A Brief Study of the *Nurse Scheduling Problem (NSP)*

Lizzy Augustine, Morgan Faer, Andreas Kavountzis, