KYBERNETIKA — VOLUME 18 (1982), NUMBER 2

FAST ALGORITHMS FOR FINDING A SUBDIRECT DECOMPOSITION AND INTERESTING CONGRUENCES OF FINITE ALGEBRAS

JIŘÍ DEMEL

A fast algorithm is suggested for finding a subdirect decomposition of a given finite algebra into subdirectly irreducible ones. This algorithm is an essential improvement of that given in [4]. As a by-product, fast algorithms are presented for finding some interesting congruences of the given algebra.

All algebras are supposed to be finite and given by tables of their operations.

A fast algorithm is suggested for finding a subdirect decomposition of a given finite algebra into subdirectly irreducible ones. This algorithm is an essential improvement of that given in [4]. As a by-product, fast algorithms are presented for finding some interesting congruences of the given algebra.

All algebras are supposed to be finite and given by tables of their operations.

Subdirect product is a useful construction of "complicated" algebras from "simpler" ones. Also, the parallel composition of automata [7] can be regarded as a subdirect product. In [3] a fast algorithm is presented deciding whether a given automaton (Mealy, Moore or Medvedev) is subdirectly irreducible.

Since subdirect products and subdirect irreducibility are closely related to congruneces, this result is based on an algorithm finding minimal non-identical congruences. In [5] this method is generalized to algebras and the algorithm is asymptotically improved. Also, Hopcroft's and Karp's algorithms [8] and [9], originally designed for automata, can be modified to search for interesting congruences of an algebra. The present paper can be viewed as a free continuation of the papers [3] and [5].

By an *algebra* we mean a pair $\mathfrak{A} = (A, F)$ where A is a finite set and F is a finite set of operations. An *operation* f is a mapping $f : A^s \to A$ (A^s is the s-th power of A); the number s is the *arity* of f and we shall denote it by $\operatorname{ar}(f)$.

Throughout the paper n denotes $\operatorname{card}(A)$ and r is the maximal arity of operations. All the running times of algorithms will be given for classes of algebras with the same number of operations that have arity at most r.

A *congruence* on an algebra $\mathfrak{A} = (A, F)$ is an equivalence ε on A satisfying the *substitution property*:

(SP) for every operation $f \in F$ with $\operatorname{ar}(f) = s > 0$ whenever we have an s-tuple $(x_1, \ldots, x_s) \in A^s$ and an element $y \in A$ such that $(x_i, y) \in \varepsilon$ for some i, $1 \leqq i \leqq s$, then $(f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_s), f(x_1, \ldots, x_s)) \in \varepsilon$.

On every algebra we have at least two congruences – the *identical congruence*, denoted by Δ, i.e. $(x, y) \in \Delta$ iff $x = y$, and the *trivial congruence* denoted by ∇, i.e. $\nabla = A^2$. Congruences as equivalences are partially ordered by inclusion.

If ε, δ are equivalences, we denote by $\varepsilon \wedge \delta$ the *meet* of ε, δ, i.e. their intersection, and by $\varepsilon \vee \delta$ the *join* of ε, δ, i.e. the smallest equivalence containing both ε and δ. It is well-known that if ε, δ are congruences, so are $\varepsilon \wedge \delta$ and $\varepsilon \vee \delta$.

Let $\mathfrak{A} = (A, F)$, $\mathfrak{A}_i = (A_i, F_i)$ for $i = 1, \ldots, m$, be algebras with the same arities of operations. Then $\mathfrak{A}$ is a *subdirect product* of the algebras $\mathfrak{A}_i$, $i = 1, \ldots, m$, if $\mathfrak{A}$ is a subalgebra of their direct product and for any i and any $v \in A_i$ there is an element $(x_1, \ldots, v, \ldots, x_m) \in A$.

The algebra $\mathfrak{A}$ is called *subdirectly irreducible* if, whenever $\mathfrak{A}$ is a subdirect product of $\mathfrak{A}_i$, $i = 1, \ldots, m$, then there is j such that the canonical projection $\pi_j : \mathfrak{A} \to \mathfrak{A}_j$ is an isomorphism.

For more details see e.g. [6].

