Proceedings of the 5th WSEAS International Conference on Applications of Electrical Engineering, Prague, Czech Republic, March 12-14, 2006 (pp208-212)

Dedicated Hardware for Scheduling Problems using Genetic Algorithm

MASAYA YOSHIKAWA, HIDEKAZU TERAI
Department of VLSI System Design,
Ritsumeikan University
1-1-1, Nojihigashi, Kusatsu, Shiga, 525-8577
JAPAN

Abstract: - NSP (Nurse Scheduling Problem) as a scheduling task consists of assignment of shifts and holidays to nurses for each day on the time horizon, taking into consideration a variety of conflicting interests or objectives between the hospitals and individual nurses. Many works have done for this problem using Genetic Algorithm (GA). The GA is one of the most powerful optimization methods based on the mechanics of natural evolution. However, the problem of the processing time stemming from a population-based search exists in GA. In this paper, we propose a new architecture for high-speed nurse scheduling using GA. The proposed architecture is flexible for many genetic operations on GA. Moreover, the proposed architecture realized not only the pipeline on evaluation phase, but also the pipeline on evolutionary phase on GA. Simulation results evaluating the proposed architecture are shown to the effectiveness comparison with software processing.

Key-Words: - Genetic Algorithm, Dedicated Hardware, Nurse Scheduling Problem, Evolutionary pipeline, Genetic Operation

1 Introduction

Scheduling problems are generally NP-hard combinational problems. NSP (Nurse Scheduling Problem)[1] is one of these scheduling problems. NSP as a scheduling task consists of assignment of shifts and holidays to nurses for each day on the time horizon, taking into consideration a variety of conflicting interests or objectives between the hospitals and individual nurses. Given a number of nurses with specifics skills and working agreements, a contract may consist of general constraints as there are restrictions on the number of nurses for each shift; the maximum number of shifts in a week, a mouth, etc. Moreover, a number of personal wishes or desires representing nurse's preferences are allowed. For example, a demand for the desired day off, demand for doing certain shift on a certain day with a certain nurse, etc[1].

Many works have done for this problem using Genetic Algorithm (GA)[2],[3]. The GA is one of the most powerful optimization methods based on the mechanics of natural evolution. GA has the three operators of selection, crossover, and mutation. However, the problem of the processing time stemming from a population-based search exists in GA.

In this paper, we propose a new architecture for high speed nurse scheduling using GA. Examples of GA hardware design for the purpose of implementing GA in hardware for reduction of computational time are found in Scott[4], Graham and Nelson[5], and Yoshida[6].

Most of these previous works deal with small-scaled problems and limit to some fixed genetic operations. Therefore, it is not suitable for actual industrial problems such as NSP. No previous studies have, to our knowledge, applied GA hardware to NSP, as this study does using the proposed architecture. The proposed architecture is flexible for many genetic operations on GA. Moreover, the proposed architecture realized not only the pipeline on evaluation phase, but also the pipeline on evolutionary phase on GA.

2 Hardware of GA

2.1 Coding

NSP represents a problem to decide a matrix, so that each X_{ij} element of the matrix expresses that nurse i works on day j. X_{ij} consists of day shift, night shift, late-night shift and holiday. Coding of an individual set day shift to “00”, night shift to “01”, late-night shift to “10” and holiday to “11”. Coding in the proposed architecture (hereafter, called HGA: Hardware of GA) is shown in Fig.1. In HGA, the number of nurses is variable, the nurse's minimum number is 16 persons, and the nurse's maximum number is 48 persons.

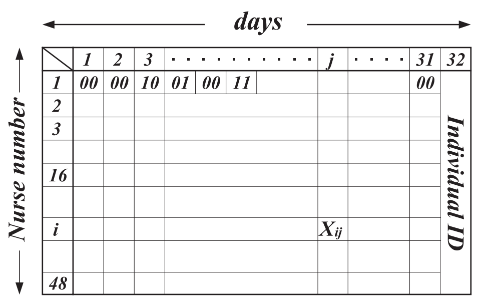


Fig.1: Example of coding

And then, the days, which can be deal with, are 31 days and stored ID of each individual in the column on the 32nd as shown in Fig.1.

