the opponent made an aggressive bet in the last round, how does it affect my odds of winning in this round?"). More complex queries would also look at the temporal ordering between hands (i.e. "If my opponent has just lost a lot of money, how will this affect his play style?"). I looked at approaches used in other datasets which incorporate temporal data, such as the ones presented in this article series: http://blog.iandavis.com/2009/08/10/representing-time-in-rdf-part-1/ , but none of them seemed to fit well to my requirements.

Keeping in line with my main goals I decided the best approach would be to design a simple solution, then try some sample queries on it and see what needs to be changed to make more interesting queries possible. After repeating this step a few times I noticed that some measure of redundancy is helpful for query design as, depending on your query's goals, the ease with which a piece of stored information can be retrieved varies greatly. Of course too much redundancy is problematic because it creates performance issues and reduces the amount of data you can effectively use. Eventually I settled on the current design, which breaks up the game in incrementally smaller chunks. I'll describe this more thoroughly when I list classes and predicates.

At this point, I felt that to get more out of the data some preprocessing would be necessary. For instance, in order to detect cases where one player made a bet in an attempt to intimidate the other player out of the hand(i.e. a bluff), it is necessary to know something about the strength of the player's hand. If the player actually has a very strong hand, then his large bet would be justified, but if he actually has a weak hand, then the bet would be an attempt to bluff the opponent. However, poker has a relatively complex ruleset involving the strength of hands. It contains rules such as: 5 cards of the same suit (a flush) are stronger than 5 cards of consecutive rank (a straight), but weaker than 4 cards of identical rank (four of a kind). This kind of knowledge of the game is necessary for more interesting queries, but encoding these rules in the query itself is infeasible. Instead I decided to encode such information directly in the data, by using the PokerEval (http://freecode.com/projects/poker-eval) library. Using this library I added data regarding the strength of each player's hand at every point in time such that direct comparison between two hands

is easy. I also added data regarding the odds of each player winning (and the odds of a tie) at any point in time, which can sometimes be a better measure of a player's strength than the pure value of his current hand(i.e. even if he currently has a weak hand, because of the available communal cards he may actually have good odds of building a strong hand by the end of play). With this new information we can design queries checking such things as whether a player made a wrong bet(i.e. he had small chances of winning, choose to put more money in the pot and subsequently lost it) and whether a player won by luck or not (i.e if before receiving the last card he had small chance of winning, but he got a good final card and snagged victory).

Regarding the URIs, I tried to use pre-existing ones whenever possible. However, since there are no other poker datasets, I had to create quite a few predicates and classes of my own. The ones I was able to find are all from dbpedia since wikipedia has a few articles discussing poker. For the nodes corresponding to a certain period of the game (such as the node for a match and the node for a betting round), I used the URL of the poker competition logs, with various hash suffixes added denoting exactly to which portion of the logs that particular node refers to. The rdf, rdfs and xsd vocabularies were also used for various predicates and datatypes.

I'll briefly describe the vocabulary terms I used.

Out of the standard vocabularies (rdf, rdfs and xsd) I used:

- **xsd:integer** & **xsd:decimal**-describe things such as the odds of winning and the pot size
- **rdf:_n** – to define what my sequence and bag classes contain
- **rdf:type** & **rdf:value –** for defining classes and the numerical value of actions or hands
- **rdfs:comment –** for strings that are only meant to aid human understanding
- **rdfs:isDefinedBy**, **rdfs:subClassOf**, **rdfs:domain** & r**dfs:range** were all used to define the custom classes and predicates which are described further below

The dbpedia resources are as follows:

- <http://dbpedia.org/resource/List_of_poker_hands#>, with the particular name of each poker hand following the hash (eg: #Straight_Flush)
- <http://dbpedia.org/resource/Glossary_of_poker_terms#>, with "act" "community_card" and "stack" following the #
- <http://dbpedia.org/resource/Pot_(poker)>, signifying the amount of money in play
- <http://dbpedia.org/resource/Suit_(cards)> and <http://dbpedia.org/resource/Rank> describing the attributes of the cards
- <http://dbpedia.org/resource/Big_blind> showing the blind and minimum bet

