An Improvement Technique for Simulated Annealing and Its Application to Nurse Scheduling Problem

Young-Woong Ko, DongHoii Kim, Minyeong Jeong, Wooram Jeon, Saangyong Uhm and Jin Kim*

Dept. of Computer Engineering
Hallym University
Chuncheon, Gangwondo 200-702 Republic of Korea
{yuko,kdh,jmy,jwr,suhmn,jinkim}@hallym.ac.kr

Abstract

The simulated annealing was perceived as a useful method for many intractable problems. However, it needs additional strategies to cope with time complexity due to an initial state and search space reduction. In this work, we suggested an efficient transition rule and applied it to a nurse scheduling problem. It uses a cost matrix to reduce a set of candidates, which results in performance improvement. The experimental results showed that the suggested method generated a feasible solution for the nurse scheduling problem faster in time and better in quality compared to traditional simulated annealing.

Keywords: nurse scheduling problem, simulated annealing, cost matrix

1. Introduction

Daily work in any organization providing 24/7 services is divided into shifts. Shifts are defined as periods of time during which different groups of employees are in the line of duty. In such organizations, employees are assigned into a set of shifts in such a way that several constraints are satisfied. The constraints may be set up by staffing requirements, the rules by the administration and labor contract clauses.

In nurse scheduling problem (NSP), nurses are employed to be assigned into a set of shifts and rest days in a timetable called nurse roster. Each shift must meet these constraints. It was proven to be NP-hard even if it was with only a subset of real world constraints [1].

In early days, the problem was formulated as a selection of a timetable that minimized an objective function that balanced the trade-off between staffing coverage and nurses’ schedule preferences [2, 3]. They focused on the simplified versions of the problem which were too small or ignored too many constraints to be practical. Abdannadher et al., tackled the problem using the Constraint Logic Program (CLP) framework [4]. Jan et al. and Aickelin et al., applied genetic algorithm (GA) [5, 6] and Li et al., employed Bayesian optimization algorithm [7]. Kundu et al., applied genetic algorithm and simulated annealing (SA) and compared their performances with others [8]. The nurse scheduling problems covered in these researches are categorized by certain criteria [9].

In this paper, we solved the cyclic nurse scheduling problem with the following components [8]: Firstly, the preference of each nurse on a day and a shift. Secondly, minimal coverage constraints of required nurses per shift and per day like any shift scheduling problem. Thirdly, case-specific constraints by personal time requirements,
specific workplace conditions, etc. Hence, the objective of the NSP is to satisfy nurses' requests as much as possible while fulfilling the employers' concerns.

In this paper, we suggest simulated annealing with a cost matrix (CMSA) to find a schedule that optimizes a cost function and satisfies a set of constraints. In the next section, we will briefly introduce NSP and its cost function and in Section 3, SA and the cost matrix and the transition rule in CMSA. Section 4 is concerned with experiments and results. Finally, conclusion 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 by assigning one out of a number of possible shift patterns to each nurse. These schedules have to satisfy working contracts and meet demands for a number of nurses of different grades on each shift, while being seen to be fair by the staff concerned. Therefore NSP is essentially a scheduling problem which suits a number of constraints. Constraints are usually classified as soft or hard constraints. Hard constraints must be satisfied fully in any working schedule and do not allow any violation. That is, a schedule which does not satisfy hard constraints cannot be a feasible solution. Possible examples include restrictions on the number of nurses for each shift, maximum number of shifts in a week or a month, etc. Soft constraints may be broken, but of which breaches must be minimized. In other words, soft constraints are desired to be satisfied, but violation does not make it an infeasible solution. Some examples are demands for a desired day-off or doing certain shift on a certain day with a certain nurses, etc. Generally, every nurse works on three shifts, morning, evening, and night, and rests for some holidays.

There are various kinds of hard and soft constraints we have to consider. However, because the main objective of this study is to show fast and efficient SA approach, we confined constraints as follows.

(a) Hard constraints

(i) There are constraints on the number of nurses for each working shift per day. The number of nurses for morning, evening, and night shift should be within a range between the maximum and the minimum values.

(ii) There are constraints for the working patterns. Morning after night shift, evening after night, morning after evening shift and three consecutive night shifts should be avoided.

(b) Soft constraints

There are constraints for the total number of off-days (o), night (n), morning (m) and evening (e) shifts during a certain periods for each nurse.