Theorem 1. ([2]) Every algebra is isomorphic to a subdirect product of subdirectly irreducible algebras.

Theorem 2. ([2]) If an algebra $\mathfrak{A}$ is a subdirect product of algebras $\mathfrak{A}_i$, $i = 1, \ldots, m$, then there exist congruences ε_i, $i = 1, \ldots, m$, on $\mathfrak{A}$ such that $\mathfrak{A}_i \cong \mathfrak{A}/\varepsilon_i$ for all $i = 1, \ldots, m$ and $\{\varepsilon_i \mid i = 1, \ldots, m\}$ is a *separative system of congruences*, i.e.

(Sep) $$\bigwedge_{i=1}^{m} \varepsilon_i = \Delta .$$

Theorem 3. ([2]) A finite algebra is subdirectly irreducible iff either it has exactly one minimal non-identical congruence or it has only one element.

Corollary 4. An algebra $\mathfrak{A}/\varepsilon$ is subdirectly irreducible iff

(Max) there exists a pair of distinct elements (x, y) of $\mathfrak{A}$ such that ε is a maximal congruence on $\mathfrak{A}$ with $(x, y) \notin \varepsilon$, i.e. $(x, y) \notin \varepsilon$ and whenever $\varepsilon' \supset \varepsilon$, $\varepsilon' \neq \varepsilon$ then $(x, y) \in \varepsilon'$ for any congruence ε'.

The above Theorems and Corollary 4 offer a strategy for construction of algebras $\mathfrak{A}_i$, $i = 1, \ldots, m$, of which the given algebra $\mathfrak{A}$ is a subdirect product: Having constructed a system of congruences $\{\varepsilon_i \mid i = 1, \ldots, m\}$ satisfying (Sep) and (Max), one can simply construct $\mathfrak{A}_i$ as $\mathfrak{A}/\varepsilon_i$. The construction of each quotient-algebra $\mathfrak{A}/\varepsilon_i$ can be carried out in time $O(p^r)$ where p is the number of all congruence classes of ε_i.

Note that if $m \geqq n$, then at least one congruence can be deleted from the system without violating (Sep) and (Max).

So the original problem has been reduced to the problem

(P1) Given an algebra $\mathfrak{A} = (A, F)$, find a system of congruences $S = \{\varepsilon_i \mid i = 1, \ldots, m\}$ fulfilling (Sep), (Max) and $m < n$.

The problem (P1) will be solved by the algorithm SUBDECOMP and Theorem 9 using a procedure MAXNOTTWO which solves the following problem (P2):

(P2) Given an algebra $\mathfrak{A} = (A, F)$ and a pair $(v, w) \in A^2 \setminus \Delta$, find a congruence ε such that (i) $(v, w) \notin \varepsilon$;
(ii) ε is maximal with respect to (i), i.e. if $\gamma \supset \varepsilon$, $\gamma \neq \varepsilon$ then $(v, w) \in \gamma$ for any congruence γ.

The main part of the procedure MAXNOTTWO is formed by the procedure SAFEPARTITION. In addition, two procedures MAXINEQUIV and MINCONG are used to solve the following problems (P3) and (P4) concerning congruences:

(P3) Given an algebra $\mathfrak{A}$ and an equivalence $\delta \neq \mathrm{V}$, find the greatest congruence $\varepsilon \subseteq \delta$.

This problem can be solved by a slight modification of Hopcroft's algorithm [8] for minimization of finite automata.

(P4) Given an algebra $\mathfrak{A}$, a congruence δ on $\mathfrak{A}$ and a pair $(x, y) \notin \delta$, find the minimal congruence ε with $\delta \subseteq \varepsilon$ and $(x, y) \in \varepsilon$.

This problem can be solved by a modification of Hopcroft's and Karp's algorithm [9] originally designed for testing equivalence of automata.

There is a natural generalization of both (P2) and (P3):

(P5) Given an algebra $\mathfrak{A} = (A, F)$ and a relation $R \subseteq A^2 \setminus \Delta$, find a congruence ε with (i) $\varepsilon \cap R = \emptyset$;
(ii) ε is maximal with respect to (i).

