ProbSy - A System for the Calculation of Probabilities in the Card Game Bridge

Anders L. Madsen
Aalborg University
Department of Computer Science
Frederik Bajers Vej 7E
DK-9220 Aalborg Ø
Denmark
anders@cs.auc.dk

Lars M. Nielsen
Hugin Expert A/S
Niels Jernes Vej 10
Box 8201
DK-9220 Aalborg Ø
Denmark
ln@hugin.dk

Finn V. Jensen
Aalborg University
Department of Computer Science
Frederik Bajers Vej 7E
DK-9220 Aalborg Ø
Denmark
fvj@cs.auc.dk

Abstract

We present an application designated to the calculation of probabilities in the card game bridge. The application can be used by bridge players to improve their decisions in the game. We describe in detail how a Bayesian network is dynamically generated depending on the case presented to the application. We also present a communication language which makes the specification of all kinds of relevant evidence simple. The communication language also makes it easy to specify requests for probabilities of complex properties of the distribution of cards. We explain how the probability of a strategy for declarer's play being successful is calculated by transforming the corresponding condition to disjunctive normal from. Finally, we give examples showing the use of PROBSY.

Introduction

Bridge is a card game played by four players which are initially dealt 13 cards each. Each player is usually associated with one of the four directions of the world and they are organized in two pairs such that *North* and *South* are partners and similarly are *East* and *West*. The game consists of two phases, an auction and the play of the cards. During the auction each partnership bids to get a contract. The pair who makes the highest bid gets a contract to win a certain number of tricks. The two players who get the contract are called declarer and dummy while the players of the other pair are called the defenders (Francis, Truscott, & Francis 1994). Throughout this paper we assume *South* to be declarer.

After the auction declarer's left hand opponent makes a lead and dummy places her cards face up on the table and she no longer participates in the game. Now, only two hands are unknown to the three players still remaining in the game. Based on information revealed during the auction, each player has an opinion about the hands he or she cannot see. That is, each player is uncertain about the distribution of the hidden cards, because each player has imperfect information about the state of the game (Blair, Mutchler, & Liu 1993). After the lead is made, declarer (the decision maker) should decide on a strategy (also referred to as a plan) for playing the two hands he controls.

Many bridge players use much time to perform after-game analysis. In particular, declarer's play is often heavily discussed. Such discussions are often focused on the probability of the cards held by the defenders being distributed in a certain way. Therefore, many bridge players could benefit from a tool which can be used to perform after-game analysis. PROBSY (short for probability calculation system) is such a tool.

PROBSY is primarily a tool for calculating the probability of a strategy being successful given some knowledge about the distribution of cards. A strategy is successful if declarer wins at least the number of tricks bid during the auction. Given a strategy declarer makes his contract if the cards are distributed in a certain way.

In the rest of the paper we describe how a Bayesian network is dynamically generated each time the user wants to calculate probabilities of properties of the distribution of the defenders cards. We also present a language to communicate with the inference engine. Using the communication language it is possible to specify properties of the distribution known to the decision maker and conditions under which a strategy will succeed. Next, we describe how probabilities of conditions specified in the communication language is calculated. Two examples are used to show what kinds of decisions PROBSY can be used to support. Finally, we compare PROBSY with another system and provide a discussion.

Bayesian Networks

A Bayesian network is an acyclic graph whose nodes represent random variables and whose edges represent causal dependencies between these variables. A Bayesian network is a well suited representation of a joint probability distribution over a set of variables. The representation can be simplified by exploiting causal independences among the variables.

We use Bayesian networks to represent the joint probability distribution of the cards initially distributed between the defenders only. It is not necessary to represent the cards initially held by declarer and dummy, because as declarer we are only interested in properties of the cards distributed between the defenders.

Each of the cards known to be held by one of the defenders initially are represented by a random variable with two states, one for each of the defenders (*East* and *West*). As we do not represent the cards held by declarer and dummy in the network, PROBSY has to dynamically generate a new Bayesian network each time the user wants to calculate the probability of a set of conditions.

Figure 1 shows a simplified version of a Bayesian network generated by PROBSY. The network is simplified compared to the actual network generated in the sense that no intermediate nodes are included in the figure. Only the nodes necessary to get an impression of how the model is build are shown.

