Regular Issue Brief Papers

Microcode Optimization with Neural Networks

Sunil Bharitkar, Kazuhiro Tsuchiya, and Yoshiyasu Takefuji

Abstract— **Microcode optimization is an NP-complete combinatorial optimization problem. This paper proposes a new method based on the Hopfield neural network for optimizing the wordwidth in the control memory of a microprogrammed digital computer. We present two methodologies, viz., the maximum clique approach, and a cost function based method to minimize an objective function. The maximum clique approach albeit being near $O(1)$ in complexity, is limited in its use for small problem sizes, since it only partitions the data based on the compatibility between the microoperations, and does not minimize the cost function. We thereby use this approach to condition the data initially (to form compatiblity classes), and then use the proposed second method to optimize on the cost function. The latter method is then able to discover *better* solutions than other schemes for the benchmark data set.**

Index Terms—**Microcode optimization, maximum clique, neural networks, NP-complete.**

I. Introduction

MICROCODE optimization is an important problem in designing effecient microprogrammed controllers in a digital system. Typically, the microcode is stored in a control memory (ROM or RAM) of the system. The control signals that are to be activated at a given time is specified by a microinstruction. Each of the microinstructions may have microoperation(s) associated with it which are responsible for the low-level manipulation of the data in the system. For e.g., a microinstruction may be responsible for performing addition, and the individual microoperations may be reading/writing the data into buffers to perform this addition. A collection of microinstructions is called a microprogram. Microprograms may be changed easily by changing the contents of the control memory, thereby demonstrating the flexibility in their use. However, some of the associated disadvantages are [16], extra hardware cost due to the control memory and its access circuitry, performance penalty imposed on accessing the control memory. These disadvantages have discouraged the use of microprogramming in RISC (reduced instruction set) machines, where chip area and circuit delay must both be minimized. Microprogramming continues to be used in CISC's (complex instruction set) such as the Pentium and Motorola 680X0.

Manuscript received February 12, 1998; revised November 17, 1998. This work was supported in part by a grant from Tokyo Electric Power Corporation (TEPCO).

S. Bharitkar is with the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-2564 USA.

K. Tsuchiya is with the System and Software Laboratory, Fuji Electric Corp., 1, Fuji-Town, Hino-City, Tokyo 191, Japan.

Y. Takefuji is with the Department of Environmental Information, Keio University, Endoh, Fujisawa 100, Japan.

Publisher Item Identifier S 1045-9227(99)03996-X.

Microcode optimization is an NP-hard problem [1]. There are several schemes that have been proposed for considering such a problem. The strategies are categorized as [2]: 1) word dimension reduction [3]–[9]; 2) bit dimension reduction [10]–[18]; 3) state reduction [19]; and, 4) heurestic reduction [20], [21].

The best algorithm (in terms of a lower cost function) was proposed by Puri *et al.* [10]. It may be possible to minimize this objective function (cost function) so as to find a better solution. Due to the diverse and successful application of neural networks in solving optimization problems (e.g., [22]–[28]), it was hypothesized that a lower cost could be achieved by such a method.

In this paper we present a Hopfield neural network (HNN) approach for microcode bit reduction. Specifically, we propose two submethods for this problem: neural-network maximum clique (NNMC) [26] and neural-network cost optimizer (NNCO). The NNMC is a near $O(1)$ complexity scheme, and generated optimal solutions for small problem sizes (in terms of the number of microoperations), however, the NNMC did not have an explicit cost function to minimize, hence it failed to give the lowest possible known solutions for large problem sizes (having number of microoperations >17). Since the NNMC partitions the data into compatibility classes (with an insignificant increase in computation time due to increasing problem size), we used this conditioned data as the input to the NNCO method to obtain better solutions for the DEC and IBM microinstruction set.

The organization of this paper is as follows. In Section II, we provide the problem statement and an example. Section III provides some background information in graph theory. In Section IV we provide the relevant background to neural networks. We also present the NNMC and the NNCO for microcode optimization. Section V contains the results. Section VI concludes this paper. The Appendix contains one such discovered solution configuration for the DEC and IBM microinstruction set.

