Complementary experiments for the evaluation of UFA inference algorithms

D. Fredouille

October 31, 2003

1 What is this text ?

This text is an complement to the paper [CF03] on the inference of unambiguous automata. It relates experiments completing the results obtained on benchmarks used in the article. These experiments enable a better comparison of the evaluated algorithms. A discussion on the results is given. Definitions and notations are taken from the article [CF03].

2 Complementary experiments: inferring DFAs with the hill-climbing heuristic

2.1 Description

The article [CF03] gives an experimental comparison of different greedy algorithms. The first infers DFAs with the EDSM heuristic [LPP98], the second UFAs with the hill-climbing heuristic, and the last algorithm is DLT2, inferring RFSAs [DLT01].

The complementary experiment given here consisted in inferring DFAs with the hill-climbing heuristic. This means that the deterministic merging implying the most state-merging for determinisation is choosen at each step of the inference algorithm. As for other algorithms, we used this heuristic in two different frameworks : the biased inference (or compatibility) and

Generator	Regular expressions			
size of $\mathcal{S}$	50	100	150	200
MAJ	64.7	66.7	62.3	62.8
Db*edsm*	**83.7**	85.5	91.8	92.1
D*edsm*	79.5	81.7	89.7	93.1
Db*hc*	80.4	89.4	**93.9**	93.8
D*hc*	77.4	81.5	83.5	91.9
Ub*hc*	75.8	82.9	91.7	91.2
U*hc*	76.0	81.5	88.8	91.0
DLT2	81.7	**91.7**	92.3	**95.9**

Figure 1: Recognition level of algorithms for the languages obtained by the generator of regular expressions.

the unbiased inference (or functionality). The corresponding algorithms are respectively denoted by Db*hc* and D*hc*.

2.2 Results

Arrays 1 to 8 give results obtained for the different algorithms (with more details than in the article). These results include runs of Db*hc* and D*hc*.

3 Discussion on the results

3.1 Concerning the EDSM and the hill-climbing heuristics

The EDSM heuristic has been developed for DFAs inference and evaluated on automata obtained by the Abbadingo DFAs generator. The experiments show that this heuristic is very specialized. Indeed, EDSM seems a very interesting heuristic (better than hill-climbing) when we infer DFAs and that the target is a obtained by a DFA generator (matches d10 and d11).

However, when we consider other generators (regular expressions, NFA or UFA), the EDSM heuristic gives less interesting results (matches e10, e11, n10, n11, u10 and u11). Indeed, the hill-climbing heuristic seems to be (in general) better than EDSM on these benchmarks.

Generator	NFA			
size of $\mathcal{S}$	50	100	150	200
MAJ	69.0	66.2	65.7	67.8
Db*edsm*	67.1	70.0	73.1	73.3
D*edsm*	67.0	67.6	70.7	71.0
Db*hc*	69.0	71.8	75.9	76.8
D*hc*	66.3	67.4	71.8	72.9
Ub*hc*	**70.4**	70.8	74.0	73.1
U*hc*	67.0	71.2	73.7	71.3
DLT2	69.8	**74.8**	**77.1**	**79.4**

Figure 2: Recognition level of algorithms for the languages obtained by the generator of NFAs.

Generator	UFA			
size of $\mathcal{S}$	50	100	150	200
MAJ	83.8	82.1	81.4	81.9
Db*edsm*	89.2	91.1	94.3	93.2
D*edsm*	79.1	81.0	89.6	90.2
Db*hc*	**91.2**	**92.0**	**94.7**	**94.1**
D*hc*	77.3	82.4	89.2	88.4
Ub*hc*	90.7	91.9	94.2	93.8
U*hc*	89.7	89.8	92.5$*$	91.6
DLT2	88.6	90.4	91.9	92.7

Figure 3: Recognition level of algorithms for the languages obtained by the generator of UFAs.

Generator	DFA			
size of $\mathcal{S}$	50	100	150	200
MAJ	70.7	71.0	72.5	73.8
Db*edsm*	69.1	**73.3**	74.8	76.3
D*edsm*	65.7	68.3	70.4	74.7
Db*hc*	65.6	69.8	70.5	74.5
D*hc*	63.8	67.7	70.8	73.8
Ub*hc*	70.4	**73.4**	74.5	77.5
U*hc*	**71.1**	72.9	**75.9**	**77.8**∗
DLT2	61.9	65.1	68.3	70.7

Figure 4: Recognition level of algorithms for the languages obtained by the generator of DFAs.

