

Local Search Heuristics for NFA State Minimization Problem*

Andrey V. Tsyganov

Department of Higher Mathematics, Ulyanovsk State Pedagogical University, Ulyanovsk, Russia
Email: andrew.tsyganov@gmail.com

Received July 2, 2012; revised July 26, 2012; accepted August 6, 2012

ABSTRACT

In the present paper we introduce new heuristic methods for the state minimization of nondeterministic finite automata. These methods are based on the classical Kameda-Weiner algorithm joined with local search heuristics, such as stochastic hill climbing and simulated annealing. The description of the proposed methods is given and the results of the numerical experiments are provided.

Keywords: Nondeterministic Finite Automata; State Minimization; Heuristics; Local Search; Parallelism

1. Introduction

Finite automata (FA) are widely used in various fields and especially in the theory of formal languages. We suppose that the reader is familiar with the basics of automata theory (see, for example, [1]) and provide only some necessary definitions.

Let Σ be an alphabet and $w = a_1 a_2 \dots a_n$ where $\forall i: a_i \in \Sigma$ be a word. A set of words $L \subseteq \Sigma^*$ is called a *language*.

The *nondeterministic finite automaton* (NFA) is a 5-tuple $A = (Q, \Sigma, \Delta, I, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $I \subseteq Q$ and $F \subseteq Q$ are respectively the sets of initial and final states. Transitions of the automaton A are often described by the transition function $\delta: Q \times \Sigma \rightarrow 2^Q$. The NFA is called *deterministic* (DFA) iff $|I| = 1$ and $\forall q \in Q, \forall a \in \Sigma: |\delta(q, a)| = 1$ (or $|\delta(q, a)| \leq 1$).

Finite automata may be used to recognize and define languages. Two automata are called equivalent if they recognize one and the same language. For each NFA the equivalent DFA may be constructed using the powerset construction process (each state of such DFA is a subset of states of the original NFA).

For a word $w = a_1 a_2 \dots a_n$ the reverse word is $\bar{w} = a_n a_{n-1} \dots a_1$, for a language L the *reverse language* is $\bar{L} = \{\bar{w} | w \in L\}$ and for an automaton $A = (Q, \Sigma, \Delta, I, F)$ the *reverse automaton* is $\bar{A} = (Q, \Sigma, \bar{\Delta}, F, I)$ where

$$(q_i, a, q_j) \in \bar{\Delta} \text{ iff } (q_j, a, q_i) \in \Delta.$$

For a given language L a DFA which recognizes it and has the minimum possible number of states is called the *canonical automaton* and a DFA which recognizes $\bar{L}$ and has the minimum possible number of states is called the *reverse canonical automaton* (these automata are unique for L up to isomorphism).

The NFA state minimization problem is formulated as follows: for a given NFA A find an automaton which is equivalent to it and has the minimum possible number of states. Note that solution of this problem may not be unique. As it is shown in [2] the state minimization problem for NFA is PSPACE-complete. The worst case complexity for the same problem for DFA is

$$O(|Q| \cdot |\Sigma| \cdot \log |Q|).$$

All known exact NFA state minimization methods use different types of exhaustive search and are computationally hard. Very often they become impractical even for relatively small automata. This is one of the reasons why they are not implemented in software tools that deal with finite automata and related structures, such as AMoRE [3], FSM [4], Vaucanson [5], JFlap [6]. Moreover, only few of such tools provide heuristic NFA state minimization algorithms.

In the present paper we propose new heuristic methods for NFA state minimization problem which are based on the classical Kameda-Weiner algorithm [7] and well-known local search heuristics (metaheuristics). The novelty of these methods is that the most time consuming part of the exact algorithm is replaced with fast heuristic procedures. The obtained methods are not exact but they allow to reduce minimization time.

*The work is supported by the Ministry of Education and Science of the Russian Federation (grant number 1.919.2011).

The remainder of the paper has the following structure. In Sections 2 and 3 a brief description of the Kameda-Weiner algorithm and local search heuristics is given, in Section 4 the proposed algorithm is described and in Section 5 the results of some numerical experiments are provided.

2. Kameda-Weiner Algorithm

Lets us consider the brief description of the Kameda-Weiner algorithm (for detailed description see [7]).