II. Microcode Optimization

Consider Table I having the ROM (read only memory) words (instructions): $W_1, W_2, \cdots, W_5$ with $a, b, \cdots, k$ as subcommands (operations).

In general, the subcommands s_i and s_j are compatible if: $s_i \in W_h \Rightarrow s_j \notin W_h, \forall h$. Thus from Table I, observe that a and g are compatible, while a and b are incompatible.

TABLE I
COMPATIBILITY RELATION BETWEEN SUBCOMMANDS

Word	Subcommand
W_1	a, b, c, d, e, f
W_2	c, g, h, i
W_3	a, b, h, i, j
W_4	d, h, k
W_5	f, h

A compatibility class Γ_i, contains members (subcommands) that are pairwise compatible. For e.g., from Table I, subcommands a and g could belong to this class. A minimal solution to this allocation problem is any set of compatible classes

$$\Delta = [\Gamma_1, \Gamma_2, \cdots, \Gamma_n] \tag{1}$$

such that 1) every subcommand s_i is contained in one compatibility class and 2) the *cost function*

$$L = \sum_{i=1}^{n} \lceil \log_2(|\Gamma_i| + 1) \rceil \tag{2}$$

is minimized. Here, $|\Gamma_i|$ denotes the number of subcommands in class Γ_i. For the example considered in Table I, the following minimal solution $(L = 9, n = 7)$ can be easily verified

$$\Delta = [\{a\}, \{b\}, \{c\}, \{d, g, j\}, \{e, i, k\}, \{f\}, \{h\}]. \tag{3}$$

Thus, given a set of microinstructions (words), the microcode optimization problem is: *partition the subcommands into a set of compatibility classes such that (2) is minimized.*

III. BACKGROUND

Definition 1: A graph $G(V, E)$ consists of nodes (V) and edges (E). We can map the microcode optimization problem described in Section II into a graph theory problem as follows. The subcommands are represented as nodes, and their compatibility being indicated by an edge (a presence of an edge indicating compatibility, and no edge representing incompatibility). We define compatibility between node i and node j by the following mathematical notation:

$$d_{ij} = \begin{cases} 1 \Rightarrow \text{compatiblity} \\ 0 \Rightarrow \text{incompatiblity.} \end{cases} \tag{4}$$

For the microcode example given in Table I, a corresponding graph is shown in Fig. 1.

Definition 2: The density ρ of a graph is given by

$$\rho = \frac{|E|}{\gamma} \tag{5}$$

where

$$\gamma = \frac{n(n-1)}{2}. \tag{6}$$

Here, $|E|$ = number of edges, and n = number of nodes in the graph.

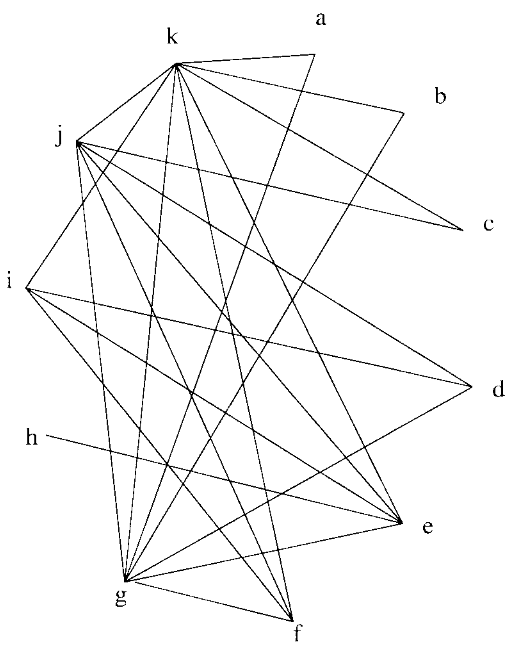

Fig. 1. Compatibility graph for example in Table I.

