Nurse Scheduling Problem using Approximation Algorithms with Cost Bit Matrix

Young-Woong Ko,Saangyong Uhmn, Jin Kim

Department of Computer Engineering, Hallym University, Chuncheon, Gangwondo, 200-702 Republic of Korea
jinkim@hallym.ac.kr

Abstract: We applied simulated annealing and genetic algorithm to nurse scheduling problem. For time complexity problem of these algorithms, we suggested efficient operators using a matrix called a cost bit matrix. Each cell in the matrix indicates any violation of constraints. A cell with 1 indicates that the corresponding assignment violates any of constraints and needs further consideration. The experimental results showed that both algorithms with the suggested operators generated a nurse scheduling faster in time and better in quality compared to those without the matrix. In addition, simulated annealing with the matrix performed better than the corresponding implementation of genetic algorithm.

[Ko, Y.-W., Uhmn S., Kim J. **Nurse Scheduling Problem using Simulated Annealing and Genetic Algorithm with Cost Bit Matrix**. *Life Sci J* 2014;11(7):689-693] (ISSN:1097-8135). http://www.lifesciencesite.com. 99

Keywords: nurse scheduling problem (NSP), simulated annealing, genetic algorithm, cost bit matrix

1. Introduction

Any organization providing a round-the-clock service divides its daily work into consecutive shifts each of which is a period of time for a group of employees on duty. Each employee is assigned to a set of shifts, which must satisfy several constraints that may be set up by staffing requirements, rules by the administration and labor contract clauses.

Nurse scheduling problem (NSP) is an instance of a scheduling problem in which each nurse is assigned to a set of shifts and rest days in a timetable called a nurse roster (Ender, 2005; Ernst et al., 2004; Burke et al., 2006, Cheang et al., 2003, Bard and Purnomo, 2007). It was proven to be NP-hard even with only a few of real world constraints (Osgogami and Imai, 2000). Miller et al. and Warner et al. formulated NSP as the selection of a timetable (Miller et al., 1976; Warner and Prawda, 1972). They simplified the problem to include too small or ignore too many constraints to be practical. Jan et al. and Aickelin et al. applied genetic algorithms (GA) to NSP (Jan et al., 2000, Aickelin and Dawsland, 2004). Kundu et al. applied genetic algorithm and simulated annealing (SA) to the same problem instances and compared their performances with others (Kundu et al., 2008).

Because NSP may include many constraints and there can be several different instances with different set of constraints, the problem instance must be defined clearly. In this study, we consider a cyclic nurse scheduling problem with following constraints as in (Kundu et al., 2008). An instance includes three components (1) the preference of each nurse as an aversion to particular days and shifts, (2) minimal coverage constraint of the number of nurses per shift and per day, (3) case-specific constraint of personal time requirements, specific workplace conditions, and so on. The objective of this problem is to satisfy nurses' requests as much as possible while fulfilling the employers' concerns.

In this paper, we applied simulated annealing and genetic algorithm to NSP and compared their performance in time and quality of solutions. In addition to generic form of them, we implemented modified versions with so-called a cost bit matrix for time complexity of those algorithms. In the next section, we will briefly introduce NSP and a cost function and in section 3 a cost bit matrix and operators in simulated annealing and genetic algorithm will be given. Section 4 will provide experimental results. Finally, conclusions and further work are discussed in section 5.

2. Problem Description

2.1 Nurse Scheduling Problem

NSP is to create weekly or monthly schedules for N nurses that must satisfy several constraints set by labor contracts and administrative requirements. Therefore, NSP is essentially a scheduling problem that satisfies a number of constraints. Constraints are usually classified under two categories: soft and hard constraints. Hard constraints should be always satisfied in any schedule while soft constraints can be violated. A schedule that does not satisfy any of hard constraints cannot be a feasible one. Possible examples include restrictions on the number of nurses for each shift and the maximum number of shifts in a week or a month. Soft constraints can be violated but as minimal as possible. In other words, the soft constraints are expected to be satisfied, but violation does not make it an infeasible solution. Some examples are requests for a desired day off or a certain shift on a certain day. Generally, there are three shifts, morning, evening, and night, and an off-day.