Suppose that NFA A is given. The algorithm searches for the minimum state NFA(s) equivalent to A using binary matrix RAM (Reduced Automaton Matrix) which is constructed as follows.

First, canonical automaton B and reverse canonical automaton C for A are constructed. Note that each state of these automata is a subset of states of the automaton A . Then, for each nonempty states p_i of $B (i=1, \dots, m)$ and q_j of $C (j=1, \dots, n)$ the element r_{ij} of the RAM is defined by the following formula

$$r_{ij} = \begin{cases} 0, & p_i \cap q_j = \emptyset, \\ 1, & p_i \cap q_j \neq \emptyset. \end{cases}$$

Let X be a subset of rows and Y a subset of columns of the RAM. Then $X \times Y$ is called a (prime) grid if it satisfies the following conditions: 1) all intersections of its rows and columns contain 1 s; and 2) neither X nor Y can be enlarged without violating the first condition.

The set of grids cover RAM if each 1 in it belongs to at least one grid in the set. A minimum cover of RAM is a cover which consists of the minimum number of grids.

Let us consider the NFA A with transition table shown in **Table 1** (the example is taken from [7]).

Tables 2 and **3** show the canonical automaton B and the reverse canonical automaton C of A respectively. RAM of A is presented in **Table 4**.

There are 4 grids in RAM: 1) $\{1,3\} \times \{2,3\}$, 2) $\{2,3\} \times \{1,2\}$, 3) $\{3\} \times \{1,2,3\}$, 4) $\{1,2,3\} \times \{2\}$. It is easy to see that the first two grids make the minimum cover of RAM.

Given a cover of RAM one can construct an NFA which *may be* equivalent to the original NFA A (in this case the cover is called *legitimate*). This is done by the means of the special *intersection rule*. The number of states in the constructed NFA equals to the number of grids in the cover. In the considered example the minimum cover of RAM is legitimate and yields the minimum state NFA shown in **Table 5**.

The general schema of the Kameda-Weiner minimization technique is described by **Algorithm 1**. Note that steps 1, 3 and 4 of this algorithm theoretically have exponential complexity. On practice the construction of the canonical automata usually performed rather quick and the most time consuming parts of the algorithm are steps

Table 1. NFA A.

		a	b
$\rightarrow$	1	$\{1,3\}$	$\{2\}$
$\leftarrow$	2	$\{1\}$	$\{2,3\}$
$\leftarrow$	3	$\{1\}$	$\{3\}$

Table 2. Canonical automaton B.

		a	b
$\leftarrow$	$b_1 = \{2,3\}$	b_2	b_1
$\rightarrow$	$b_2 = \{1\}$	b_3	b_1
$\leftarrow$	$b_3 = \{1,3\}$	b_3	b_1

Table 3. Reverse canonical automaton C.

		a	b
$\leftarrow$	$c_1 = \{1\}$	c_2	c_4
$\leftarrow$	$c_2 = \{1,2,3\}$	c_2	c_2
$\rightarrow$	$c_3 = \{2,3\}$	c_1	c_2
	$c_4 = \emptyset$	c_4	c_4

Table 4. RAM of A.

	$\{1\}$	$\{1,2,3\}$	$\{2,3\}$
$\{2,3\}$	0	1	1
$\{1\}$	1	1	0
$\{1,3\}$	1	1	1

Table 5. Minimum state NFA equivalent to A.

		a	b
$\rightarrow$	p_1	$\{p_1, p_2\}$	$\{p_2\}$
$\leftarrow$	p_2	$\{p_1\}$	$\{p_2\}$

Algorithm 1. Kameda-Weiner algorithm.

Require: NFA A

- 1: Construct canonical automata B and C
- 2: Construct RAM
- 3: Find all grids of RAM
- 4: Find minimum legitimate cover(s) of RAM and construct minimum state NFA(s) using intersection rule

Ensure: Minimum state NFA(s) equivalent to A

3 and 4 which perform the exhaustive search for grids and covers respectively.

3. Local Search Heuristics

Local Search (LS) is a group of metaheuristic optimization techniques which are widely used especially in combinatorial optimization. Its general schema is described by **Algorithm 2**.

Each LS algorithm starts from some initial solution (line 1) and then iteratively updates it (lines 2 - 4) until stop condition is satisfied (in the most simple case it is the maximum number of steps). The Neighbor() function finds neighbors of the current solution Solution and the Update() function changes the current solution depending on the found neighbors. The quality of the solutions is compared using the special Cost() function.