2.2 Population

Since GA is the multi-point search algorithm, the memory holding an individual group called population is needed. Furthermore, two memorys are required in order to evolve by GA. One is for the present generation memory holding the present population and the other is for the next-generation memory holding a next-generation population. Memory utilization is the maximum, when there are 48 nurses and the number of individuals is 1024. In this case, the memory of 393216Byte per generation is required. Furthermore, since the present generation and the next generation are totaled, the memory of 786432Byte is required. Therefore, SRAM of 4 bitx512Kword is used as a memory for population in HGA. Fig.2 shows the composition of a memory. A memory address is 19 bits (11bit+8bit), and uses 1 bit of MSB for distinction of the present generation memory and a next-generation memory.

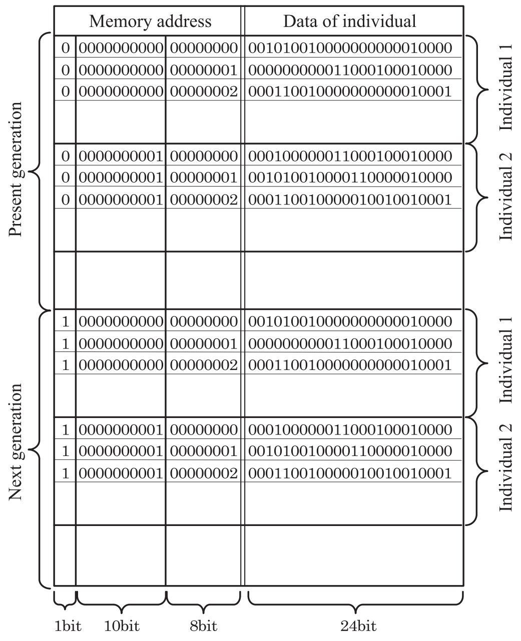
2.3 Evaluation and Selection

2.3.1 Hard constraints

Existing constraints are generally divided into two categories; hard constraints and soft constraints. Table.1 shows the constraints adopted in HGA. Since the solution (schedule) which does not satisfy constraints from (1) to (5) is a lethal gene (meaningless as a schedule), these five constraints are defined as hard constraint. In coding of Fig.1, the hard constraint is constraint of a column to the schedule for one month.

2.3.2 Soft constraints

The constraints from (6) to (15) are defined as soft constraint. The soft constraints are satisfied as much as possible. Therefore, they are introduced to the evaluation function as a penalty. In coding of Fig.1, the soft constraint is constraint of a line to the schedule for one month.



Memory address			Data of individual
0	0000000000	00000000	001010010000000000010000
0	0000000000	00000001	000000000011000100010000
0	0000000000	00000002	000110010000000000010001
0	0000000001	00000000	000100000011000100010000
0	0000000001	00000001	001010010000110000010000
0	0000000001	00000002	000110010000010010010001
1	0000000000	00000000	001010010000000000010000
1	0000000000	00000001	000000000011000100010000
1	0000000000	00000002	000110010000000000010001
1	0000000001	00000000	000100000011000100010000
1	0000000001	00000001	001010010000110000010000
1	0000000001	00000002	000110010000010010010001

Fig.2: Composition of the memory

Table.1: Example of constraints

Classification	Description
Human related limitation	(1) chief nurse is exempted from night shifts (2) do not assign night shifts for unskilled nurses only
Daily restrictions	(3) number of day shift nurses must be > required number of nurses for day shift (4) number of night shift nurses must be = required number of night shift nurses (5) number of late-night shift nurses must be = required number of late-night shift nurses (6) the basic late-night shift pattern is illegal (7) the number of late-night shift is illegal (8) the interval between late-night shift patterns must be at least > one week (9) combination of the day shift and late-night shift is illegal (10) combination of the night shift and late-night shift is illegal (11) combination of the day shift and night shift is illegal (12) request for the holiday when the coming next day is due night shift, is illegal (13) request for the holiday when it is due night shift, is illegal
Month's schedule	(14) number of holidays = number of designated holidays (15) number of late-night shift for all nurse is equal.