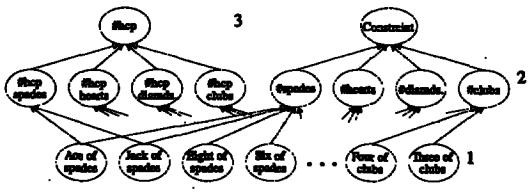


Figure 1: An example of a simplified version of a Bayesian network generated by PROBSY.

The figure contains three different blocks of nodes. Block 1 contains 26 nodes. Each node in this block represents one of the cards held by the defenders. We call the nodes in block 1 the *card nodes*.

Block 2 contains four nodes. These nodes are used to represent the number of cards initially held by *West* in each of the suits. If there are n card nodes in block 1 of a particular suit, the corresponding node in block 2 has $n+1$ states labelled $0, 1, \ldots, n$. Note that when we know the number of cards held by *West* it is easy to calculate the number of cards held by *East* and vice versa. We refer to the nodes in block 2 as the *suit counting nodes*.

All card nodes in block 1 representing cards in the same suit are parents of the corresponding suit counting node. This makes it easy to determine the state of the suit counting node given a configuration of the parent nodes. Given a configuration of the parents, the state of a suit counting node reflects the number of cards held by *West* in the parent configuration.

Block 3 contains five nodes. Each of the four bottommost nodes represent the number of high card points (Aces(4), Kings(3), Queens(2), and Jacks(1)) initially held by *West* in the corresponding suit. These four nodes are the parents of the fifth node, and they are referred to as the *suit hcp counting nodes*. The fifth node represents the total number of high card points initially held by *West* and is referred to as the *hcp counting node*. The five nodes have states reflecting the possible number of high card points held by *West* in any suit and totally. Again, if we know the number of high points held by *West* either in a certain suit or totally, then it is easy to calculate the number of high card points held by *East* and vice versa. The state of the fifth node is determined in a fashion similar to how the state of a suit counting node is determined.

Finally, the node labelled *Constraint* is not part of any block. This node is used to insure that the probabilities calculated are conditioned on the fact that each player initially holds 13 cards. The constraint node has two states *yes* and *no*. The parent nodes of the node are the suit counting nodes. For each configuration of the parent nodes the state of the constraint node is *yes*, if the configuration reflects a situation where *West* initially holds 13 cards. Otherwise the constraint node is in state *no*. This makes it easy to condition on the fact that each player initially holds 13 cards: select state *yes* in the constraint node, see (Jensen 1996) for details on the technique of using constraints in Bayesian networks. Please note that the card nodes are the only nodes without parents. The prior probability distribution of a card node is uniform, and all other probability distributions are logical in the sense that the state of a node is deterministically determined by the configuration of the parent set.

To support the use of Bayesian networks we use HUGIN. HUGIN performs the propagation of evidence in a secondary structure known as a junction tree, see (Jensen 1996) for a description of junction trees. We encountered practical problems related to the size of the junction tree generated by the HUGIN inference engine. The problem is that the size of the cliques in the junction tree grows exponential in the number of parents of each node in the Bayesian network. One solution to this problem is to use modelling techniques as described in (Jensen 1996).

A number of intermediate nodes is added to the network to minimize the amount of space needed to store the joint probability distribution and to ease the specification of certain kinds of evidence. The connections added are all logical, so the conditional probability tables are easy to specify. We have in particular used the technique called *divorcing*. Assume a node B has a set of parents $A_1, \ldots, A_n$. Then $A_1, \ldots, A_i$ is divorced from $A_{i+1}, \ldots, A_n$ by introducing a new mediating node C with $A_1, \ldots, A_i$ as parents and B as a child. The assumption behind this technique is that the set of configurations of $A_1, \ldots, A_i$ can be partitioned into sets $c_1, \ldots, c_k$ such that whenever two configurations $a_1, \ldots, a_i$ and $a'_1, \ldots, a'_i$ are elements of the same set c_i, then $P(B \mid a_1, \ldots, a_i) = P(B \mid a'_1, \ldots, a'_i)$. The divorcing variable C has states $c_1, \ldots, c_k$, see (Jensen 1996) for further details.

Figure 2 shows in more detail how divorcing is used in the Bayesian networks dynamically generated.

The Language

To ease the specification of information known to the decision maker (declarer) and the requests for probabilities of various properties of the distribution of defenders cards, a communication language has been developed.