2.2. Cost Function

We have to define a cost function to evaluate schedules for each nurse. Let N and D be the number of nurses and days, and s be one of the three shifts or an off-day required to be scheduled. Then, NSP is represented as a problem to decide an $N \times D$ matrix, so that each element $x_{ij}$ of the matrix express that nurse $i$ works on day $j$ where $x_{ij} = \{m, e, n, o\}$. 


(a) To evaluate the violation of hard constraint (i), we define \( m_j, e_j, n_j \) be the total number of nurses for morning, evening, and night shift on day \( j \). If any of these numbers are not between minimum and maximum number of each shift \( (m_{\text{min}}, m_{\text{max}}), (e_{\text{min}}, e_{\text{max}}), (n_{\text{min}}, n_{\text{max}}) \), cost \( c_1 \) will be increased by 1.

(b) To evaluate the violation of hard constraint (ii), working patterns will be examined. Any violation of working pattern will increment cost \( c_2 \) by 1. Figure 1 shows the description of calculation of \( c_1 \) and \( c_2 \).

(c) To evaluate the violation of soft constraint, we define \( M_i, E_i, N_i, O_i \) be the total number of \( m, e, n, o \) for nurse \( i \) during the period of \( D \) and \( M_{\text{req}}, E_{\text{req}}, N_{\text{req}}, O_{\text{req}} \) be the required number of \( m, e, n, o \) for all nurses during the period of \( D \). If any of these numbers \( M_i, E_i, N_i, O_i \) are not matched to required numbers \( M_{\text{req}}, E_{\text{req}}, N_{\text{req}}, O_{\text{req}} \) respectively, cost \( c_3 \) will be increased by 1. Figure 2 shows the description of calculation of \( c_3 \).

```c
C1()
for (j=1; j<=N; j++) {
    if ((m_j<m_{\text{min}}) || (m_j>m_{\text{max}})) {c_1=c_1+1;}
    if ((e_j<e_{\text{min}}) || (e_j>e_{\text{max}})) {c_1=c_1+1;}
    if ((n_j<n_{\text{min}}) || (n_j>n_{\text{max}})) {c_1=c_1+1;}
    if (cost increased) {for (i=1; i<=D; i++) {v_{ij}=1;}}
}

C2()
for (i=1; i<=D; i++) {
    for (j=1; j<=N; j++) {
        if ((x_{ij-1}=n) && (x_{ij}=m)) {c_2=c_2+1;}
        if ((x_{ij-1}=n) && (x_{ij}=e)) {c_2=c_2+1;}
        if ((x_{ij-1}=e) && (x_{ij}=m)) {c_2=c_2+1;}
        if ((x_{ij-2}=n) && (x_{ij-1}=n) && (x_{ij}=n)) {c_2=c_2+1;}
        if (cost increased) {v_{ij}=1;}
    }
}

C3()
for (i=1; i<=D; j++) {
    if (M_i!=M_{\text{req}}) {c_3=c_3+1;}
    if (E_i!=E_{\text{req}}) {c_3=c_3+1;}
    if (N_i!=N_{\text{req}}) {c_3=c_3+1;}
    if (O_i!=O_{\text{req}}) {c_3=c_3+1;}
    if (cost increased) {for (j=1; j<=N; j++) {v_{ij}=1;}}
}
```

Figure 1. Description of Cost Function c1() and c2() and Cost Matrix v

Figure 2. Description of Cost Function c3() and Cost Matrix v
Different weights can be assigned for the costs $c1, c2,$ and $c3$. Then, the final cost function $f$ is

$$f = c1 \times w1 + c2 \times w2 + c3 \times w3$$

where $w1, w2$ and $w3$ are weight values for $c1, c2$ and $c3$.

Our goal is to minimize the cost function $f$ so as to find an optimal nurse schedule. The simplest method to find an optimal nurse schedule is a brute force approach to evaluate all possible nurse schedules and find a feasible one with the minimum cost. It guarantees that the solution is feasible with the minimum cost. The number of all possible nurse schedules is $4^{D \times N}$. If $D$ and $N$ increase, this approach is intractable. This is a class of problems for which it is believed that no efficient algorithm exists, called NP-hard (Papadimitriou, 1993). In other words, the algorithms that are guaranteed to find an optimal solution in reasonable time with the size of $D$ and $N$ may not exist. To overcome this problem, we use simulated annealing which is an approximation algorithm. SA provides an acceptable good solution in a reasonable amount of time, rather than an optimal solution.