	Generator	Regular expressions			
	size of $\mathcal{S}$	50	100	150	200
e1	D*edsm*-**Db*edsm***	7,12,**11**	5,13,**12**	5,16,**9**	3,23,**4**
e2	Ub*hc*-**Db*edsm***	5,12,**13**	11,6,**13**	**10**,14,6	6,10,**14**
e3	U*hc*-**D*edsm***	**11**,10,9	**10**,11,9	8,13,**9**	6,8,**16**
e4	**Db*hc***-D*hc*	**10**,15,5	**13**,14,3	**14**,14,2	**6**,23,1
e5	U*hc*-**Ub*hc***	**9**,13,8	9,5,**16**	5,13,**12**	6,13,**11**
e6	**DLT2**-Ub*hc*	**15**,8,7	**17**,8,5	**10**,10,**10**	**18**,9,3
e7	**DLT2**-Db*edsm*	**11**,11,8	**17**,8,5	7,13,**10**	**8**,20,2
e8	**DLT2**-Db*hc*	**11**,11,8	**12**,11,7	8,13,**9**	**6**,21,3
e9	**Db*hc***-Ub*hc*	**15**,6,9	**13**,11,6	**10**,11,9	**16**,11,3
e10	**Db*hc***-Db*edsm*	7,14,**9**	**11**,10,9	**8**,16,6	**6**,22,2
e11	D*hc*-**D*edsm***	6,13,**11**	**8**,14,**8**	2,15,**13**	**5**,21,4
e12	Ub*hc*-**Db*hc***	9,6,**15**	6,11,**13**	9,11,**10**	3,11,**16**
e13	U*hc*-**Db*edsm***	6,11,**13**	8,10,**12**	6,13,**11**	4,11,**15**

A triple represents the sequence "victories of algorithm 1, number of ties, victories of algorithm 2".

Figure 5: Matches between algorithms for the languages obtained by the generator of regular expressions.

	Generator	NFA			
	size of $\mathcal{S}$	50	100	150	200
n1	D*edsm*-**Db*edsm***	10,8,**12**	5,13,**12**	5,10,**15**	6,12,**12**
n2	**Ub*hc***-Db*edsm*	**16**,9,5	**11**,12,7	**12**,10,8	**12**,10,8
n3	**U*hc***-D*edsm*	**14**,8,8	**15**,6,9	**14**,9,7	**16**,3,11
n4	**Db*hc***-D*hc*	**15**,8,7	**13**,14,3	**15**,9,6	**11**,14,5
n5	U*hc*-**Ub*hc***	4,11,**15**	4,20,**6**	5,18,**7**	6,11,**13**
n6	**DLT2**-Ub*hc*	9,6,**15**	**12**,13,5	**14**,10,6	**16**,11,3
n7	**DLT2**-Db*edsm*	**14**,8,8	**17**,8,5	**13**,15,2	**17**,11,2
n8	**DLT2**-Db*hc*	**12**,12,6	**11**,12,7	**8**,14,**8**	**13**,12,5
n9	**Db*hc***-Ub*hc*	9,7,**14**	**11**,11,8	**12**,14,4	**15**,8,7
n10	**Db*hc***-Db*edsm*	**11**,14,5	**12**,13,5	**14**,12,4	**12**,16,2
n11	**D*hc***-D*edsm*	6,16,**8**	4,21,**5**	**7**,19,4	**13**,11,6
n12	Ub*hc*-**Db*hc***	**14**,7,9	8,11,**11**	4,14,**12**	7,8,**15**
n13	U*hc*-**Db*edsm***	**13**,9,8	**13**,12,5	**11**,11,8	**11**,8,**11**

A triple represents the sequence "victories of algorithm 1, number of ties, victories of algorithm 2".

Figure 6: Matches between algorithms on the benchmark obtained by the NFA generator.