There are various kinds of hard and soft constraints we may consider. Because the main objective of this study is to provide an improvement for SA and GA and compare their performance, we confined the constraints as follows.

(a) Hard constraints

(i) Number of nurses for each working shift per day. The number of nurses for morning, evening, and night shift should be between the minimum and the maximum values.

(ii) Working patterns. Morning after night shift, evening after night, morning after evening shift and three consecutive night shifts are not allowed.

(b) Soft constraints

(i) Total number of off-days (*o*), night (*n*), morning (*m*) and evening (*e*) shifts during the certain period of days for each nurse.

2.2 Cost Function

We have to define a cost function for NSP to optimize. Let *N* and *D* be number of nurses and days, and *s* be one of the three shifts or an off-day. Then, NSP may be represented as a problem to decide an $N \times D$ matrix, *X*, whose element x_{ij} represents that nurse *i* works on day *j* where $x_{ij} = \{m, e, n, o\}$. We define m_j, e_j, n_j as total number of nurses for morning, evening, and night shift on day *j*. These numbers must be between the minimum and the maximum number of nurses for each shift, m_{min}, e_{min}, n_{min}, m_{max}, e_{max}, and n_{max}. We also define M_i, E_i and N_i as total number of each shift, morning, evening, and night of a nurse *i* and their requirements M_{req}, E_{req} and N_{req}. We can define three costs for each requirement as follows.

$$C_1 = \sum_{j=1}^{D} (cm_j + ce_j + cn_j)$$

$$C_2 = \sum_{i=1}^{N} CM_i + CE_i + CN_i)$$

$$C_3 = \sum_{i=1}^{N} \sum_{j=1}^{D} cp_{ij}$$

where

CM_i : Cost for M_i. 0 if $M_i = M_{req}$ else 1.

CE_i : Cost for E_i. 0 if $E_i = E_{req}$ else 1.

CN_i : Cost for N_i . 0 if $N_i = N_{req}$ else 1.

cm_j : Cost for m_j. 0 if $m_{min} \le m_j \le mmax$ else 1.

ce_j : Cost for e_j . 0 if $e_{min} \le n_j \le e_{max}$ else 1.

cn_j : Cost for n_j . 0 if $n_{min} \le n_j \le n_{max}$ else 1.

cp_{ij} : Cost for cyclic working pattern of nurse *i*.

0 if $x_{ij\text{-}1}x_{ij} \in \{ne, nn, nm\}$ else 1.

Based on these costs, we can define an objective cost function as follows.

$$C_{total} = C_1 * w_1 + C_2 * w_2 + C_3 * w_3$$

where w_1, w_2, and w_3 are weight values for the cost C_1, C_2 and C_3.

Our goal is to minimize the cost function C_{total} so as to find an optimal nurse schedule. The simplest method is a brute force approach which evaluates all possible nurse schedules. It guarantees a feasible schedule with the minimum cost. However, the number of all possible nurse schedules is $4^{N \times D}$. If *N* and *D* increase, this approach is intractable. This is a class of problems called *NP-hard* (Papadimitrioud, 1993), which means an algorithm that guarantees to find an optimal solution with the size of *N* and *D* in reasonable time may not exist. To overcome this problem, we applied couple of approximation algorithms: simulated annealing and genetic algorithm.

3. Cost bit matrix

Because of page limit, we only provide a brief description of a cost bit matrix and its application: simulated annealing with a cost bit matrix (CMSA) and genetic algorithm with a cost bit matrix (CMGA). Traditional SA (TSA) and GA (TGA) can be found in (Kirkpatric and Gelatt Jr, 1983) and (Goldberg, 1989).