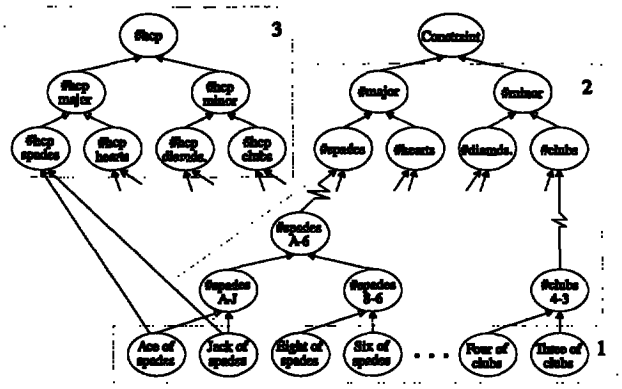


Figure 2: An example of a Bayesian network generated by PROBSY.

The language is based on a small set of operators with two important properties. First, the operators make it possible to specify all kinds of information the decision maker might have about the distribution of the defenders cards. Second, all kinds of properties of the distribution are easily specified using the operators of the language. PROBSY is not intelligent, so the user has to extract and specify all the information she might have about the distribution of the defenders cards. This information can either be extracted from the auction or from previously played tricks. Furthermore, if the user wants to know the probability of certain properties of the distribution, then she has to specify these properties as a condition herself.

The language contains five fundamental operators. These operators can be used to indicate the number of cards a certain player holds in a certain suit, to indicate the number of high card points a certain player holds, and to indicate the location of a certain card. Because the language basically only consists of three simple operators the more complex the evidence and the properties are, the more complex conditions have to be specified as input. The precision of the calculated probabilities depends on how careful the user is when specifying the evidence and the properties. Below we explain the special operators of the language. In the descriptions <suit> indicates one of the four suits (♣, ♢, ♡, ♠) and <direction> indicates one of the two defenders (*West* or *East*). The operators have the following syntactical form:

<suit> (<direction>) eg. diamonds(West)=3. This operator refers to the number of cards initially held by the player indicated (<direction>) in the suit indicated (<suit>).

hcp (<direction>) eg. hcp(West)=12. This operator refers to the total number of high card points initially held by the player indicated (<direction>).

hcp(<direction>, <suit>) eg. hcp(West, spades) in [1,3]. This operator refers to the number of high card points initially held by the player indicated (<direction>) in the suit indicated (<suit>).

hand(<card>)=<direction> eg. hand(dA)=West. This operator can be used to specify that the player indicated (<direction>) initially held the card indicated (<card>).

hand(<card$_1$>, <card$_2$>)=<direction> eg. hand(sK, sQ)=West. This operator can be used to specify that the player indicated (<direction>) played the first card (<card$_1$>), when he just as well could have played the second (<card$_2$>), if he held both cards. That is, this operator is used to handle situations where one of the defenders might have had a restricted choice.

The first four operators are pointers to specific sets of states in specific variables of the Bayesian networks. The last operator shown is the only operator which required addition of nodes to the Bayesian network generated. When this operator is used, the nodes corresponding to the two cards indicated gets a common child with two states: one to indicate that <direction> played the first card and one to indicate that she played the second card. Now choosing the state corresponding to the card actually played makes the two card nodes d-connected. This new node is used to handle situations where the rule of restricted choice applies. This topic is elaborated further later on.

The communication language is based on the set of Boolean combinations of the operators. The rules of precedence are such that parentheses are calculated first following in order by negations, conjunctions and disjunctions. This implies that $a \vee b \wedge c$ is short for $a \vee (b \wedge c)$.

To ease the specification of complex conditions it is possible to define macros. Macros make it possible to build more complex conditions in a bottom-up fashion. Consider the two examples below showing how macros are defined:

```
finesse_sp=hand(sJ)=West or
    hand(sJ)=East and spades(East)=1;

1of2_finesses_in_cl=hand(cJ)=East or hand(cK)=East;
```

First we define the macro *finesse_sp*[1] to equal a condition specifying that *West* holds ♠J or *East* holds ♠J and precisely one spade. The second macro (*1of2_finesses_in_cl*) is defined to equal a condition specifying that *East* holds ♣J or ♣K.

To make PROBSY calculate the probability of properties of the distribution of the defenders cards a string and a condition corresponding to the properties has to be specified. The string is used when the results of the computations are displayed.