	Generator	UFA			
	size of $\mathcal{S}$	50	100	150	200
u1	D*edsm*-**Db*edsm***	1,4,**25**	1,6,**23**	2,13,**15**	6,6,**18**
u2	**Ub*hc***-Db*edsm*	**17**,6,7	**13**,14,3	7,15,**8**	**13**,9,8
u3	**U*hc***-D*edsm*	**21**,3,6	**22**,6,2	**14**∗,8,7	**12**,6,**12**
u4	**Db*hc***-D*hc*	**25**,4,1	**23**,6,1	**17**,11,2	**23**,6,1
u5	U*hc*-**Ub*hc***	5,15,**10**	4,14,**12**	∗3,15,**11**	3,12,**15**
u6	DLT2-**Ub*hc***	6,11,**13**	4,14,**12**	3,13,**14**	6,13,**11**
u7	DLT2-**Db*edsm***	8,13,**9**	6,14,**10**	1,14,**15**	8,10,**12**
u8	DLT2-**Db*hc***	4,13,**13**	4,13,**13**	0,11,**19**	7,10,**13**
u9	**Db*hc***-Ub*hc*	11,7,**12**	**6**,19,5	**13**,11,6	**10**,11,9
u10	**Db*hc***-Db*edsm*	**10**,17,3	**7**,22,1	**8**,18,4	**7**,21,2
u11	**D*hc*-D*edsm***	**12**,7,11	**13**,8,9	**10**,10,**10**	6,13,**11**
u12	Ub*hc*-**Db*hc***	**12**,7,11	5,19,**6**	6,11,**13**	9,11,**10**
u13	U*hc*-**Db*edsm***	**11**,9,10	9,10,**11**	4∗,12,**13**	4,15,**11**

A triple represents the sequence "victories of algorithm 1, number of ties, victories of algorithm 2".

Figure 7: Matches between algorithms on the benchmark obtained by the UFA generator.

	Generator	DFA			
	size of $\mathcal{S}$	50	100	150	200
d1	D*edsm*-**Db*edsm***	4,10,**16**	2,7,**21**	2,12,**16**	7,10,**13**
d2	**Ub*hc***-Db*edsm*	**13**,13,4	**11**,14,5	**8**,16,6	**15**,11,4
d3	**U*hc***-D*edsm*	**19**,10,1	**19**,8,3	**24**,5,1	**19**∗,6,4
d4	**Db*hc***-D*hc*	**8**,19,3	**14**,13,3	**6**,18,**6**	**9**,16,5
d5	**U*hc***-Ub*hc*	**8**,16,6	4,21,**5**	**10**,18,2	**10**∗,16,3
d6	DLT2-**Ub*hc***	1,7,**22**	1,2,**27**	2,6,**22**	2,5,**23**
d7	DLT2-**Db*edsm***	1,7,**22**	3,4,**23**	0,10,**20**	2,10,**18**
d8	DLT2-**Db*hc***	2,11,**17**	1,12,**17**	2,17,**11**	1,16,**13**
d9	Db*hc*-**Ub*hc***	1,12,**17**	2,8,**20**	0,10,**20**	4,9,**17**
d10	Db*hc*-**Db*edsm***	3,12,**15**	2,12,**16**	1,14,**15**	3,18,**9**
d11	D*hc*-**D*edsm***	4,15,**11**	6,14,**10**	**7**,17,6	4,18,**8**
d12	**Ub*hc***-Db*hc*	**17**,12,1	**20**,8,2	**20**,10,0	**17**,9,4
d13	**U*hc***-Db*edsm*	**14**,12,4	**10**,15,5	**16**,9,5	**16**∗,10,3

A triple represents the sequence "victories of algorithm 1, number of ties, victories of algorithm 2".

Figure 8: Matches between algorithms on the benchmark obtained by the DFA generator.

3.2 The choice between biased inference (compatibility) and unbiased inference (functionality).

3.2.1 Results

Algorithms using compatibility are much better, on benchmarks created such that L and $\Sigma^* \setminus L$ are not "symmetrical" (NFAs, regular expressions and UFAs), than algorithms using functionality (all together, in matches e5, n5 and u5, Ub*hc* won 135 times against 63 for U*hc*, in matches e1, n1 and u1 Db*edsm* won 168 times against 56 for D*edsm*, in matches e4, n4 and u4 Db*hc* won 175 times against 37 for D*hc*).

The use of compatibility improving inference results on benchmarks, we can remark that the comparison provided between EDSM and DLT2 in [DLT01] has to be updated. The use of compatibility (with the EDSM heuristic or with the hill-climbing heuristic) implied that Db*edsm* and Db*hc* are not completely irrelevant when the target is obtained by the NFA generator or the regular expression generator. Indeed, the performance of these algorithms are closer (than we previously thought) to DLT2 on the NFAs benchmark; on the regular expressions benchmark they are even better in some cases.