3.1 Description

A cost bit matrix *V* is an $N \times D$ matrix whose each cell is set if any constraint is violated. Table 1 depicted a sample schedule and its corresponding cost bit matrix. The numbers in the first column means nurses. The numbers of each shift and off-day for nurse *i* is represented as m_i, e_i, n_i, and o_i and the numbers of nurses for each shift and off day of day *j* is represented as m_j, e_j, n_j, and o_j.

Table 1. A sample schedule and its corresponding cost bit matrix when m_{min}=m_{max}=2, e_{min}=e_{max}=1, n_{min}=n_{max}=1, M_{req}=2, E_{req}=2, N_{req}=2, O_{req}=1.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun	m_i	e_i	n_i	o_i
1	*n*	*n*	*o*	*m*	*m*	*e*	*e*	2	2	2	1
2	*e*	*e*	*m*	*o*	*m*	*n*	*n*	2	2	2	1
3	*m*	*m*	*e*	*e*	*n*	*n*	*o*	2	2	2	1
4	*o*	*m*	*m*	*n*	*e*	*e*	*n*	2	2	2	1
5	*m*	*o*	*n*	*m*	*n*	*e*	*e*	2	2	2	1
m_j	2	2	2	2	2	0	0				
e_j	1	1	1	1	1	3	2				
n_j	1	1	1	1	2	2	2				
o_j	1	1	1	1	1	0	1				

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	0	0	0	0	1	1	1
2	0	0	0	0	1	1	1
3	0	0	0	0	1	1	1
4	0	0	0	0	1	1	1
5	0	0	0	0	1	1	1

When calculating C_1, C_2, and C_3, the cost bit matrix is set as follows: If CM_i, CE_i, or CN_i is not 0, all cells on *i*-th row of the matrix are set to 1 or 0 otherwise; if cm_j, ce_j, or ne_j is not 0, all cells on *j*-th column are set to 1 or 0 otherwise; if cp_{ij} is 1, v_{ij} is set to 1 or 0 otherwise. Figure 1 depicted pseudo code for the costs. For example, the assignments of Friday, Saturday and Sunday violating the constraints make the corresponding columns of the cost bit matrix be set to 1 in Table 1.

3.2 Application

In CMSA and CMGA, the cost function C_{total} is evaluated whenever a new schedule X_{new} is generated from the current schedule X_{cur} and the cells of the matrix v_{ij} is set to1 if any constraint is violated or 0 otherwise. When a new schedule is generated, v_{ij} of the matrix V is used to determine if x_{ij} of X_{new} can be changed or not.

3.2.1 Simulated annealing

```
C1()
{
  int $c_1$=0;
  for(j=1; j<=$N$; j++)
  {
    if (($m_j$<$m_{min}$)||($m_j$>$m_{max}$)){$c_1$=$c_1$+1;}
    if (($e_j$<$e_{min}$)||($e_j$>$e_{max}$)){$c_1$=$c_1$+1;}
    if (($n_j$<$n_{min}$)||($n_j$>$n_{max}$)){$c_1$=$c_1$+1;}
    if ($c_1$ != 0)
      for(i=1; i<=$D$; i++)
        $v_{ij}$=1;
  }
  return $c_1$;
}

C2()
{
  int $c_2$;
  for(i=1;i<=$D$;j++)
  {
    if ($M_i$!=$M_{req}$) $c_2$=$c_2$+1;
    if ($E_i$!=$E_{req}$) $c_2$=$c_2$+1;
    if ($N_i$!=$N_{req}$) $c_2$=$c_2$+1;
    if ($O_i$!=$O_{req}$) $c_2$=$c_2$+1;
    if ($c_2$ != 0)
      for(j=1; j<=$N$; j++)
        $v_{ij}$=1;
  }
  return $c_2$;
}

C3()
{
  int $c_3$=0;
  for (j=1; j<=$D$; j++)
  {
```