The simplest LS algorithm is called Hill Climbing (HC). In HC the Update() function always selects the best neighbor of the current solution (the steepest move). The Stochastic Hill Climbing (SHC) is a variant of HC which randomly chooses one of the neighbors and decides whether to move to it or to consider another neighbor.

The disadvantage of the HC algorithms is that they can easily stuck in the local optimum. The Simulated Annealing (SA) is a more complicated LS algorithm which tries to avoid this problem. **Algorithm 3** shows in detail the minimization process using SA.

The distinctive feature of the algorithm is the usage of the control parameter T (temperature) which slowly decreases as the number of iterations k increases. The Neighbor() function generates a neighbor of the current solution and the Update() function accepts it with the probability

$$P = \begin{cases} 1, & \text{if } \Delta \leq 0; \\ e^{-\frac{\Delta}{T}}, & \text{if } \Delta > 0, \end{cases} \quad (1)$$

where $\Delta = \text{Cost}(\text{NewSolution}) - \text{Cost}(\text{Solution})$.

As it follows from (1) the algorithm always accepts the generated neighbor if it is not worse than the current solution. If the neighbor is worse than the current solution it is accepted with some probability which decreases as the temperature decreases. So, the worse moves are made more often in the beginning of the optimization process. In classical SA the best solution is not stored (lines 2, 3, 8 - 12 are missing) and the algorithm returns the current solution.

More details on heuristic optimization algorithms can be found in [8,9].

4. Combinig Kameda-Weiner Algorithm with Local Search Heuristics

First of all let us consider in more detail the last step of

Algorithm 2. Local search.

Require: InitialSolution

```

1:  Solution := InitialSolution
2:  repeat
3:      Solution := Update(Neighbor(Solution))
4:  until StopCondition()

```

Ensure: Solution

Algorithm 3. Simulated annealing.

Require: InitialSolution

```

1:  Solution := InitialSolution
2:  BestSolution := Solution
3:  BestCost := Cost(Solution)
4:  T := InitialTemperatue()
5:  k := 0
6:  while not StopCondition() do
7:      NewSolution := Neighbor(Solution)
8:      NewCost := Cost(NewSolution)
9:      if NewCost < BestCost then
10:         BestSolution := NewSolution
11:         BestCost := NewCost
12:      end if
13:      Solution := Update(Solution, NewSolution, T)
14:      k := k + 1
15:      T := UpdateTemperature(T, k)
16:  end while

```

Ensure: BestSolution

the Kameda-Weiner algorithm, i.e. the exhaustive search for minimum legitimate covers (see **Algorithm 4**). Here M is a set of minimum state NFAs. IsLegitimateCover() function tests whether the set of grids is a legitimate cover and IntersectionRule() constructs NFA using the cover. The bounds $i_{\min}$ and $i_{\max}$ of the main loop may be calculated as follows: $i_{\min} = \lceil \log_2 N_B \rceil$ where N_B is the number of states in canonical automaton B , $i_{\max} = \min(N_G, N_A - 1, N_B - 1)$ where N_G is the number of grids in RAM, N_A is the number of states in A and N_B is the number of states in B .

For each step of the outer loop in the inner loop (lines 5 - 10) all possible i -combinations of grids have to be analyzed. The idea of the heuristic methods proposed in this paper is to replace this computationally hard process

Algorithm 4. Exhaustive search for minimum legitimate covers.

```

1: Calculate  $i_{\min}$  and  $i_{\max}$ 
2:  $M := \emptyset$ 
3: for  $i$  from  $i_{\min}$  to  $i_{\max}$  do
4:    $found := false$ 
5:   for all  $i$ -combinations of grids  $Comb$  do
6:     if IsLegitimateCover( $Comb$ ) then
7:        $M := M \cup \{IntersectionRule(Comb)\}$ 
8:        $found := true$ 
9:     end if
10:   end for
11:   if  $found = true$  then
12:      $break$ 
13:   end if
14: end for

```

by non-exhaustive LS procedures (of course, this may result in obtaining approximate solutions, *i.e.* *reduced* NFAs). This approach is described by **Algorithm 5**.