Although this is not going to be used for the problem of subdirect decomposition, it represents an interesting problem in its own right. Its solution is discussed at the end of the paper.

To simplify the description of the algorithms given below, the problems stated above can be, without loss of generality, restricted to unary algebras only. Indeed, this follows from the following observation: As far as the substitution property is considered, the following procedures have no effect:

(i) each nullary operation is omitted,
(ii) each s-ary operation f is replaced by $s \,.\, n^{s-1}$ unary operations obtained by fixing $s - 1$ entries in f.

Hence each algebra can be converted to a unary algebra with the same congruences (considered as equivalences on the same set) in time proportional to the size of all tables of operations. In most cases, this does not influence asymptotically the total time. However the need of the conversion can be avoided by modifying the algorithms to deal with arbitrary algebras.

Thus the descriptions of algorithms will be given for unary algebras only although time bounds will be given for arbitrary algebras.

The conversion of an arbitrary algebra to unary algebra with the same congruences enables us to modify some algorithms, originally designed for automata, to be used for algebras. Names of unary operations correspond to input symbols, the elements of the underlying set of the algebra correspond to states of the automaton. Congruences of the algebra coincide with congruences on the states of the automaton.

Lemma 5 ([8]). There exists a procedure MAXINEQUIV(δ) that replaces a given equivalence δ by the greatest congruence contained in δ and that needs time at most $O(n^r \,.\, \log n)$.

Lemma 6 ([9]). There exists a procedure MINCONG1(δ) that replaces a given equivalence δ by the smallest congruence $\varepsilon \supseteq \delta$ and that needs time at most $O(n^r)$ for $r > 1$ and $O(n \,.\, G(n))$ for $r = 1$, where $G(n) = \min \{i \in \mathbb{N} | \underbrace{\log \log \dots \log}_{i-\text{times}} n \leqq 1\}$.

In the above algorithms MAXINEQUIV and MINCONG1 equivalence δ should be stored and maintained in different data structures. In MAXINEQUIV, classes of equivalence δ are represented by doubly linked lists and by an array that assignes to each element the name of the class in which it is contained. In this data structure operations Delete and Insert can be performed in a constant time.

In MINCONG1, the system of disjoint sets (i.e. classes of the equivalence) is represented by the tree data structure ([10], [11], [1]) for quick operations Union and Find. In this data structure $k \geqq n$ operations Find and $n - 1$ Unions can be performed in time $O(k \,.\, G(n))$, where $G(n) = \{i \in \mathbb{N} | \underbrace{\log \log \dots \log}_{i-\text{times}} n \leqq 1\}$ but if (k/n) growths sufficiently (e.g. if $k/n \geqq \log_2 \log_2 n$) then the time needed for k Finds and $n - 1$ Unions is $O(k)$.

Suppose that for a given element x and a given equivalence δ the procedure FIND(x, δ) yields the name of the equivalence class c in δ such that $x \in c$. Further, suppose that for given two names of classes a, b in δ, the procedure UNION(a, b, c, δ) replaces the classes a, b by their union and call the resulting class c.

Both the data structures mentioned above (i.e. the data structures used in MAXINEQUIV and MINCONG1) cannot be maintained simultaneously, but they can be easily converted each to the other in time $O(n)$.

We shall denote by $\bar{\varepsilon}$ the set of names of congruence classes of equivalence ε.

Lemma 7. There exists a procedure MINCONG(δ, x, y, LIST) that replaces a congruence δ by the smallest congruence ε containing both δ and (x, y), and in LIST returns a sequence of all triples (a, b, c) of names of congruence classes such that during the execution of MINCONG, classes a, b were replaced by $a \cup b$ and the resulting class was called c. The time needed by the procedure MINCONG is $O(p^r)$ for $r > 1$ and $O(p \,.\, G(p))$ for $r = 1$, where p is the number of congruence classes of δ and the function G is as given in Lemma 6.

Proof. One can either apply procedure MINCONG1 to the quotient-algebra $\mathfrak{A}/\delta$ or (which is, in fact, the same) make a further modification of the algorithm to deal with representants of congruence classes only.