* An interesting point to note here is that I had some trouble with the "Pot" and "Suit" URIs because they contain parentheses. I was using "rapper" to generate graphs and XML and it seems that rapper can't handle parentheses in suffixes to URIs, so instead of saying "db:Pot_(poker)" I was instead forced to use the full <http://dbpedia.org/resource/Pot_(poker)>. Using the escape characters %28 and %29 instead of parentheses doesn't help either as rapper can't handle "%". Even worse, rdfcat apparently doesn't like parentheses at all; even with the full URIs I get an error.

The resources corresponding to a stage of play all use the competition log's URL, followed by the specifications of the competition data we're using (the logs have a couple different datasets). Since I am using the 2012, two player, no limit dataset, the base URI is therefore: <http://www.computerpokercompetition.org/downloads/competitions/2012/logs/#2012-2p-nolimit/>

This base URI is followed by:(here pred<n> means that you can have any of pred0,pred1,pred2 etc)

- **match<n>** - represents a whole match(i.e. a file in the logs)
- **match<n>_deal<m>** - represents one dealt hand in "match<n>"(i.e. a line in a file)
- **match<n>_deal<m>_p1** and **match<n>_deal<m>_p2 –** represents the first and second player to act during this dealt hand.
- **match<n>_deal<m>_p<k>_hc1** and **match<n>_deal<m>_p<k>_hc2** shows the two private cards (hole cards) of the players
- **match<n>_deal<m>_stage<k>_bet0** – this node has two different meanings, it stands both for the "k"th betting stage in "match<n>_deal<m>" AND for the first decision that needs to be taken in this betting stage. I decided to overload it in this way because there was no practical reason to create two distinct nodes for this purpose, and since every dealt hand can have up to 4 betting rounds, the number of nodes adds up quickly.
- **match<n>_deal<m>_stage<k>_bet<p>** - where p IS NOT 0. As mentioned above, when p is 0 there a double meaning to the node. When p is not 0, this is just the "p"th decision that needs to be taken during the current betting stage.
- **match<n>_deal<m>_stage<k>_bet0_ComCards** – are the community cards that have been revealed up to this point (these are further individually identified by appending CC0, CC1, etc to the above URI which gives you each card)
- finally, **match<n>_deal<m>_stage<k>_bet<p>_nextAct** – describes the action that was ultimately taken following the situation "match<n>_deal<m>_stage<k>_bet<p>", this action dictates what the next situation will be(i.e. if the player folded the dealt hand is over, if he called the betting round is over and if he raise the betting round continues)

The identities of the players (or bots in this case) are also shown relative to the log URI, just by appending the bot's name to the base URI.

The custom defined vocabulary items use a schema file on my homepage as their base URI(<http://homepages.inf.ed.ac.uk/s0954584/poker-schema.xml/#>).
These items fall into 2 categories:

First there are the classes. These are:

- **Match** – represents one of the "match<n>" instances. Is also a subclass of Seq as it holds an ordered sequence of dealt hands (Deals)
- **Deal** – class for the "match<n>_deal<m>", again subclasses Seq as it holds ordered betting rounds
- **BettingRound –** class for the "match<n>_deal<m>_stage<k>_bet0", represents an entire round of bets and also subclasses Seq as it hold ordered BettingDecisions
- **BettingDecision –** represents the actual situation an agent finds himself in when he has to decide what action to take
- **BettingAction –** represents the action that the agent eventually took when faced with a BettingDecision
- **Player –** a player at an instant of time, subclasses Bag as it holds 2 unordered cards
- **CommunityCards –** the communal cards, subclasses Seq as it has a ordered list of cards
- **Card –** a single playing card
- **Hand –** this just represents the possible poker hands (flush, straight, etc)

And secondly there are the predicates (note that almost all of them have defined domains and ranges, check http://homepages.inf.ed.ac.uk/s0954584/poker-schema.n3 for details)