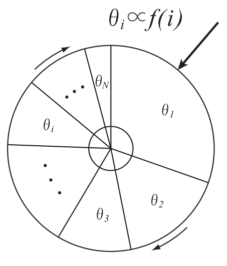
2.3.3 Selection operation

For GA, it's important to set suitable evaluation parameters for controlling the selection of individuals. The selection operations are implemented as roulette wheel selection, ranking selection and elitism.

The roulette wheel selection method uses the percentage represented by the evaluation value for each individual with respect to the sum total of evaluation values for all individuals are the selection probability for each individual. As a result, if the evaluation value for the *i*-th individual is *f(i)*, then a roulette wheel showing this percentage as a wedge can be used, as shown in Fig.3(1).

To put this concretely, let us consider the case of the evaluation values shown in Fig.3(2). First, the product of the individual evaluation values is calculated in alphabetical order, as shown in Fig.3(3). Next, a random number taking a value in the range of "0" to "product of the evaluation values for all individuals" is generated. For Fig.3(3), if the generated random number is from "0" to "149", individual *A* will be selected. And then, if it is from "150" to "199", individual *B* will be selected. The Roulette wheel selection runs in this fashion.

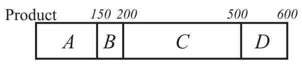
The ranking selection method attempts to preserve descendants using a probability determined for each initial rank assigned to individuals by using the evaluation value. To put this in concrete terms, let us consider a case in which there are four individuals with the evaluation values shown in Fig.3(2), and with the selection probabilities shown in Fig.3(4).



(1) Roulette wheel

individual	A	B	C	D
Evaluation Value	150	50	300	100

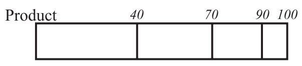
(2) Evaluation values for individuals



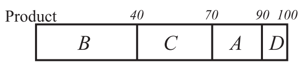
(3) Product of evaluation values

Ranking	1	2	3	4
Selection probability	40%	30%	20%	10%

(4) Selection probabilities for individuals



(5) Product of selection probabilities



(6) Correspondence between selection probability and individual

Fig.3: Example of selection operation

First, the product of the selection probability is calculated in descending rank as shown in Fig.3(5) as a form of preprocessing (performed only once for the first generation). Next, individuals are sorted in descending order for the evaluation value. As is illustrated in Fig.3(6), the sorted individuals correspond to the products of the selection probability in Fig.3(5). Then random numbers are generated in the same fashion as seen in the roulette wheel selection, and the ranking selection is implemented by selecting individuals.

In addition, if a circuit which inherits the individuals with the highest evaluation value unconditionally in the next generation is added, the elitism can be implemented.

2.4 Crossover and mutation

Crossover operations are implemented as one-point crossover, two-points crossover and uniform crossover. These crossovers can be implemented by controlling the mask. One-point crossover will be explained concretely using Fig.4 as an example. In Fig.4, the crossover position is between the third bit and the fourth bit. At this point, before the crossover position, that is, between 1bit and 3bit, the mask is set to "1", and for 4bit and later, the mask is set to "0". Here, when the mask is "0", a normal circuit is used, and when it is "1", a circuit which performs crossovers is used. Uniform crossover is implemented by allocating the 1 and 0 mask randomly. Therefore, the crossover circuit consists of two multiplexer (MUX) and a controller as shown in Fig.5. Mutation operations are implemented as one-point mutation and swap mutation. The mutation is implemented by inverting the genes.