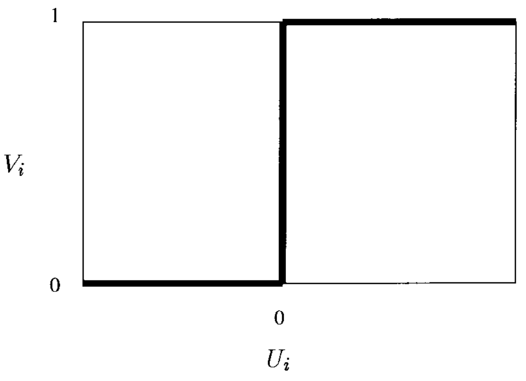

Fig. 2. Binary neuron function.

Definition 3: Let $G(V, E)$, where be an arbitrary undirected graph. Two vertices i and j are called *adjacent* is they are connected by an edge. A *clique* of G is a subset of V, in which every pair of vertices are adjacent. A clique is called *maximal* if it is not a subset of another clique, and the highest cardinality maximal clique is called a *maximum clique*. For, e.g., from Fig. 1, the maximum clique consisits $g - f - j - k$. It has been proven that obtaining a maximum clique from a graph is NP-Complete [29].

IV. THE ALGORITHMS

Before presenting the proposed algorithms (NNMC and NNCO), we digress to provide a brief description of the activation functions used in the model.

The McCulloch–Pitts binary neuron model (Fig. 2) has been widely used in solving optimization problems. It was also used in the NNMC algorithm. The model is stated as follows:

$$V_j = \begin{cases} 1, & U_j \geq 0 \\ 0, & U_j < 0. \end{cases} \tag{7}$$

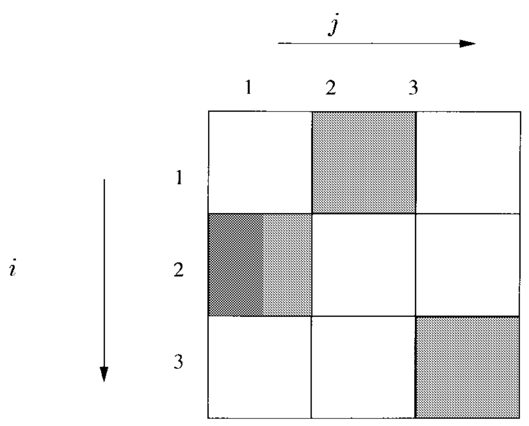

Fig. 3. Maximum neuron function (shaded areas indicate active neurons).

Another type of transfer function is the one-dimensional maximum neuron function. The advantage of using this function is that every converged state corresponds to a feasible (acceptable) solution [30]. The mathematical representation for the one-D maximum function is

$$V_{ij} = \begin{cases} 1, & U_{ij} > U_{ik}; \forall k, k \neq j \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

For, e.g., in Fig. 3, the maximum neurons (i.e., the active neurons) correspond to filled squares in the 3×3 neuron array. This function represents the allocation of subcommands (indicated by the rows) to compatibility classes (indicated by the columns). In other words, when the i, jth neuron is activated, we can state that subcommand i is allocated to compatibility class j.

Finally, both the proposed algorithms use a sequential mechanism for updating a neuron output.

With the relevant information, we now proceed to explain the proposed algorithms for microcode optimization.

A. NNMC with Binary Neuron Function

This method is based on the approach developed by Takefuji *et. al.* [26]. The objective was to maximize the number of vertices of the selected complete subgraph. The dynamical equation for generating a clique of a graph is given by

$$\frac{dU_i}{dt} = -A \sum_{j=1}^{n} (1 - d_{ij})V_j + B\theta \cdot \left(\sum_{j=1}^{n} (1 - d_{ij})V_j + V_i \right) \tag{9}$$