The following example shows how to make PROBSY calculate the probability of a condition build as a conjunction of the macros defined above.

[1] A finesse in bridge is defined as the attempt to win a trick with a lower ranking card by taking advantage of a favorable position of higher ranking cards held by the defenders (Francis, Truscott, & Francis 1994).

```
"Plan 1 : " : finesse_sp and 1of2_finesses_in_cl
```

Probability Calculation

Statements in the communication language can refer to suits, cards, and high card points. It is easy to deal with statements only involving a single variable as its probability distribution is available when the network is initialized. It gets a little more complicated when dealing with composite conditions (conjunctions and disjunctions of literals[2] or other composite conditions). Composite conditions consisting of conjunctions only can be entered as evidence in the Bayesian network by entering the literals one by one. The probability of the condition is equal to the probability of the evidence entered. The probability of the evidence entered is achieved as a side effect of propagation. That is, Bayesian networks support calculation of the probability of a composite condition consisting of conjunctions only very well. The probability of a composite condition including disjunctions cannot be calculated as easily.

To calculate the probability of a composite condition we transform it to disjunctive normal form. A condition C is in disjunctive normal form if: $C = \bigvee_{i=1}^{n}(\bigwedge_{j=1}^{m_i} l_{ij})$, where the l_{ij}'s are literals. Any condition can be transformed to an equivalent condition in disjunctive normal form, where two conditions C_1 and C_2 are equivalent if a model for C_1 is also a model for C_2 and vice versa.

To calculate the probability of an arbitrary condition in disjunctive normal form we use the following fact:

$$P(\bigvee_{i=1}^{n}(\bigwedge_{j=1}^{m_i} l_{ij})) = \sum_{\mathcal{X}\subseteq\{x\in\mathbb{Z}|1\leq x\leq n\}} (-1)^{(|\mathcal{X}|+1)} P(\bigwedge_{i\in\mathcal{X}}(\bigwedge_{j=1}^{m_i} l_{ij})),$$

where the l_{ij}'s are literals.

When faced with a composite condition including disjunctions PROBSY transforms it to an equivalent condition in disjunctive normal form and calculates the probability of the equivalent condition using the probabilities of conjunctions only. The transformations are performed using a set of rules based on the DeMorgan laws and the distributive laws. The time complexity of the transformations is exponential in the number of disjunctions.

An alternative approach to the calculation of the probability of an arbitrary logical formula is to create a node (or a set of nodes) in the network representing the formula. This will eliminate the need to transform the formula into disjunctive normal form, but it will increase the complexity of the junction tree constructed from the Bayesian network considerably as new dependencies are introduced. If more than one statement is specified on input, then either we get a larger more complicated Bayesian network and therefore a larger maybe intractable junction tree or we have to construct a new Bayesian network and therefore a new junction tree for each statement. With the approach we have taken it is possible to use the same junction tree to calculate the probability of all the statements specified on input.

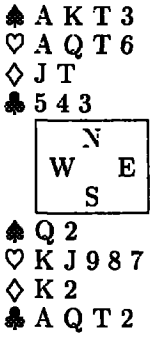
Each time PROBSY has to calculate the probability of a conjunction it has to perform a propagation. To reduce the number of propagations performed we compare the number of propagations needed to calculate the probability of a condition C with the number of propagations needed to calculate probability of $\neg C$. We choose to calculate the probability of the condition requiring the fewest number of propagations.

Another practical opportunity to speedup the calculation of a composite condition in disjunctive normal form is to exploit that a conjunction might appear more than once in the composite condition. Furthermore, one conjunction might be implied by one of the other conjunctions in the composite condition. We have not in the current implementation of PROBSY exploited these opportunities for speedup.

Examples

We now give two examples showing what kinds of decisions PROBSY can be used to support.

Consider the distribution of cards shown below:



We are as *South* declarer of a contract to make 12 tricks with hearts as trumps. The lead made by *West* is a heart. After playing an additional round of trumps where everyone follows suit we have to decide between two different strategies. The first strategy is based on a successful finesse in spades through *West* while the second is based on playing for a drop of $\spadesuit J$. If we succeed in establishing four tricks in spades, then the two diamonds held by declarer can be discarded on the third and fourth round of spades. This implies that we only need one out of two finesses in clubs to make the contract. On the other hand if $\spadesuit J$ does not drop, then we can make the contract if both finesses in clubs are successful.