Lemma 8. There exist procedures JOIN(α, β) and MEET(α, β) that produce equivalences $\alpha \vee \beta$ and $\alpha \wedge \beta$ in time $O(n)$.

Now, we can describe the procedure SUBDECOMP using the procedure MEET and MAXNOTTWO. The latter will be described below.

Procedure SUBDECOMP:

```
begin S:= ∅; δ := ∇;
  while δ ≠ Δ do
    begin
      choose c ∈ δ̄ with card(c) ≧ 2;
      choose distinct elements v, w ∈ c;
      ε := MAXNOTTWO(v, w);
      δ := MEET(ε, δ);
      insert ε into S
    end;
  return S
end;
```

Theorem 9. The procedure SUBDECOMP solves (P1) in time $O(n \cdot t + n^2)$, where t is time needed for one execution of procedure MAXNOTTWO which solves problem (P2).

Proof. The proof of correctness is easy. The proof of time bound is based on the fact that during repetition of the *while*-loop the number of classes of δ is increased.

The procedure MAXNOTTWO solving (P2) is based on the following lemma:

Lemma 10. Let $\mathfrak{A} = (A, F)$ be an algebra, let $v, w \in A$, $v \neq w$. Let β be a congruence and δ an equivalence such that

(i) $(v, w) \notin \delta$ and $\beta \subseteq \delta$;

(ii) every congruence $\gamma \supseteq \beta$ containing $(x, y) \notin \delta$ contains (v, w).

Then the greatest congruence ε contained in δ (i.e. the solution of (P3)) is also a greatest congruence not containing (v, w) (i.e. the solution of (P2)).

Proof. The statement follows from the fact that (ii) is equivalent to

(ii′) every congruence $\gamma \supseteq \beta$ not containing (v, w) is contained in δ.

The congruence ε is the greatest one fulfilling both $\beta \subseteq \varepsilon$ and $(v, w) \notin \varepsilon$, so it is one of maximal congruences not containing (v, w).

An equivalence δ from Lemma 10 can be constructed by a procedure SAFEPARTITION(v, w) described below.

```
Procedure SAFEPARTITION(v, w):
 begin β := Δ; B := β̄ \ {FIND(v, β)};
   while B ≠ ∅ do
     begin
       choose e ∈ B;
       choose x ∈ e;
       copy β into α;
       MINCONG(α, v, x, LIST);
       if FIND(v, α) = FIND(w, α)
         then B := B \ {e}
         else
           for all (a, b, c) ∈ LIST do
             begin
               UNION(a, b, c, β);
               if a ∈ B & b ∈ B then B := (B \ {a, b}) ∪ {c}
               if a ∈ B & b ∉ B then B := B \ {a};
               if a ∉ B & b ∈ B then B := B \ {b};
             end
     end;
   copy β into δ;
   a := FIND(v, δ);
   all classes from δ̄ \ {a} replace by their union;
   return δ
 end;
```

Theorem 11. The procedure SAFEPARTITION produces the equivalence δ fulfilling the assumptions of Lemma 10 in time $O(n^2 . G(n))$ if $r = 1$ and $O(n^{r+1})$ if $r \geqq 2$.

Proof. The proof of the time bound is easy since $\mathrm{card}(B)$ decreases in each repetition of the *while*-loop. To prove the correctness, it suffices to prove that after each repetition of the *while*-loop, the following hold:

(a) β is a congruence;

(b) B is a subset of $\bar{\beta}$.

(c) Denoting $a = \mathrm{FIND}(v, \beta)$ and $D = \bar{\beta} \setminus (B \cup \{a\})$ we have:
if $x \in a$ and $y \in \bigcup\{d \mid d \in D\}$ then
(c1) every congruence $\gamma \supseteq \beta$ containing (x, y) contains (v, w).

The proof of (a) and (b) is easy, let us prove (c). We proceed by induction.