$$V_i = \begin{cases} 1, & U_i \geq 0 \\ 0, & U_i < 0 \end{cases} \tag{10}$$

where, d_{ij} (compatibility between two subcommands) is one if neuron i is connected to neuron j (i.e., the two subcommands, i and j are compatible), zero otherwise; $d_{ii} = 1$ (in graph theory, d_{ij} represents an element of the *adjacency* matrix, i.e., when $d_{ij} = 1$, vertex i is adjacent to vertex j; and, when $d_{ij} = 0$, vertex i is not adjacent to vertex j); n is the number of neurons, and

$$\theta(x) = \begin{cases} 1, & x = 0 \\ 0, & x \neq 0. \end{cases} \tag{11}$$

The first term in (9) discourages neuron (subcommand) i to have a nonzero output if neuron i is not adjacent to neuron j. The second term in (9) encourages neuron i to have a nonzero output if neuron i is adjacent to all the other neurons and the output of neuron i is zero. Because a neuron which has many adjacent neurons is more likely to belong to a clique, the coefficient B is changed by the number of adjacent neurons λ in our algorithm (see [26] for a detailed description) as

$$B = \frac{20\lambda}{n\rho} \quad \text{and} \quad A = 1 \tag{12}$$

where ρ is defined by (5).

The algorithm is described below.

1) Set $n = \text{ProblemSize}, m = 1$.
2) Setup the compatibility graph connections $d_{ij}; i, j = 1, 2, \cdots, n$ depending on the problem.
3) Initialize the input to the neurons using a uniform random number generator, viz., $U_i(0) \in [-1, 1]; i = 1, 2, \cdots, n$.
4) Apply the binary neuron model and determine the output $V_i(0); i = 1, 2, \cdots, n$.
5) Call function GenerateMaximumClique( ), given below.
6) Assign the subcommands obtained from five to compatibility class $\Gamma_m; m = m + 1$.
7) Repeat 5–7 until all subcommands are in the compatibility classes.
8) Compute L from (2).

The GenerateMaximumClique( ) algorithm is as follows.

5.1) Compute:

$$\Delta U_i(k) = -A \sum_{j=1}^{n} (1 - d_{ij})V_j(k) + B\theta \cdot \left(\sum_{j=1}^{n} (1 - d_{ij})V_j(k) + V_i(k) \right). \tag{13}$$

5.2)

$$U_i(k+1) = U_i(k) + \Delta U_i(k). \tag{14}$$

5.3) Compute $V_i(k+1)$ from (7) using (14).

5.4) If $V_i(k) = 1$ and $\Sigma_{j=1}^{n} (1 - d_{ij})V_j(k) = 0$; or, $V_i(k) = 0$ and $\Sigma_{j=1}^{n} (1 - d_{ij})V_j(k) \neq 0, i = 1, 2, \cdots, n$; then terminate this procedure else $k = k + 1$.

The advantage of using this method is a negligible convergence time to the global minimum for small problem sizes (viz., *ex1, ex2, cont1, cont2, dac89, mult*; see Tables II and III for the considered problem sizes). For the other problem sizes, the steady-state solution was always higher than the lowest known cost function. This is due to the fact that the clique algorithm has constraints in the form of compatibility connections, but has no objective cost function to minimize [so as to yield a structure similar to (5)]. Since this algorithm partitions the subcommands in a negligibly small computation time, it was decided that this partitioned information would be

TABLE II
RESULTS

Example	μops	Our algo.		Puri's algo.		Hong's algo.		Nagle's algo.		Baer's algo.	
		L	T	L	T	L	T	L	T	L	T
$ex1$[16]	8	6*	$\approx$	6	0.01	6	0.07	7	0.07	6	0.25
$ex2$[18]	11	9*	$\approx$	9	0.01	9	0.19	9	0.03	9	0.11
$ex3$[15]	25	16*	0.1	16	0.11	17	1.48	19	1.38	16	1.93
$ex4$[11]	32	7*	$\approx$	7	0.01	7	0.34	7	0.47	7	0.4