3. Algorithmic Flow of CMSA

3.1. Simulated Annealing

SA is a probabilistic approach that can be used to find a global minimum of a function for combinatorial optimization problems. To apply this algorithm, a set of state spaces $S = \{ s_1, ..., s_n \}$ 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 [10, 11].

The criteria of the acceptance rules are:

(a) If $\Delta E \leq 0$, accept a new state $s_{new}$.

(b) If $\Delta E > 0$, 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 the state $s_{current}$ 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 [12, 13]. The main disadvantage of SA is its requirement of a large amount of computation time [1]. This is because SA is based on Monte Carlo methods, which allows a new state with a higher cost than that of a current state. To reduce this computation time, a speedup strategy using a cost matrix is used in this paper.
3.2. Cost Matrix and Transition Rule for CMSA

The very first nurse schedule \(s_{\text{first}}\), which is an \(N \times D\) matrix, is obtained by randomly assigning each nurse to one of the three shifts or day-off on each day. The cost \(E_{\text{first}}\) is calculated by applying cost function \(c_1, c_2\), and \(c_3\). This cost becomes current cost \(E_{\text{current}}\). And new schedule is generated by applying a transition rule to current schedule. A traditional transition rule is that the value of randomly chosen cell from the current schedule is changed randomly to make a new nurse schedule \(s_{\text{new}}\).

In this study we use a cost matrix \(v\) which is also an \(N \times D\) matrix to apply a transition rule to the current schedule to make a new state more efficiently. The value of each cell in the cost matrix \(v\) is assigned to 0 or 1 when the cost of new schedule is calculated. Initially the value in each cell of \(v\) is set to 0. When the cost functions \(c_1(), c_2(),\) and \(c_3()\) are applied to the new schedule and if the cost is increased by 1, the value of the corresponding cell of \(v\) is set to 1. Figure 1 shows how to set the value of \(v\). The value 1 in a cell of the cost matrix \(v\) shows the violation of constraints. Now we apply a transition rule to the cell \(x_{ij}\) with certain probability \(p\) only if the corresponding cell \(v_{ij}=1\). Figure 3 shows the description of our transition rule.

```
Transition()
    For(i=1;i<=N;i++){
        For(j=1;j<=D;j++){
            if((rand() mod100)<p&v_{ij}=1}{
                x_{ij}=random(d, e, n, o);
            }
        }
    }
```

Figure 3. The Transition Rule for SA

3.3. Cost Matrix and Transition Rule Temperature Scheduling for CMSA

The schedule implemented in SA is \(T = T_i \times e^i\) where \(e\) is a constant defining the rate of annealing, \(i\) is the iteration number, \(T_i\) the initial temperature and \(T\) the current temperature. The value of \(e\) can be easily calculated from the total number of iterations, \(k\), the final temperature, \(T_f\), and \(T_i\)

\[
e = \left(\frac{T_f}{T_i}\right)^\frac{1}{k}
\]

Figure 4 shows the pseudo code of SA for the experiments in this paper.

4. Experiments and Results

The proposed CMSA and traditional SA(TSA) were implemented in C and run 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 incorporated in both methods with exact same conditions. The goal was to check whether the algorithm could actually generate an acceptable NSP and to compare proposed SA with traditional SA. The random number generator \(\text{rand}()\) was used. Identical problem instances were solved by both methods. 100 problems were generated randomly for each set. 15 nurses for \(N\) and from 1 to 4 weeks for \(D\) were used and \(w_1=5\), \(w_2=5\) and \(w_3=1\) were assigned. Hard constraints are same for all the problems (\(m_{\text{min}}=4\), \(m_{\text{max}}=6\), \(e_{\text{min}}=3\), \(e_{\text{max}}=5\)) and (\(n_{\text{min}}=3\), \(n_{\text{max}}=5\), and
soft constraints are proportional to the periods ($M_{req}=2$, $E_{req}=2$, $N_{req}=2$ and $O_{req}=1$ for 1 week). The probability $p$ in Figure 3 was 2%.

**Figure 4. Pseudocode for SA**

The methods were compared on the basis of four criteria: number of problem solved with cost=0 ($E_{opt}=0$), the average cost of the solution obtained, and the average number of iterations to reach the final state $s_{opt}$ and the time $T_{opt}$. The CMSA outperformed TSA in every aspect. Table 1 shows the performance of two methods.