Figure 1. Pseudo code for three costs, C_1, C_2, and C_3.

```
      if (($x_{ij-1}$==n)&&($x_{ij}$==e)) $c_3$=$c_3$+1;
      if (($x_{ij-1}$==e)&&($x_{ij}$==m)) $c_3$=c$_3$+1;
      if (($x_{ij-2}$==n)&&($x_{ij-1}$==n)&&($x_{ij}$==n))
        $c_3$=$c_3$+1;
      if ($c_3$ != 0) $v_{ij}$=1;
    }
  }
  return $c_3$;
}
```

Figure 1. Simulated annealing

SA is a probabilistic approach that can be used to find a global optimum of a function for combinatorial optimization problems. To use this algorithm, a set of state $S=\{s_1, ..., s_n\}$ and a cost function $C:S\rightarrow R$, where R is the set of real numbers, should be defined. A real value $C(S)$ should be assigned to each state $s\in S$. The goal of the optimization problem is to find an optimal state s_{opt} whose score is min(max)$\{s_i|1 \leq i \leq n \}$. SA continuously generates a new candidate state s_{new} from a current state $s_{current}$ by applying transition rules and acceptance rules. The criteria of the acceptance rules are: (a) if $\Delta E \leq 0$, accept a new state s_{new} (b) if accept a new state s_{new} with probability $P(\Delta E) = e^{-\frac{\Delta E}{T}}$ where T is a temperature and $\Delta E = C(S_{new}) - C(S_{current})$ is a cost difference. Probability $P(\Delta E)$ prevents the system from fixation at a local minimum. A state $s_{current}$ is called a local minimum if there is no new state s_{new} in S that is generated from $s_{current}$ by applying the transition rules and has a lower cost than that of $s_{current}$.

Temperature T controls a probability to accept a new state s_{new}. Initially, T starts from a high temperature and after each iteration T decreases based on an annealing schedule and becomes zero eventually. The probability of accepting a new state with a higher cost than that of the current also decreases as temperature T decreases. If a careful annealing schedule and a number of iterations are given, SA converges to a global minimum state s_{opt}. Because of efficient performance by this characteristic, SA has been applied to many combinatorial problems

In CMSA, a new schedule is generated by applying a transition rule to X_{cur}. The cost bit matrix is used to determine whether a transition rule is applied or not. If v_{ij} is 1, a transition rule is applied to the corresponding assignment x_{ij} in X_{cur}. In Figure 2, pseudo code for a transition function is given.

```
Transition(){
  for (i=1; i<=$N$; i++){
    for (j=1; j<=$D$; j++){
      if ((rand()%100<p)&&($v_{ij}$==1)
        $x_{ij}$=random(d, e, n, o);
    }
  }
`
```

Figure 2. A transition function for CMSA

3.2.2 Genetic algorithm

GA is a search algorithm to simulate the process of natural selection. GA starts with the set of potential solutions called a population and evolves toward more optimal solutions. The solutions are evaluated by a fitness function. The fitness value represents the quality measure of a solution so that the algorithm can use it to select ones with better genetic material for producing new solutions and further generations. This simulation of evolution allows survival of better solutions and extinction of inferior ones. The goal is to find better solutions in each generation. The process of evolution is carried out by selection, crossover and mutation. In terms of GA, those processes are called genetic operators. The selection chooses superior solutions in every generation and assures that inferior solutions are extinct. The crossover operator chooses two solutions from current population and generates a new solution based on their genetic material. Selection and crossover operators will expand good features of superior individuals through the whole population. They will also direct the search process towards a local optimum. The mutation operator changes the value of some genes in a solution and helps to search other parts of problem space.