TABLE III
RESULTS

Example	μops	Our algo.		Puri's algo.		Hong's algo.	
		L	T	L	T	L	T
$cont1$[32]	14	8	$\approx$	8	0.01	9	0.16
$cont2$[32]	14	6	$\approx$	6	0.01	6	0.11
$cont3$[33]	23	8	$\approx$	8	0.01	8	0.36
$cont4$[33]	20	9	$\approx$	9	0.04	9	0.38
$dac89$[13]	14	9	$\approx$	9	0.01	9	0.21
$mult$[34]	17	12	$\approx$	12	0.01	12	0.64
$cpu16$[10]	29	18	0.02	18	0.1	18	2.0
$pdp-9$[14]	36	**22**	**0.08**	23	0.04	23	4.4
$ibm360$[34]	115	**37**	**5.5**	38	1.3	38	105.5

used by a second stage that incorporated the objective function in its motion (dynamical) equation.

This second stage is the NNCO and is explained below.

B. NNCO with Maximum Neuron Function

In this method, the dynamical equation for microcode optimization is given by

$$\frac{dU_{ij}}{dt} = \kappa_{ij}\left[\sum_{s\in\Gamma_j} f(d_{is})\right]\epsilon - L \tag{15}$$

$$V_{ij} = \begin{cases} 1, & U_{ij} > U_{ik}; \forall k, k \neq j \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

$$\kappa_{ij}(\phi) = \begin{cases} 1, & \phi = 0 \\ 0, & \phi > 0 \end{cases} \tag{17}$$

$$f(\tau) = \begin{cases} 0, & \tau = 1 \\ 1, & \tau = 0. \end{cases} \tag{18}$$

In (15), ϵ is a user defined cost function value required to be achieved (e.g., the following values of ϵ were set for $pdp-9$: $\epsilon = 22$, and $ibm360$: $\epsilon = 37$). L is defined by (2).

The indexes i, j are identifiers for the subcommand and compatibility class, respectively. Thus in (15) and (16), U_{ij}, V_{ij} are input(output) to(of) neuron i in compatibility class Γ_j (e.g., when $V_{ij} = 1$, subcommand i is assigned to compatibility class Γ_j).

$\kappa_{ij}(\phi)$ is a complex function of the compatibility d_{ij} [explained after (10) in Section IV-A] between subcommands in different classes.

The effect of (17) and (18) on (15) is explained below.

Observe that, when a subcommand i is compatible with all other subcommands in class Γ_j, (i.e., when $d_{is} = 1, \forall s \in \Gamma_j$), then $\Sigma_{s\in\Gamma_j}\ f(d_{is}) = 0$ using (18). Thus, from (17) see that $\kappa_{ij}[\Sigma_{s\in\Gamma_j}\ f(d_{is})] = 1$. From (15), $dU_{ij}/dt = \epsilon - L = \alpha$. If a subcommand is not compatible with even one subcommand in class Γ_j (i.e, when $d_{is} = 0$ for some $s \in \Gamma_j$), then $\Sigma_{s\in\Gamma_j}\ (d_{is}) > 0$ using (18). Thus, from (17) see that $\kappa_{ij}[\Sigma_{s\in\Gamma_j}\ f(d_{is})] = 0$. From (15), $dU_{ij}/dt = -L = \beta$. Now, $\alpha > \beta$, which means that if a subcommand i is not compatible in class Γ_j (due to the incompatible subcommands in Γ_j), it will have stronger negative bias as compared to the fact if subcommand i was compatible with class Γ_j. This in turn allows a lesser chance of firing (activating basd on the maximum neuron function) of the incompatible neuron i.

Steady state of the system (15) is reached when all conditions of compatibility are satisifed, i.e.,

$$\frac{dU_{ij}}{dt} = 0 \Rightarrow \kappa_{ij}(\phi) = 1; \forall i, j \tag{19}$$

and the steady-state solution is

$$\epsilon = L \tag{20}$$

The algorithm for this method is given below.