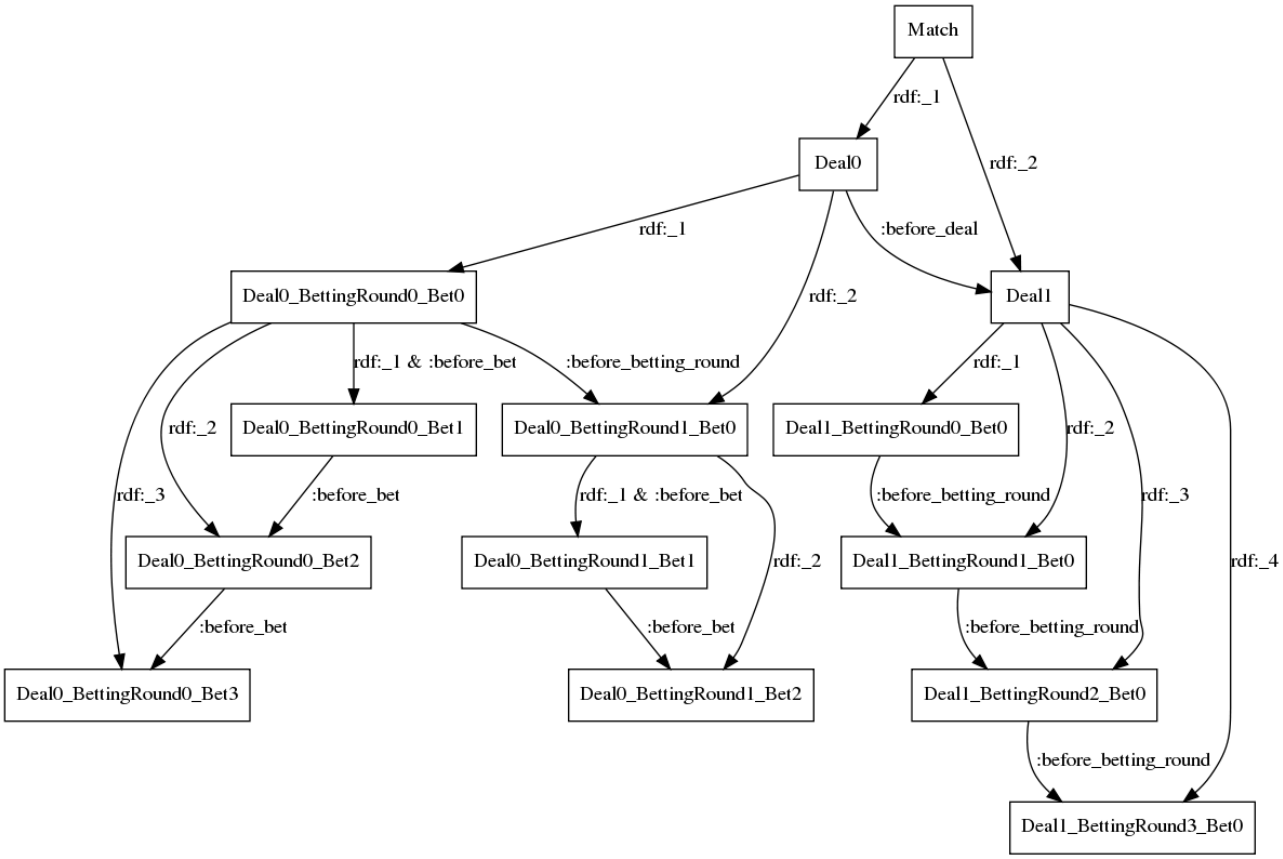
- **card_suit** and **card_rank** - show what suit and rank an individual card has
- **player1** and **player2** – indicates which player acts first and which second
- **played_by** – is the real identity of the player placeholders we use everywhere (i.e the real name of "player 1")
- **deal_no** – is the index of the current dealt hand (between 0 and no of hands played)
- **winner** – is the player who won either the current dealt hand or the overall match
- **amount_won** – as before, used to show winnings both per hand and per match
- **betting_stage** – indicates the name of the current BettingRound (as in flop, turn, etc)
- **acting_player** – indicates which player is taking a certain BettingAction
- **action_type** – shows what action is being taken in a BettingAction node(eg: raise, call)
- **bets_cur_round** – is the amount that has been wagered during the current BettingRound
- **cur_hand_P1** and **cur_hand_P2** - are the types of poker hands that player 1 and player 2 currently have (eg: One_Pair, Three_of_a_kind), these will change as new community cards are revealed
- **win_prob_p1**, **win_prob_P2** and **tie_prob** - are the calculated probabilities of a player winning or a tie occurring. These too will change as the community cards are revealed and at the end of the dealt hand(on the river) one of them will be 1 and the others 0
- **before_bet** – is the first temporal predicate, it is used between consecutive BettingDecision, showing their chronological order
- **before_betting_round** – is the second temporal predicate, and it is used between consecutive BettingRounds
- finally, **before_deal** – is the third temporal predicate and is used to show the chronological order between consecutive dealt hands (i.e. Deals)