After selection and crossover in CMGA, two new schedules can be obtained. With mutation probability P_m, the two schedules can be mutated. In this study, a cost bit matrix is used to determine whether a mutation operator is applied to these schedules. A mutation operator is applied to the cell x_{ij} in a schedule with cell change probability P_{cc} only if the corresponding cell v_{ij}=1. The mutation operator is presented in Figure 3.

```
Mutation()
{
  for (i=1; i<=N; i++)
  {
    for(j=1; j<=D; j++)
    {
      if ((rand()%100<p)&&(v_ij==1))
        x_ij=random(d, e, n, o);
    }
  }
}
```

Figure 1. A mutation operator for CMGA

4. Experimental results

We implemented TSA, CMSA, TGA, and CMGA in C and ran on a PC with an Intel® Core(TM) i5-2520M 2.5 Ghz CPU and 4GB of memory. All the constraints described in this paper were included in all implementations with exact same conditions. The goal was to check whether the proposed algorithms, CMSA and CMGA, could actually generate an acceptable NSP efficiently and compare them with the corresponding traditional algorithms. The random number generator `rand()` was used when necessary. Each set of instances is consisted of 100 problems generated randomly. The number of nurses, N, is 15 and the number of weeks, D, is from one to four. The weights for cost functions are w_1=5, w_2=5 and w_3=1, respectively. Hard constraints are same for all the problems ($d_{\min}$=4, $d_{\max}$=6, $e_{\min}$=3, e_{max}=5, $n_{\min}$=3, $n_{\max}$=5) and soft constraints for one week are D_{req}=2, E_{req}=2, N_{req}=2, and O_{req}=1, and are proportional to the number of weeks for two to four weeks. The crossover probability P_c=0.03, mutation probability P_m=0.01 and cell change probability p_{cc}=0.01 were applied.

The TSA and CMSA were compared on the bases of three criteria: the average cost C_{final} of the obtained schedules, the average number of iterations to reach the final schedule, and execution time T_{final}. The TGA and CMGA were compared on the following values: the average cost C_{final} of the obtained schedules, the average number of generations to reach the final schedule, and execution time T_{final}.

In Table 2 and 3, the experimental results of the algorithms were given.

Table 2. Performance results of TSA and CMSA.

Week	Method	C_{final}	Iterations		T_{final}
			to C_{final}	Total	(sec)
1	TSA	3.6	511,141	1×10^6	19.1
	CMSA	0.0	221,806	1×10^6	2.9
2	TSA	7.4	2,677,684	5×10^6	66.3
	CMSA	0.0	2,062,308	5×10^6	35.4
3	TSA	10.4	1,1236,294	20×10^6	393.8
	CMSA	0.0	1,0120,325	20×10^6	246.3
4	TSA	13.8	5,7878,176	100×10^6	2845.2
	CMSA	0.0	4,8276,960	100×10^6	1472.4

Table 3. Performance results of TGA and CMGA.

Week	Method	C_{final}	Generations to C_{final}	Populations	T_{final} (sec)
1	TGA	8.8	2.33×10^5	100	81.7
	CMGA	7.6	2.21×10^5	100	13.3
2	TGA	9.4	3.77×10^5	400	750.3
	CMGA	6.4	4.22×10^5	400	516.6
3	TGA	15.8	4.40×10^5	1,000	4,292.2
	CMGA	8.8	4.40×10^5	1,000	4,207.3
4	TGA	24.0	5.50×10^5	3,000	21,104.0
	CMGA	15.6	5.00×10^5	3,000	15,492.0

As seen in Table 2, CMSA outperformed TSA in all criteria. While CMSA provided a schedule with C_{final}=0 for all weeks, TSA didn't. As the number of weeks is increased, the difference between these two methods in every criterion became larger. However, TSA and CMSA showed slightly different results in Table 3. Both of TGA and CMGA failed to find a schedule with C_{final}=0 for all weeks even though they executed much longer in time than TSA and CMSA. Unlike the number of required iterations in TSA and CMSA, the numbers of generations for TGA and CMGA are very similar or almost identical. In addition, the differences in execution time T_{final} between TGA and CMGA were not as large as those between TSA and CMSA.