1) Obtain the partitioned data from the NNMC method.
2) Set the user defined cost function ϵ (as mentioned previously, this value is set reasonably lower than the best known cost function for the problem).
3) Evaluate $\kappa_{ij}(\phi)$ by hypothetically placing subcommand i in class Γ_j.
4) Evaluate L [from (2)] resulting from this placement.
5) Compute

$$\Delta U_{ij}(k) = \kappa_{ij}\left[\sum_{s\in\Gamma_j} f(d_{is})\right]\epsilon - L. \tag{21}$$

6) Evaluate

$$U_{ij}(k+1) = U_{ij}(k) + \Delta U_{ij}(k). \tag{22}$$

7) Use the maximum neuron model (16), to update the neuron activations. If the i, jth neuron has activated (i.e., if $V_{ij}(k+1) = 1$), truly assign subcommand i to class Γ_j.
8) Increment the iteration step.
9) Repeat Steps 3)–8), until steady state or maximum number of iterations have been reached.

We examined the solution generating ability and behavior using this method. By adjusting ϵ we were able to obtain the best known solutions [10]. Unfortunately, this method demonstrated oscillations (i.e., some solutions repeating in a periodic fashion) for different initializations (i.e., clique solutions obtained from the NNMC method). This was most likely due to the system having converged to some local minimum which it had no means of escape.

Accordingly we modified this method to incorporate a perturbation mechanism known as the omega function [31]. Step 5) in the algorithm was replaced by the following modification,

5) If$(iteration\%10 < \omega)$

$$\Delta U_{ij}(k) = \overbrace{\kappa_{ij}\left[\sum_{s\in\Gamma_j} f(d_{is})\right]\epsilon - L}^{I} + \overbrace{\sum_{l} \Omega\left(\kappa_{il}\left[\sum_{s\in\Gamma_j} f(d_{is})\right]\right)}^{II} \tag{23}$$

else

$$\Delta U_{ij}(k) = \kappa_{ij}\left[\sum_{s\in\Gamma_j} f(d_{is})\right]\epsilon - L \tag{24}$$

where

$$\Omega(z) = \begin{cases} \eta, & z = 1 \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

where % denotes the modulo operator and ω, η are constants. Term II in (23) represents a form of a *bonus* or incentive given to a neuron (subcommand) in proportion to the number of classes it are compatible with, so that it may have a higher chance of being activated.

We experimented with various values for ϵ, η, and ω.

When we reduced the value of ϵ below the cost results given in Tables II and III (e.g., $\epsilon = 12$ for *mult* in Table III), we were unable to obtain convergence to a solution. One possible reason is that the cost obtained in Tables II and III could be optimal (difficult to prove analytically for problems of this form), since the constraints were not satisfied (i.e., $\kappa_{ij}(\phi) \neq 1$) for lower values of ϵ thereby resulting in a nonconvergence to a solution.

As mentioned previously, the effect of ω is to introduce perturbation to move the system out of a local minimum. The value of ω was kept between three and six. Larger values of ω [i.e., using (23) more often than (24)] affect the convergence time to a steady-state solution, since term II introduces an offset to the original differential equation (15). Smaller values of ω do not introduce any significant perturbation to the system.

The value of η was set equal to unity. It can be easily seen that a larger (smaller) value of η results in a larger (smaller) contribution of Term II (compared to Term I) to (23). So, we may look at η as a quantity that adjusts the "intensity" of perturbation. Observing the effects of Term I during experiments, one can then adjust this quantity appropriately.

V. Results

The following notation is used for the Tables II and III, μops is the number of microoperations, L is the minimum cost obtained from the algorithm in bits, $*$ indicates optimized cost, time (T) is in seconds (measured on a SunSparc-20 for the proposed algorithm) and $\approx$ denotes negligible computation time.

In this section, we shall present the results obtained on using the NNMC and NNCO methods. From Table II, we observe that the proposed algorithm performs as good as the other algorithms in finding the *optimal* solutions (with a negligible computation time). From Table III, we see that Puri's algorithm and our algorithm perform equally well for problems with the number of microoperations ranging from 14–29. However, we see that better solutions (i.e., a lower cost function) were obtained for the DEC PDP-9 and IBM 360 microinstruction set examples. One such solution configuration obtained is given in the Appendix.