At the beginning we have $D = \emptyset$, hence (c) holds. Assume (c) was true at the end of the previous repetition of the *while*-loop and let $B \neq \emptyset$. Then $x \in e \in B$ is chosen and MINCONG(α, v, x, LIST) is executed. If $(v, w) \in \alpha$, then (c1) holds for all (z, y) with $z \in a$ and $y \in e$; (c) holds. If $(v, w) \notin \alpha$, then all unions just formed in α are now repeated in β and, at the same time, B is maintained either by replacing a, b by c, which does not change $\bigcup\{d \mid d \in D\}$, or by deleting a or b from B. In the latter case no new elements are inserted into D and $\bigcup\{d \mid d \in D\}$ can increase only by increasing some classes that have already been in D. Also the class a can increase. But exactly those classes are now in $\beta = \alpha$. Hence condition (c) holds after every repetition of the *while*-loop.

After finishing the *while*-loop, an equivalence δ is produced with exactly two classes a and $\bigcup\{d \mid d \in D\}$. From (c) it follows that after the last repetition of the *while*-loop δ fulfils the assumptions of Lemma 10, which concludes the proof.

The procedure MAXNOTTWO can be described as follows:

Procedure MAXNOTTWO(v, w):
  begin
    δ := SAFEPARTITION(v, w);
    MAXINEQUIV(δ);
    return δ
  end;

Theorem 12. The procedure MAXNOTTWO solves (P2) in time $O(n^2 \,.\, G(n))$ if $r = 1$ and $O(n^{r+1})$ if $r \geqq 2$.

Proof follows immediately from Theorem 11 and Lemma 10.

Corollary 13. The procedure SUBDECOMP solves (P1) in time $O(n^3 \,.\, G(n))$ if $r = 1$ and $O(n^{r+2})$ if $r \geqq 2$.

Finally, let us discuss the solution of (P5). If R is a complement of an equivalence δ, then (P5) coincides with (P3) and is solved by the procedure MAXINEQUIV. If $R = \{(u, v)\}$ then (P5) coincides with (P2) and it is reduced to (P3) using the procedure SAFEPARTITION.

If $0 < \text{card}(R) \leqq k$ for some fixed k then the reduction of (P5) to (P3) can be carried out by a similar method and with the same time bound as in SAFEPARTITION. For this purpose Lemma 10 should be slightly generalized:

Lemma 14. Let $\mathfrak{A} = (A, F)$ be an algebra, let $R \subseteq A^2 \setminus \Delta$, $R \neq \emptyset$. Let β be a congruence and δ an equivalence such that

(i) $R \cap \delta = \emptyset$ and $\beta \subseteq \delta$,

(ii) for every congruence $\gamma \supseteq \beta$ containing $(x, y) \notin \delta$ we have $\gamma \cap R \neq \emptyset$.

Then the greatest congruence ε contained in δ (i.e. the solution of (P3)) is also a greatest congruence disjoint with R (i.e. the solution of (P5)).

```
Procedure SAFEPARTITION1(R):
  begin
    β := Δ;
    R̂ := {x | (x, y) ∈ R or (y, x) ∈ R for some y ∈ A};
    for all (x, y) ∈ R̂² ∖ R do
      begin
        copy β into α;
        MINCONG(α, x, y, LIST);
        if R ∩ α = ∅ then copy α into β
                      else insert (x, y) to R
      end;
    B := {a ∈ β̄ | a ∩ R̂ = ∅};
    C := {a ∈ β̄ | a ∩ R̂ ≠ ∅};
  while B ≠ ∅ do
    begin
      choose e ∈ B and x ∈ e;
      for all a ∈ C do
        begin
          choose y ∈ a;
          copy β into α;
          MINCONG(α, x, y, LIST);
          if α ∩ R = ∅ then
            begin
              for all (a, b, c) ∈ LIST do
                begin
                  UNION(a, b, c, β);
                  if a ∈ B and b ∈ B
                    then B := (B ∖ {a, b}) ∪ {c}
                    else B := B ∖ {a, b};
                  if a ∈ C or b ∈ C
                    then C := (C ∖ {a, b}) ∪ {c}
                end;
              go to S2
            end;
        end;
S2:   B := B ∖ {e}
    end;
  copy β into δ;
  all classes from δ̄ ∖ C replace by their union;
  return δ
end;
```

Theorem 15. The procedure SAFEPARTITION1 produces the equivalence δ fulfilling the assumptions of Lemma 14. If $\mathrm{card}(R) \leqq k$ for some fixed k then it requires time $O(n^2 \,.\, G(n))$ for $r = 1$ and $O(n^{r+1})$ for $r \geqq 2$.