Below we show how the conditions under which the first mentioned strategy is successful can be specified:

```
"Plan 1 : " : finesse_sp and 1of2_finesses_in_cl
```

The second plan can be specified in a similar way.

PROBSY can also handle situations where the rule of restricted choice is involved (Francis, Truscott, &

[2]We refer to conditions involving only one variable as *literals*. We also refer to negations of literals as literals.

Francis 1994). The issue is that the play of a card which may have been selected as a choice of equal plays increases the probability of the player initially holding a card combination in which his choice was restricted. We use the following example to make things clear. Assume that we as declarer need one trick in the suit shown below to make the contract:

Let us also assume that we have enough entries to reach the hand held by declarer to lead a small spade toward dummy twice. On the first round of spades we lead ♠2 and *West* plays low (♠6), dummy the queen, and *East* wins the trick with the ace. Now we have to decide between the nine and the jack when the second round of spades is played from declarer and *West* once more plays low (♠8). The decision is easily made using PROBSY. Enter the evidence (the cards played by the defenders) as a condition and specify two strategies corresponding to playing ♠9 and playing ♠*J* on the second round of spades, respectively. The evidence can be specified as a condition in the following way:

```
hand(sK,sA)=East and hand(s6)=West and hand(s8)=West
```

The two strategies can be specified as:

```
"Play the jack : " : hand(sA)=West;
"Play the nine : " : hand(sT)=West
```

The result of the computations shows that it is better to play ♠*J* (0.65) on the second round of spades than to play ♠9 (0.47). The number 0.65 indicates the probability of winning one trick in the spade suit given declarer follows the first plan. That is, the probability of *West* holding ♠*K* is 65%. Similarly for the second plan. If we perform the calculations without noticing that the rule of restricted choice applies, the probabilities are 48% and 48%, respectively.

Comparison

There exists other systems which can be used to decide between different strategies. Hans van Staveren has implemented a system named DEALER(Staveren 1993) which among other things can be used to calculate probabilities of properties of the distribution. DEALER establishes the probabilities using sampling. The system generates a number of distributions and counts the number of distributions fulfilling the properties specified by the user. This implies that the probabilities calculated depends heavily on the number of distributions generated and especially on the number of distributions fulfilling the properties. The probabilities calculated by PROBSY on the other hand are the correct probabilities. Furthermore, when calculating the probability of very complex properties or properties with a low probability of being satisfied DEALER can be very slow compared to PROBSY. Furthermore, by means of sampling only, it is not easy to handle situations where the rule of restricted choice applies.

The time complexity of PROBSY only depends on the number of disjunctions in the transformed condition, not on the number of conjunctions. Whereas the time complexity of DEALER depends on both the number of conjunctions and the number of disjunctions in the original condition.

On the other hand DEALER has some functionality which is not supported by PROBSY. This is mainly the ability to sample. PROBSY can however with some care be extended to support sampling, because this feature is supported by the HUGIN Application Program Interface. To extend PROBSY with facilities supporting sampling we have to handle evidence expressed as a condition including disjunctions.

Conclusion

PROBSY is well suited for bridge players to use for after-game analysis. This is due to two reasons. The first reason is that the system calculates the probability of a condition very fast - usually less than a second on a PC with an Intel Pentium processor. The easy specification of the input is the second reason. The user interface eases the specification of the cards controlled by declarer and the communication language makes it easy to specify the evidence and the conditions.

The system can be extended with features such as sampling, the possibility to specify partnership agreements, and a better support for the specification of complex evidence.

The PROBSY application and a manual which in detail describes how to use the system can be found at the following URL:

```
http://www.hugin.dk/probsy/
```

References

Blair, J. R. S.; Mutchler, D.; and Liu, C. 1993. Games with imperfect information. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning*, 59–67.

Francis, H.; Truscott, A.; and Francis, D. 1994. *The Official Encyclopedia of Bridge.* American Contract Bridge League, Inc., 5th edition.

Jensen, F. V. 1996. *An Introduction to Bayesian Networks.* UCL Press, London.

Staveren, H. v. 1993. Dealer - a bridge hand generator program. http://rgb.anu.edu.au/Bridge/Programmes/Dealer-Staveren/.