VI. Conclusion

In this paper, we presented two methods used in conjunction for the microcode optimization problem. The NNMC method (which does not have an explicit cost function in the motion equation) when used separately was not able to find solutions for large problem sizes (as mentioned earlier). However, when coupled with the NNCO method, the resulting algorithm was able to discover better solutions for the DEC and IBM microinstruction set.

However, in the future it may be possible to incorporate an additional cost function term in the maximum clique motion equation (9), thereby possibly eliminating the use of the NNCO method. As an alternative, it could be possible to use the NNCO method by itself (i.e., randomly initializing the subcommand-class allocation, instead of using partitioned data available through the NNMC method).

Appendix

A. Solution Configuration for DEC-PDP9 Microcode

$\Delta = [\{29\}, \{6,13,16,17,30,32\}, \{31\}, \{7,10,14\}, \{28\}, \{12,22,27\}, \{26\}, \{25\}, \{1,2,3,4,5,15,18\}, \{9,21,23,24, 34,35,36\}, \{19,20,33\}, \{11\}, \{8\}]$.

B. Solution Configuration for IBM360 Microcode

$\Delta = [\{84, 40, 93, 94, 72, 73, 81, 83, 107, 88, 89, 90, 91, 92, 41, 95, 96, 97, 98, 99\,100\,101\,102\,103\,108\,109\}, \{50, 17, 18, 19, 20, 21, 22, 23, 24, 25, 85, 29, 31, 32, 51, 52, 76, 28, 47, 53, 54, 55, 56, 57, 86, 77, 78, 48\}, \{106\,110\,111\}, \{60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 16\}, \{114\}, \{104\,105\,112\}, \{79, 80, 74\}, \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}, \{26, 27, 45, 46, 30, 34, 39, 42, 43, 44, 33, 35, 36, 37, 38\}, \{59\}, \{58, 82, 87\}, \{0, 1\}, \{113\}, \{2, 49, 75\}]$.

Acknowledgment

The authors would like to thank Dr. R. Puri from University of Calgary for providing us with the microcode examples, and the anonymous reviewers for their valuable comments.