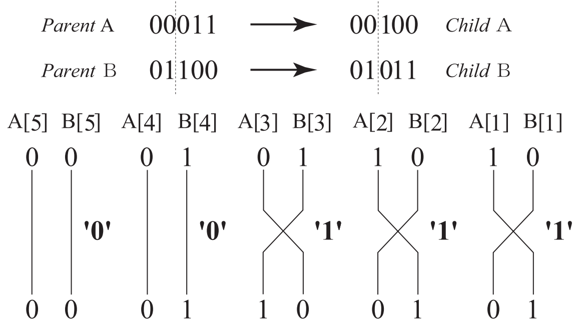


Fig.4: Example of one-point crossover using mask

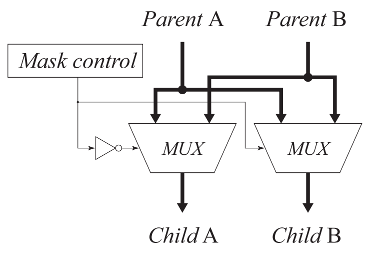


Fig.5: Example of crossover circuit

3 Circuit and Pipeline

Fig.6 shows the block diagram of HGA. The HGA consists of selection circuit, evaluation circuit, preprocessing of evaluation circuit, crossover circuit, mutation circuit, condition of mutation circuit, data path switch and two SRAMs for population (current generation and next generation). The transaction flow of HGA consists of three stages which are read and crossover(RC), mutation (M), evaluation and write (EW). Fig.7 shows pipeline of HGA on evolution phase.

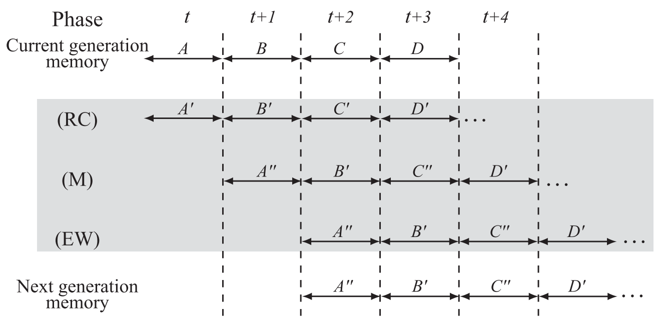


Fig.7: Pipeline on evolution phase

4 Experimental Results

The HGA has been designed by Verilog-HDL and synthesized by the Synplicity Synplify. The frequency of HGA is set up with 33 MHz. Table.2 shows the gate size. In order to evaluate the performance of HGA, we compared with software processing. The software processing was implemented in the C language and run on a Linux PC(CPU: PentiumIV 2.4GHz, Memory 1GHz). Table.3 shows experimental results.

