Table 1: Tournament results between players, all performed on the same set of 2000 deals. In each match, the column player played against two clones of the row player. We report the average score-per-list difference (row-column) and score-per-list (standard deviation in brackets) for row and column players in each match-up.

row \ col	Kermit			Tracker			Tracker-rm36			theCount			theCount-rm36		
	r-c	row	col	r-c	row	col	r-c	row	col	r-c	row	col	r-c	row	col
Mistake0.03	379	874(46)	495(55)	344	812(47)	468(55)	262	756(48)	494(50)	108	756(48)	648(48)	133	761(48)	628(46)
Mistake0.10	-213	634(48)	847(48)	-90	693(48)	783(52)	47	684(48)	637(53)	-161	660(48)	821(48)	-144	662(49)	806(48)
Mistake0.25	-638	460(52)	1098(45)	-726	460(50)	1186(48)	-498	468(49)	966(56)	-618	511(50)	1129(52)	-564	493(52)	1057(53)
XSkat	-741	648(42)	1389(47)	-826	612(36)	1438(39)	-662	594(31)	1256(63)	-818	630(39)	1448(52)	-808	623(39)	1431(39)

between computer players. Our goal is to compare the performance of three different *candidate* players: Kermit, the Count and a player we call Tracker. Kermit is the static, non-adaptive skat player described in [Buro *et al.*, 2009]. The Count is a modification of Kermit that uses PIPMA to adjust its bidding behaviour, as described in Section 3. Tracker is a "naive" adaptive player also based on Kermit that tracks only its own performance as soloist in the context of its current opponent match-up: after each soloist game played, Tracker updates its perceived error in its bidding evaluation function according to Equation (1) from Section 3. It then uses the average accumulated error as an offset to its bidding threshold in future games. Tracker's final adjusted bidding threshold is capped at 0.75, so as to prevent the program from believing that all soloist games are unwinnable (and therefore never playing as soloist again to gather more data) after a particularly unlucky streak.

We note that the card-play strength of our three candidate players is identical. Furthermore, the adaptive bidding of Tracker and the Count is such that against an opponent of equal card-play strength, the bidding of these two programs is nearly identical to Kermit's. Therefore, comparing these programs by playing them against each other would be futile.

Instead, we play each of these candidates against a set of *benchmark players*. Three of these are synthetic players, as described in Section 3, using mistake rates of 0.03, 0.1 and 0.25. The fourth benchmark player is a rule-based program called XSkat, which is free software written by Gunter Gerhardt. XSkat has previously been shown to be substantially weaker than Kermit [Buro *et al.*, 2009], but serves as a useful example of a player that is very different from the synthetic players and our candidate players.

In our experiments, each candidate played the same set of 2000 skat deals against two clones of each of the benchmark players (although the candidate players were not aware that their two opponents were identical for the purpose of modelling). All experiments were run on computers with eight 2.5 GHz Intel Xeon E5420 CPUs with 16 GB of RAM.

We show results of these tournaments in Table 1. The displayed score represents average points earned per 36 games, which is the length of a standard match (or "list") in most skat clubs and tournaments. The candidate players Tracker and the Count have two types of entries in this table: one with the suffix "rm 36" and one without. The "rm 36" version of each program deletes all of its opponent modelling data every 36 games; the non-suffixed version keeps its data for the duration of the 2000 game run.

Discussion

In opponent modelling, the general goal is to improve performance against some opponents without losing in self-play or becoming vulnerable to exploitation by an adaptive adversary. Our experiments against the candidate players were designed to test the first two of these three criteria and we will discuss the results in that light.

Against the Mistake0.1 player, we see little difference between Kermit and the Count; in fact, Kermit performs slightly better. This is not unexpected, because Kermit's own global error rate is very close to 0.1, and therefore this result approximates a 'self-play' match. Since Kermit is tuned to perform well in self-play, the best the Count could hope for in this match-up is to tie Kermit's performance, which it effectively does.

Meanwhile, the Count achieves a modest but, in skat terms, still substantial gain against XSkat, and a dramatic gain against the exceptionally strong Mistake0.03 player. The Count's somewhat weak performance against the Mistake0.25 player was more of a surprise. Although the Count still outperforms Kermit in the long run, this advantage is slight enough that it disappears when the Count is forced to erase its modelling data. This appears to be because although the synthetic player's cardplay is quite weak, it still bids as aggressively as Kermit, meaning that Kermit earns many points on defense and would achieve little by increasing its own bidding aggressiveness (as the Count tries to do). XSkat, while of similar card-play strength to the Mistake0.25 player, is more conservative in its bidding. A follow-up experiment where the Mistake0.25 player's bidding threshold was increased from 0.6 to 0.7 seemed to confirm the role of the bidding behaviour; in this setting, the Count (with "rm36" enabled) exceeds Kermit's list-average by 57 points.

Finally, we compare the Count to the Tracker player. Our objective in including Tracker among our candidate players was to compare the amount of data needed to perform successful opponent modelling by a naive approach against the amount needed by the Count's PIPMA. In nearly all cases, we see that Tracker performs quite similarly to the Count in the long run. However, when it is forced to delete its modelling data every 36 games, the performance of Tracker degrades quickly, becoming weaker than both Kermit and the Count against all the benchmark players except Mistake0.03, where it is roughly even with Kermit. The Count, on the other hand, becomes only very slightly weaker when it deletes its data.