Nevertheless, as the third example shows, it is sometimes a good idea to reduce a polynomial f in the set of generators by $F \setminus \{f\}$.

	`sagbi.lib`	`sagbi2.lib`
T-polynomials 1	144.1	38.4
T-polynomials 2	121.2	22.1
T-polynomials 3	659.4	200.7
Reduction 1	2095.1	1289.3
Reduction 2	1014.5	43.9
Reduction 3	11609.9	179.4
SAGBI construction 1	22.9	<1
SAGBI construction 2	636.3	<1
SAGBI construction 3	4.6	23.9
SAGBI partial 1		
3 iterations	<1	<1
4 iterations	<1	<1
5 iterations	8.7	<1
6 iterations	774.3	12.3
SAGBI partial 2		
10 iterations	<1	<1
20 iterations	16.3	2.4
30 iterations	206.2	27.7
40 iterations	1387.1	207.7
50 iterations	6556.3	1036.5

TABLE 1. Timings in seconds, computations performed on a 1600 MHz single core computer

3. Future work

The combined approach used in the implementation in `sagbi2.lib` seems to perform well in comparison to the other implementations. But there is still potential for improvement. First of all, it would be a good idea to develop a specialized Buchberger algorithm for binomial ideals. Secondly, one could think about choosing a suitable weighted monomial ordering as the binomial ideal involved is quasi-homogeneous. Thirdly, as already mentioned above, it is sometimes helpful to reduce a generator by the set of all other generators (with or without tail-reduction) to possibly lower the amount of terms of each generator or even the number of generators considered. Thus one could develop some heuristics to decide when this is useful.

Moreover, new features such as the computation of the reduced SAGBI basis could be added.