The schema file also contains two datatype definitions on the Rank and Suite URI's mentioned under dbpedia above (http://dbpedia.org/resource/Suit_(cards)) and http://dbpedia.org/resource/Rank). I chose this option so that I could specify that the strings used to represent suits and ranks have a type without having to create further nodes for that sole purpose.

I can now clarify on the treatment of time that I mentioned before. Under this model, there are 3 different granularities of time that we keep track of. First there is the ordering of whole dealt hands, specified by the predicate "before_deal". At a lower granularity, within one hand, there is the ordering of betting rounds, specified by the predicate "before_betting_round". It is worth noting that no matter how many times we follow this predicate we will always remain within the same dealt hand. And finally, at the lowest granularity, there is the ordering of betting decisions, accomplished by the predicate "before_bet". It is again worth noting that no matter how many "before_bet"s we follow (or indeed if we take the transitive closure over "before_bet"), we will never leave the current BettingRound. However, as I mentioned in the class description, these three predicates are not the only means by which time is kept track of. In addition, every more general object contains an ordered sequence of all its more specific counterparts. So "match0" has an "rdf:_1" predicate leading to "deal0" and an "rdf:_2" predicate leading to "deal1" and so on. Going down the granularity levels again, "deal0" will also have specific ordered predicates leading to "deal0_stage0_bet0" and to "deal0_stage1_bet0". And finally "deal0_stage0_bet0" will itself have ordered predicated leading to " deal0_stage0_bet1" and " deal0_stage0_bet2" etc.

This is one of the areas where, as I mentioned at the beginning, some redundancy seems helpful. By having both these temporal addressing methods we, in a sense, get both the behaviour of an array(the direct addressing via "rdf:_2") and of a linked list (the relative addressing via "prev :before_bet cur"). Having both these options makes the job of query creation significantly easier, especially seeing as many queries will need to make use of temporal information of some kind.

These temporal addressing methods are illustrated in the following graph(also available at: http://homepages.inf.ed.ac.uk/s0954584/graph_temp.png). Note that the children of Deal1_BettingRoun0_Bet0, Deal1_BettingRoun0_Bet1, Deal1_BettingRoun0_Bet2 and Deal1_BettingRoun3_Bet0 are missing for simplicity since they would be analogous to those observed in the other side of the graph for Deal0.



Several files are made available on my webpage:

- http://homepages.inf.ed.ac.uk/s0954584/graph_temp.png – the above graph
- http://homepages.inf.ed.ac.uk/s0954584/graph_1Hand.png – graph of RDF data for 1 dealt hand
- http://homepages.inf.ed.ac.uk/s0954584/graph_schema.png – graph of custom defined classes and predicates
- http://homepages.inf.ed.ac.uk/s0954584/poker_schema.n3 – definition of custom classes and predicates (view source for good formatting)
- http://homepages.inf.ed.ac.uk/s0954584/poker_schema.xml – same as the above, except converted to xml
- http://homepages.inf.ed.ac.uk/s0954584/1Hand.n3 – RDF data for 1 hand
- http://homepages.inf.ed.ac.uk/s0954584/1Match.n3 – RDF data for 1 full match