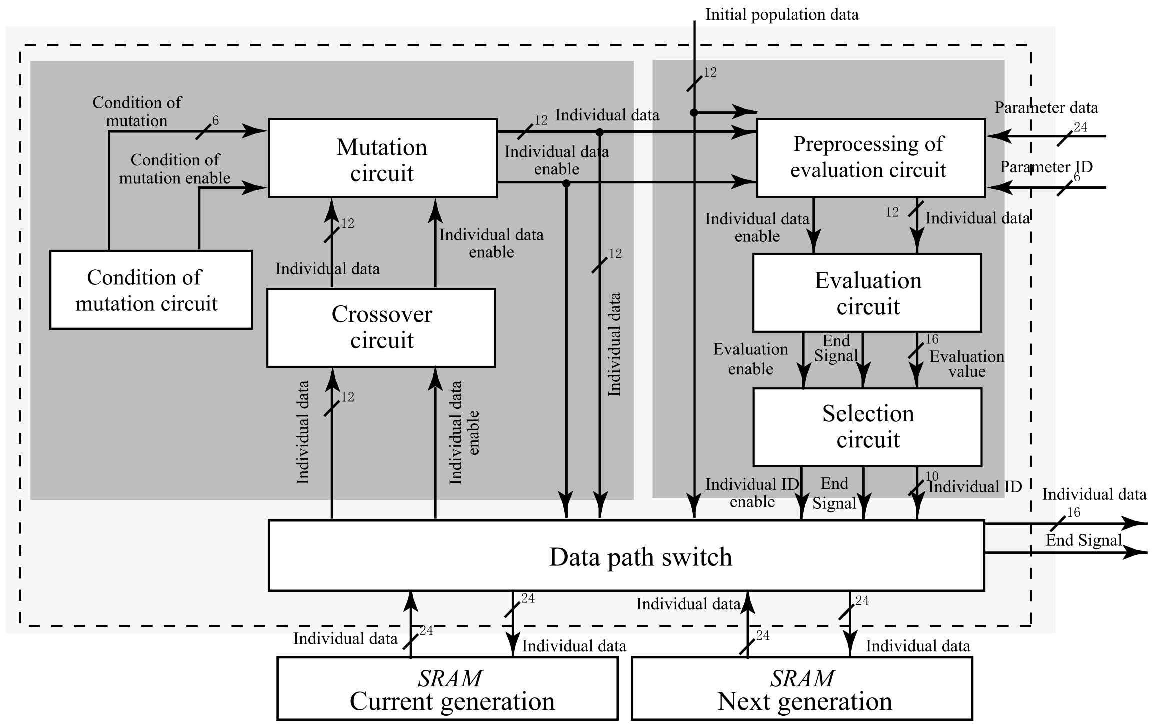


Fig.6: Block diagram

Irrespective of the number of nurses, HGA achieved more than 13.6 times the speed on average as shown in Table.3. In this research, the HGA used the FPGA, which limited the clock speed to 33MHz. However, if the clock frequency can be raised, more high speed processing is possible.

5 Conclusion

We have proposed dedicated engine architecture for Nurse Scheduling Problem and based on Genetic Algorithm. The proposed architecture achieves a high degree of freedom in evolutionary strategy by incorporating most representative GA evolutionary strategies, and architecture features pipelines for genetic operations such fitness evaluation and crossover and an evolutionary pipeline that takes the generation model into account. Experiments comparison with software processing confirmed its feasibility. Our next goal is to develop an algorithm for adaptively selecting the evolutionary algorithm, and to expand hybrid architecture combined with Simulated Annealing. This study was financed in part by a Grant-in-Aid for Young Scientists (B) (Project no.16700153) under the Grant-in-Aid For Scientific Research, extended by the Ministry of Education, Culture, Sports, Science and Technology of Japan. The authors would like to thank their supports.

References:

[1] A.Jan, M.Yamamoto, A.Ohuchi, Evolutionary algorithms for nurse scheduling problem, *Proc. of the Congress on Evolutionary Computation*, Vol.1, pp.196-203, 2000

[2] J.Holland, *Adaptation in Natural Artificial Systems*, the University of Michigan Press (Second edition ; MIT Press), (1992)

[3] Goldberg,D.E, *Genetic algorithms in search optimization*, and machine learning; Addison Wesley,(1989)

[4] S.D.Scott, A.Samal and S.Seth, HGA:A Hardware-Based Genetic Algorithm, *Int. Symposium on Field-Programmable Gate Array*, pp.53-59(1995)

[5] P.Graham and B.Nelson, Genetic Algorithm in Software and in Hardware A performance Analysis of Workstation and custom Computing Machine Implementations, *FPGAs for Custom Computing Machines*, pp.216-225(1996)

[6] N.Yoshida, T.Moriki and T.Yasuoka, GAP:Genetic VLSI Processor for Genetic Algorithms, *Proc.Second Int'l ICSC Symo. On Soft Computing*, pp.341-345(1997-9)

Table.2: Gate size

Module name	Logic Resource	ESBs
Selection circuit	8176/16640(49%)	34/104(32%)
Crossover circuit	357/16640(2%)	2/104(1%)
Mutation circuit	1160/16640(6%)	2/104(1%)
Condition of mutation circuit	78/16640(0%)	0/104(0%)
Evaluation circuit	2557/16640(15%)	0/104(0%)
Preprocessing of evaluation circuit	940/16640(5%)	0/104(0%)
Data path switch	1883/16640(11%)	0/104(0%)

Table.3: Experimental results

(1)The number of nurses is 16

# individual	HGA(s)	Software(s)	Speed(times)
128	3.691	53.72	14.5
256	7.346	103.7	14.1
512	14.684	204.04	13.8
1024	29.360	412.385	14.0

(2)The number of nurses is 32

# individual	HGA(s)	Software(s)	Speed(times)
128	7.347	98.31	13.9
256	14.628	190.77	13.0
512	29.191	372.57	12.7
1024	58.375	738.445	12.6