So, we use LS in the Kameda-Weiner algorithm to find minimum covers of RAM and then analyze their size and legitimacy. Now let us consider the details of this process. The solution of both LS methods (SHC and SA) for the considered problem is a cover which is coded by a binary vector where 1 in i -th position means that i -th grid is included in a cover and 0 means that it is not included. In the considered example vector $(1, 1, 0, 0)$ means that only first two grids are included in the cover.

To start search LS algorithms need the initial solution. The simplest way to setup the initial solution is to use the trivial solution with all bits set to 1. To obtain nontrivial initial solution **Algorithm 6** may be used.

The Cost() function simply counts the number of 1s in the vector and the Neighbor() function inverts several bits in it. After creating a neighbor of the current solution we need to check its feasibility (*i.e.* to check whether the obtained set of grids covers all 1s in RAM). If the constructed neighbor is not a feasible solution then we add one or several 1s to it using algorithm similar to **Algorithm 6**. (e.g., in the considered example the solution $(1, 0, 0, 0)$ is not feasible and it needs to be corrected).

To ensure the diversity of minimum covers one may run LS several times or use parallel versions of LS where each thread starts from its own initial solution. Note also that if the minimum legitimate covers not found then both exact and heuristic methods return the canonical automaton B if it has less number of states than the given

Algorithm 5. Heuristic search for minimum legitimate covers.

```

1: Calculate  $i_{\min}$  and  $i_{\max}$ 
2: Find minimum cover(s) of RAM using LS
3: if cover(s) of size less or equal to  $i_{\max}$  found then
4:   Select minimum legitimate cover(s) and construct minimum state NFA(s) using intersection rule
5: end if

```

Algorithm 6. Construction of initial solution for LS algorithms.

```

1: repeat
2:   Take next 1 in the RAM
3:   Randomly choose a grid that covers this 1
4:   Set to 1 the corresponding bit of the InitialSolution
5:   Eliminate all 1s that are covered by the chosen grid from the RAM
6: until there are 1s in the RAM

```

automaton A .

5. Numerical Experiments

We have implemented the proposed NFA minimization methods in the ReFaM project. ReFaM (Rational Expressions and Finite Automata Minimization) is a part of the HeO (Heuristic Optimization) library. This library is a cross-platform open source project written in C++ that provides several parallel metaheuristic optimization methods such as Genetic Algorithm (GA), Simulated Annealing (SA), Stochastic Hill Climbing (SHC) and Branch and Bound (BnB). These methods are implemented as algorithmic skeletons using metaprogramming, pattern design and different parallelization techniques (OpenMP and MPI). The latest version of the library may be obtained via SVN (the homepage of the project: <http://code.google.com/p/heo/>).

In the exact version of the Kameda-Weiner algorithm the search for grids of the RAM and the search for minimum legitimate covers were parallelized using OpenMP and MPI techniques. In the heuristic versions of this algorithm the parallel versions of SHC and SA algorithms of the HeO library were used. Each version of the algorithm is implemented in a separate solver.

Let us compare the performance of the exact and heuristic solvers for the random sample of the 100 pairwise inequivalent trim NFAs generated with the following parameters: number of states $|Q| = 6$, number of initial states $|I| = 1$, number of final states $|F| = 2$, alphabet

size $|\Sigma| = 2$, transition density $D = \frac{T}{|Q|^2 |\Sigma|} = 0.3$, where

T is the number of automaton transitions. The computational experiments were conducted using the following SMP system: Intel Core 2 Quad Q6600 2.4 Ghz, 4 Gb RAM, MS Windows XP Professional SP 3.

First of all let us consider the results of the exact solver which are presented in **Table 6**. Here, m and n are the number of rows and number of columns in RAM respectively; d is the density of ones in RAM; N_G is the number of grids in RAM; N_M is the number of the minimum state NFAs; $|Q_M|$ is the number of states in the minimum state NFAs; min and max are minimal and maximal values; μ is the mean value; σ is the standard deviation. The results were obtained using 4 threads. The average minimization time is 1.1 seconds and the total number of the minimum state automata found for the whole sample is 268, 9 automata have no equivalent minimum state NFA. As it can be seen from the table some of the automata in the sample have multiple minimum state NFAs (the mean value is 2.95) and the average number of states in the minimum state NFAs is 3.52.