<table>
<thead>
<tr>
<th>Period</th>
<th>Method</th>
<th>$E_{opt}=0$</th>
<th>$E_{opt}$</th>
<th>$L_{opt} / k$</th>
<th>$T_{opt}$ (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 week</td>
<td>CMSA</td>
<td>79/100</td>
<td>0.42</td>
<td>305612 / 1x10^6</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>TSA</td>
<td>4/100</td>
<td>5.06</td>
<td>546054 / 1x10^6</td>
<td>1.9</td>
</tr>
<tr>
<td>2 weeks</td>
<td>CMSA</td>
<td>54/100</td>
<td>1.26</td>
<td>2900782 / 5 x10^6</td>
<td>23.9</td>
</tr>
<tr>
<td></td>
<td>TSA</td>
<td>0/100</td>
<td>8.88</td>
<td>2312737 / 5 x10^6</td>
<td>30.6</td>
</tr>
<tr>
<td>3 weeks</td>
<td>CMSA</td>
<td>72/100</td>
<td>0.66</td>
<td>12192435 / 20 x10^6</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>TSA</td>
<td>0/100</td>
<td>13.14</td>
<td>9588709 / 20 x10^6</td>
<td>121</td>
</tr>
<tr>
<td>4 weeks</td>
<td>CMSA</td>
<td>97/100</td>
<td>0.06</td>
<td>51631978 / 100 x10^6</td>
<td>926</td>
</tr>
<tr>
<td></td>
<td>TSA</td>
<td>0/100</td>
<td>17.28</td>
<td>56574940 / 100 x10^6</td>
<td>1607</td>
</tr>
</tbody>
</table>

Both methods solved all of problem instances with which did not offend any of hard constraints in all periods. CMSA generated schedules with optimal cost ($E_{opt}=0$) in all periods whereas TSA could not generate schedules with optimal cost except 1 week. Also the average costs $E_{opt}$ from CMSA were smaller than TSA and cpu time $T_{opt}$ of CMSA was faster than that of TSA. CMSA was very effective compared to TSA. In all the quality of the solution from CMSA was very impressive due to its powerful transi-
tion rule with a cost matrix. In Figure 5, we presented the differences in costs and number of iterations for CMSA and TSA. It showed the average of 100 problem instances. It is clearly shown that the cost of the solution approached the optimum faster by CMSA than TSA.

Figure 5. Comparison of Number of Iterations and Cost for CMS and TSA

5. Conclusion and Future Work

In this paper, we proposed an efficient method for NSP using SA. In this method, a cost matrix is used for transition rule. This approach generated a nurse schedule faster in time and better in quality than traditional SA. 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 is aimed at experiments on the real hospital data with more constraints and diversity of requirements.

Acknowledgements

This research was supported by Hallym University Research Fund, 2011(HRF-201208-004) and Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. 2009-0077545).

References


Authors

Young-Woong Ko received the B.S., M.S. and Ph.D. degrees in Department of Computer Engineering from Korea University, Korea, in 1997, 1999 and 2003 respectively. Since 2003, he has been with the Department of Computer Engineering, Faculty of Engineering, Hallym University, where he is currently a professor. His research interests are in the areas of storage system, operating systems, and embedded systems.
Dong-Hoi Kim received MS and Ph.D degree in Computer engineering from Hallym University in 2002 and 2006, respectively. Since then he has been working as a lecturer on computer engineering at Hallym University. His research includes bioinformatics and data mining techniques.

Minyeong Jeong received bachelor's degree in Computer Engineering from Hallym University in 2012. Since then she is in Master course in Computer Engineering at Hallym University. Her research interests include bioinformatics and algorithm.

Wooram Jeon received bachelor's degree in Computer Engineering from Hallym University in 2012. Since then he is in Master course in Computer Engineering at Hallym University. His research interests include bioinformatics and pattern recognition.

Saangyong Uhm received an MS, Ph.D degree in Computer Engineering at Hallym University in 1997, 2012 respectively. He worked as a lecturer at Hallym University. His research interests include distributed computing, algorithms and bioinformatics.

Jin Kim received an MS degree in computer science from the college of Engineering at Michigan State University in 1990, and in 1996 a PhD degree from Michigan State University. Since then he has been working as a professor on computer engineering at the Hallym University. His research includes Bioinformatics and data mining.