We applied simple crossover, multi-point crossover and uniform crossover in our GA and the best one was simple crossover. CMGA was very effective compared to TGA based on the above comparison, while CMSA was the best. In all the qualities of the solutions, CMSA was very impressive because of its powerful operators with a cost bit matrix.

5. Conclusion and future works

In this paper, we applied SA and GA to NSP and proposed a strategy to improve performance of them. In CMSA and CMGA, a cost bit matrix was used to generate a new schedule efficiently, which was justified by the experimental results. In CMSA, a transition rule was applied based on the matrix. In CMGA, the selection and crossover operators were applied based on the probability only, while the mutation operator was applied based on the probability and the values in the matrix. The usage of the matrix resulted in pruning of search space that was the main cause of reduction in execution time. In addition, possibility to find feasible solutions in CMSA was increased, which made our algorithms find solutions satisfying all the constraints. These approaches, CMSA and CMGA, generated a nurse schedule faster in speed and better in quality than traditional ones, respectively. And SA and CMSA outperformed the corresponding version of GA respectively.

Although we have presented this work in terms of nurse scheduling, it should be noticed that the main idea of the approach could be applied to many other scheduling problems.

Future research aims at parallelization of the algorithms by utilizing state-of-the-art GPGPUs.

Acknowledgements:

This research was supported by Hallym University Research Fund, 2013(HRF-201309-017).

Corresponding Author:
Dr. Jin Kim
Department of Computer Engineering
Hallym University
Chuncheon, Gangwondo 200-702 Republic of Korea
E-mail: jinkim@hallym.ac.kr

References

1. Ender O. Memetic algorithms for nurse rostering. Lecture notes in computer science, In: The 20th International Symposium on Computer and Information Sciences, Springer-Verlag 2005; 482-492.
2. Ernst AT, Jiang H, Krishamoorty M, Owens B, Sier D. An annotated bibliography of personnel scheduling and rostering. Annals of Operations Research 2004;127(1):21-144.
3. Burke EK, De Causmaecker P, Petrovic S, Vanden Berghe G. Meta-heuristics for handling time interval coverage constraints in nurse scheduling. Applied Artificial Intelligence 2006;20(9): 743-766.
4. Cheang B, Li H, Lim A, Rodrigues B. Nurse rostering problems-A bibliographic survey. European Journal of Operational Research 2003;151(3):447-460.
5. Bard JF, Purnomo HW. Cyclic preference scheduling of nurses using a Lagrangian based heuristic. Journal of Scheduling 2007;10(1):5-23.
6. Osogami T, Imai H. Classification of Various Neighborhood Operations for the Nurse Scheduling Problem. In: ISAAC '00: Proceedings of the 11th International Conference on Algorithms and Computation, Springer-Verlag 2007;72-83.
7. Miller HE, Pierskalla WP, Rath GJ. Nurse Scheduling using Mathematical Programming. Operations Research 1976;24(5):857-870.
8. Warner DM, Prawda J. A Mathematical Programming Model for Scheduling Nursing Personnel in a Hospital. Management Science 1972;19(4-Part-1):411-422.
9. Jan A, Yamamoto M, Ohuchi A. Evolutionary Algorithms for Nurse Scheduling Problem. In: Proc. The 2000 Congress on Evolutionary Computation, 2000;196-203.
10. Aickelin U, Dowsland KA. An indirect Genetic Algorithm for a Nurse-Scheduling Problem. Computers & Operations Research 2004;31(5): 761-778.
11. Kundu S, Mahato M, Mahanty B, Acharyya S. Comparative Performance of Simulated Annealing and Genetic Algorithm in Solving Nurse Scheduling Problem. In: Proc. Int'l Multi Conference of Engineers and Computer Scientists 2008;1-5.
12. Kirkpatrick S, Gelatt Jr CD, Vecchi MP. Optimization by Simulated Annealing. Science 1983;220(4598):671–680.
13. Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, 1989.
14. Papadimitriou CH. Computational Complexity, Addison Wesley, 1993.

5/26/2014