3.2.2 Discussion on the results

The choice of using compatibility or functionality does influence the results. Even if this conclusion can seem very logical a posteriori, matches between U*hc*-Ub*hc* between D*hc*-Db*hc* and between D*edsm*-Db*edsm* show that this influence is strong.

This can be explained because these two way of stopping generalization have not exactly the same goal. Compatibility aims at inferring a language while given counter-examples and functionality aims at discriminating positive and negative languages. In this latter framework, counter-examples should have a meaning, which is not the case in the former.

This can explain why compatibility is much better than functionality on the considered benchmarks. However, on the benchmark of DFAs which is generated in a symmetrical manner, the choice of compatibility is also better than the choice of functionality when inferring DFAs (matches d1 and d4). On the same part of benchmarks, the choice of functionality is better when inferring UFAs (match d5, U*hc* being the best algorithm on this benchmark).

Then it seems that some - actually unknown - parameters enabling to choose between compatibility and functionality have to be taken into account.

3.3 Algorithms best suited to benchmarks

3.3.1 Results

Each algorithm seems to be better adapted to different subparts of the benchmarks. When considering the NFAs generator, DLT2 seems to be the best (table 2), for the UFAs generator, Db*hc* is better adapted (very close to Ub*hc*, see table 3), for the DFAs generator, U*hc* has the best recognition levels (table 4). For regular expressions, DLT2 has the best recognition level but algorithms for DFAs inference have very close scores (depending on the case Db*edsm* or Db*hc*).

More surprisingly, we can see that this is a UFA inference algorithm which is the best on the DFAs benchmark, and an algorithm for DFAs inference which is the best on the UFAs benchmarks.

3.3.2 Discussion on the results

As already remarked by [DLT01], the generation mode of languages has a huge influence on the results of the algorithms. This can be interpreted by considering that each algorithm is looking for some particular "structures" in the languages. If these structures are close to the one obtained by a given generator, then the algorithm has an advantage on the languages obtained by this generator. From this interpretation, it seems strange that the UFA inference algorithm U*hc* is the best on the DFAs benchmark, and that the DFAs inference algorithm Db*hc* is the best on the UFAs benchmark.

We propose in the following a discussion of these results.

- Is Db*hc* really the best forUFAs inference ?
  If we detail the difference between Ub*hc* and Db*hc* for the inference on the UFAs benchmark, we can see that on match u12 Db*hc* is the winner 40 times against 32 for Ub*hc* with 48 ties. The two algorithms have in fact nearly the same results on this benchmark. However, it is strange that Ub*hc* did not have the advantage. It is possible that this problem is linked with the lack of canonical form for UFAs. The algorithm has no way to choose between two equivalent automata with

the same language. It therefore returns an intermediate solution of the different possible automata.

- Why U*hc* is the best for DFAs inference ?
  First, let us remark that the difference obtained between algorithms for UFAs and DFAs inference is more important than in the previous case. On this benchmark, the best algorithm after U*hc* is Db*edsm*. The match d13 between U*hc* and Db*edsm* gives to the first 56 victories against 17 for the second (with 46 ties).

  We can explain this result by the conjunction of two phenomena. The first is that the drawback of UFAs inference algorithms (explained in the previous paragraph and due to the lack of canonical form) is here less important. Indeed, the target languages being generated as DFAs, we can think that there are no UFAs (for the same language) of size inferior or equal to the size of the generated automata. Therefore, the algorithm does not has to choose between different automata with the same language, and the target can be more easily reached. Moreover, the fact that algorithm U*hc* is better than algorithm D*hc* or Db*hc*, is probably due to a more cautious state-merging strategy. Indeed, choosing a wrong unambiguous state-merging at a step of the algorithm creates less constraints on following state-mergings than choosing a wrong deterministic state-merging. A wrong unambiguous merging can therefore be partly corrected by the following unambiguous mergings.

References

[CF03] Coste (F.) et Fredouille (D.). – Unambiguous automata inference by means of state-merging methods. *European Conference on Machine Learning*, 2003, pp. 60–71.

[DLT01] Denis (F.), Lemay (A.) et Terlutte (A.). – Learning regular languages using RFSA. *In : Proceedings of the 12th International Conference on Algorithmic Learning Theory, ALT'01*, pp. 348–363. – 2001.

[LPP98] Lang (K. J.), Pearlmutter (B. A.) et Price (R. A.). – Results of the abbadingo one DFA learning competition and a new evidencedriven state merging algorithm. *Lecture Notes in Computer Science*, vol. 1433, 1998, pp. 1–12.