Now let us consider the results of the heuristic solvers obtained with different number of threads $N = 1, 2, 4, 8, 16, 32, 64$ which are presented in **Tables 7** and **8**. Since LS is used only at the last stage of the Kameda-Weiner algorithm the first 4 columns of **Table 6** will be the same for heuristic solvers and we replace them with the following columns: N_T —the total number of minimum (reduced) state NFAs found for the sample, N_U —number of unminimized automata, T —average minimization time in seconds (for columns N_M and $|Q_M|$ the mean values are provided).

As it can be seen from these tables the total number of the minimum state NFAs increases and the number of unminimized automata decreases as the number of threads grows. The average minimization time is very small and remains almost constant up to 4 threads and then increases proportionally to the number of threads because the hardware used for experiments supports simultaneous execution only of 4 threads.

6. Conclusion

In the present paper we have considered new heuristic algorithms for NFA state minimization problem which is known to be computationally hard. These algorithms are a combination of the classical Kameda-Weiner algorithm and local search heuristics which are widely used in combinatorial optimization. The essential feature of the proposed algorithm is that the most time consuming part of the exact algorithm is replaced with fast local search procedures. Numerical experiments have shown that such combination is much less time consuming and allows to

Table 6. Exact algorithm results.

	m	n	d	N_G	N_M	$ Q_M $
min	1	1	0.63	1	0	1
max	16	14	1.00	92	32	5
μ	5.70	6.00	0.73	15.40	2.95	3.52
σ	2.83	2.85	0.05	17.74	5.21	0.95

Table 7. SA algorithm results.

N	N_T	N_U	N_M	$ Q_M $	T
1	69	31	1.00	3.48	0.0071
2	76	28	1.06	3.43	0.0071
4	98	25	1.31	3.47	0.0078
8	114	20	1.43	3.49	0.0148
16	122	20	1.53	3.41	0.0258
32	137	19	1.69	3.41	0.0491
64	157	16	1.87	3.45	0.0917

Table 8. SHC algorithm results.

N	N_T	N_U	N_M	$ Q_M $	T
1	69	31	1.00	3.52	0.0057
2	75	28	1.04	3.42	0.0056
4	95	26	1.28	3.46	0.0059
8	112	21	1.42	3.52	0.0111
16	128	24	1.68	3.42	0.0192
32	148	16	1.76	3.46	0.0364
64	161	16	1.92	3.45	0.0674

obtain acceptable results. In the future we plan to concentrate on the other time consuming part of the Kameda-Weiner algorithm, i.e. the exhaustive search for grids of the RAM.

REFERENCES

- [1] J. E. Hopcroft and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley Publishing Company, Reading, 1979.
- [2] T. Jiang and B. Ravikumar, "Minimal NFA Problems Are Hard," *SIAM Journal on Computing*, Vol. 22, No. 6, 1993, pp. 1117-1141. doi:10.1137/0222067
- [3] V. Kell, A. Maier, A. Potthoff, W. Thomas and U. Wermuth, "AMORE: A System for Computing Automata, Monoids and Regular Expressions," *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science on STACS 89*, Springer-Verlag, New York,

- 1989, pp. 537-538.
- [4] M. Mohri, F. Pereira and M. Riley, "AT&T General-Purpose Finite-State Machine Software Tools," 1997.
<http://www.research.att.com/sw/tools/fsm>
 - [5] S. Lombardy, R. Poss, Y. Régis-Gianas and J. Sakarovitch, "Introducing VAUCANSON," In: O. H. Ibarra and Z. Dang, Eds., *Implementation and Application of Automata*, CIAA 2003, Santa Barbara, 16-18 July 2003, pp. 96-107.
 - [6] S. H. Rodger, "JFLAP: An Interactive Formal Languages and Automata Package," Jones and Bartlett Publishers, Inc., USA, 2006.
 - [7] T. Kameda and P. Weiner, "On the State Minimization of Nondeterministic Finite Automata," *IEEE Transactions on Computers*, Vol. C-19, No. 7, 1970, pp. 617-627.
[doi:10.1109/T-C.1970.222994](https://doi.org/10.1109/T-C.1970.222994)
 - [8] J. Hromkovic, "Algorithmics for Hard Problems—Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics," Springer, Berlin, 2001.
 - [9] F. Glover and G. A. Kohenberger, "Handbook of Metaheuristics," Kluwer Academic Publishers, Boston, 2003.