1. Comparison

We have implemented a prototype implementation in `Maple 17` of Algorithms **??** and **??**. In this section, we compare the performance of these algorithms on a set of benchmarks polynomials (see `http://homepages.math.uic.edu/~jan/`). All the experiments were made on a personal computer Intel(R) Xeon(R) CPUs E5-2640 with 2.50 GHz processor, 28.3G RAM and 64 bits running under Windows 7 operating system. All computations have been done over the field $\mathbb{Q}$. The results are shown in the following tables where the second column indicates the CPU time in hours (h), minutes (m) and seconds (s). The third column represents the amount of used memory in gigabytes. The *redz* column shows the number of zero reductions in the computation of H-basis by the corresponding algorithm. The last column gives the number of polynomials in the output H-basis. The symbol $''-''$ indicates that the computation did not finish within 48 hours and we stopped it.

Our experiments show that the proposed strategy to compute a minimal basis for the syzygy module leads to dramatic decrease in the number of zero reductions. As a result, the time of producing H-bases in our algorithm reduces (in some examples so significantly) in comparison to algorithm **??**.

In some examples (such as Gerdt2) it is seen that computing H-basis is more time-consuming via our algorithm than algorithm **??**, though the number of zero reductions is reduced. The reason for this is the use of `basis` command of `Maple` to compute the syzygy module basis which is usually very efficient because of using the F4 algorithm. Another improvement is deduced from our algorithm is that a comparison of the *poly* column in the following tables shows that our algorithm computes less new generators to achieve H-bases than algorithm **??**. In the other words computed H-basis in our algorithm has smaller generating set than the other one. So the above observations confirm that applying our method to compute the syzygy module bases in the H-bases construction algorithm makes it efficient.

Sendra	time	memory	redz	poly
Alg **??**	6s	0.38	2	4
Sauer	13s	0.85	24	5

Conform1	time	memory	redz	poly
Alg **??**	19s	0.98	9	10
Sauer	42s	2.65	102	15

Redeco5	time	memory	redz	poly
Alg **??**	4m-12s	1.04	10	5
Sauer	4m-37s	19.62	36	5

Lorentz	time	memory	redz	poly
Alg **??**	6s	3.45	11	5
Sauer	5m-11s	20.05	68	6

Wisepfening	time	memory	redz	poly
Alg **??**	41m-47s	158.1	2	5
Sauer	2h-46m-9s	712.39	24	6

Arnborg-Lazard	time	memory	redz	poly
Alg **??**	4m-10s	12.73	24	28
Sauer	>48h	-	-	-

Noon	time	memory	redz	poly
Alg **??**	2h-7m-40s	196.4	15	7
Sauer	8h-5m	2006.63	467	7

Liue	time	memory	redz	poly
Alg **??**	8m-46s	1.62	11	5
Sauer	26h-33m-50s	6686.07	88	11

Random	time	memory	redz	poly
Alg **??**	15m-43s	255.041	24	32
Sauer	>48h	-	-	-

Gerdt2	time	memory	redz	poly
Alg **??**	1h-4m-55s	276.51	3	6
Sauer	1h-2m-53s	32.28	50	7

Acknowledgement

The research of the first author was in part supported by a grant from IPM (No. 94550420). The authors would like to thank Prof. H. Michael Möller for his valuable comments and persistent efforts.