Proof is similar to that of Theorem 11. After each repetition of the *while*-loop the following hold:

(a) β is a congruence;

(b) B, C are disjoint subsets of $\bar{\beta}$;

(c) denoting $D = \bar{\beta} \setminus (B \cup C)$ we have:
if $x \in \bigcup\{c \mid c \in C\}$ and $y \in \bigcup\{d \mid d \in D\}$ then for every congruence $\gamma \supseteq \beta$ if $(x, y) \in \gamma$ then $\gamma \cap R \neq \emptyset$.

Details of the proof are left to the reader.

Having a congruence ε such that $\varepsilon \cap R = \emptyset$ the problem (P5) can be solved directly by the following simple procedure. Note that as ε we can always use congruence Δ.

Procedure MAXNOT(R, ε)
 begin
  $SET := \{\{a, b\} \mid a, b \in \bar{\varepsilon},\ a \neq b,\ (a \times b) \cap R = \emptyset\}$;
  $\delta := \varepsilon$;
  for all $\{a, b\} \in SET$ **do**
   begin
    choose $x \in a$ and $y \in b$;
    copy δ into α;
    MINCONG$(\alpha, x, y, \mathrm{LIST})$;
    if $\alpha \cap R = \emptyset$ **then** copy α into δ
   end;
  $\varepsilon := \delta$
 end;

Theorem 16. Let ε be a congruence such that $\varepsilon \cap R = \emptyset$. Denote $s = \mathrm{card}(SET)$, $p = \mathrm{card}(R)$ and $q = \mathrm{card}(\bar{\varepsilon})$. Then procedure MAXNOT$(R, \varepsilon)$ will produce a solution of (P5) in time $O(q^2p + s(q^r + n + p))$ if $r \geqq 2$ and $O(q^2p + s(q \,.\, G(q) + n + p))$ if $r = 1$.

If we have no previous information concerning congruence ε such that $\varepsilon \cap R = \emptyset$, we may use $\varepsilon = \Delta$. In this case time bound will be $O(n^4 + n^{r+2})$.

(Received August 28, 1981.)

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Massachusetts 1974.

[2] G. Birkhoff: Subdirect unions in universal algebra. Bull. Amer. Math. Soc. *50* (1944), 764–768.

[3] M. Demlová, J. Demel, V. Koubek: On subdirectly irreducible automata. RAIRO Inform. Théor. *15* (1981), 23—46.

[4] M. Demlová, J. Demel, V. Koubek: Several algorithms for finite algebras. In: Fundamentals of Computation Theory 1979 (L. Budach, ed.), Akademie-Verlag, Berlin 1979, 99—104.

[5] M. Demlová, J. Demel, V. Koubek: Algorithms constructing minimal objects in algebras (to appear).

[6] G. Grätzer: Universal Algebra. D. Van Nostrand Company, Princeton, N.J. 1968.

[7] J. Hartmanis, R. E. Stearns: Algebraic Structure Theory of Sequential Machines. Prentice-Hall Inc., Englewood Cliffs, N.J. 1966.

[8] J. E. Hopcroft: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations (Z. Kohavi, A. Paz, eds.), Academic Press, New York 1971, 189—196.

[9] J. E. Hopcroft, R. M. Karp: An Algorithm for Testing the Equivalence of Finite Automata. TR-71-114. Dept. of Computer Science, Cornell University, Ithaca, N.Y. 1971.

[10] R. E. Tarjan: Efficiency of a good but not linear disjoint set union algorithm. J. Assoc. Comput. Mach. *22* (1975), 215—225.

[11] R. E. Tarjan: Reference machines require non-linear time to maintain disjoint sets. Proc. 9th Annual ACM Symposium on Theory of Computing, Boulder, Colorado 1977, 18—29.

RNDr. Jiří Demel, Stavební fakulta ČVUT (Faculty of Civil Engineering — Czech Technical University), Thákurova 7, 166 29 Praha 6. Czechoslovakia.