References

[1] E. L. Robertson, "Microcode bit optimization is NP-complete," *IEEE Trans. Comput.*, vol. C-28, pp. 316–319, Apr. 1979.
[2] T. Agerwala, "Microprogram optimization: A survey," *IEEE Trans. Comput.*, vol. C-25, pp. 962–973, Oct. 1978.
[3] R. L. Kleir and C. V. Ramamoorthy, "Optimization strategies for microprograms," *IEEE Trans. Comput.*, Special Issue on Microprogramming, vol. C-20, pp. 783–794, July 1971.
[4] F. Astopas and K. I. Plukas, "Method of minimizing microprograms," *Automat. Contr.*, vol. 5, no. 4, pp. 10–16, 1971.
[5] C. V. Ramamoorthy and M. Tsuchiya, "A high level language for horizontal microprogramming," *IEEE Trans. Comput.*, vol. C-23, pp. 791–801, Aug. 1974.
[6] S. Dasgupta and J. Tartar, "The identification of maximum parallelism in straight line microprograms," *IEEE Trans. Comput.*, vol. C-25, pp. 986–992, Oct. 1976.
[7] M. Tokoro and E. Tamura, and T. Takizuka, "Optimization of microprograms," *IEEE Trans. Comput.*, vol. C-30, pp. 491–504, July 1981.
[8] S. Davidson, D. Landskov, B. D. Shriver, and P. W. Mallett, "Some experiments in local microcode compaction for horizontal machines," *IEEE Trans. Comput.*, vol. C-30, pp. 460–477, July 1981.
[9] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. C-30, pp. 478–490, July 1981.
[10] R. Puri and J. Gu, "Microword length minimization in microprogrammed controller synthesis," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1449–1457, Oct. 1993.
[11] I. C. Park, S. K. Hong, and C. M. Kyung, "Two complementary approaches for microcode bit optimization," *IEEE Trans. Comput.*, vol. 43, pp. 234–239, Feb. 1994.
[12] A. W. Nagle, R. Cloutier, and A. C. Parker, "Synthesis of hardware control of digital systems," *IEEE Trans. Computer-Aided Dedign*, vol. 1, pp. 201–212, Oct. 1982.
[13] A. Shreshta, P. Kulshreshta, S. Kumar, and S. Ghose, "Automatic synthesis of microprogrammed control units from behaviorial descriptions," in *Proc. 26th ACM/IEEE DAC Conf.*, 1989, pp. 147–154.
[14] C. B. Silio, J. H. Pugsley, and B. A. Jeng "Control memory word width minimization using multiple valued circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 148–153, Feb. 1981.
[15] T. Jayasri and D. Basu, "An approach to organizing microinstructions which minimizes the width of control store words," *IEEE Trans. Comput.*, vol. C-25, pp. 514–521, May 1976.
[16] J. P. Hayes, *Computer Architecture*. New York: McGraw-Hill, 1978.
[17] J. Gu, S. S. Ravi, and S. H. O'Leary "Width minimization of microprograms through simulated annealing," Dept. Comput. Sci., State Univ. New York, Albany, Tech. Rep. TR-88-20, Aug. 1988.
[18] S. J. Schwartz, "An algorithm for minimizing ROM memories for machine control," in *Proc. IEEE 10th Annu. Symp. Switching Automata Theory*, 1968, pp. 28–33.
[19] V. M. Glushkov, "Minimization of microprograms and algorithm schemes," *Kibernetika*, 1966, pp. 1–3.
[20] A. T. Mischenko, "The formal synthesis of an automaton by a microprogram-II," *Kibernetika*, vol. 4, pp. 21–27, 1968.
[21] M. J. Flynn and R. F. Rosin, "Microprogramming: An introduction and viewpoint," *IEEE Trans. Comput.*, vol. C-20, pp. 727–731, July 1971.
[22] J. J. Hopfield and D. W. Tank, "Neural computation of desicions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.
[23] Y. Takefuji and K-C. Lee, "Artificial neural networks for four-coloring and k-colorability problems," *IEEE Trans. Neural Networks*, vol. 38, pp. 326–333, 1991.
[24] ——, "A near optimum parallel planarization algorithm," *Science*, vol. 245, pp. 1221–1223, 1989.
[25] K. Tsuchiya, S. Bharitkar, and Y. Takefuji, "A neural network approach to facility layout problems," *European J. Operational Res.*, vol. 89, pp. 556–563, 1996.
[26] N. Funabiki, Y. Takefuji, and K-C. Lee, "A neural network model for finding a near maximum clique," *J. Parallel and Distributed Comput.*, vol. 14, 340–344, 1992.
[27] B. Kamgar-Parsi and B. Kamgar-Parsi, "On problem solving with Hopfield neural networks," *Biol. Cybern.*, vol. 62, pp. 467–483, 1990.
[28] A. Hemani and A. Postula, "Cell placement by self organization" *Neural Networks*, vol. 3, 377–383, 1990.
[29] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*. New York: Freeman, 1979.
[30] Y. Takefuji, K-C. Lee, and H. Aiso, "An artificial maximum neural network: A winner take all neuron model forcing the state of the system in a solution domain," *Biol. Cybern.*, vol. 67, 243–251, 1992.
[31] Y. Takefuji, *Neural Network Parallel Computing*. Boston: Kluwer, 1992.
[32] M. Andrews, *Principles of Firmware Engineering in Microprogram Control*. Potomac, MD: Computer Sci. Press, 1980.
[33] B. Su, S. Ding, J. Wang, and J. Xia, "Microcode compaction with timing constraints," in *Proc. ACM/IEEE MICRO-20*, 1987, pp. 59–68.
[34] S. Husson, *Microprogramming: Principles and Practices*. Englewood Cliffs, NJ: Prentice-Hall, 1970.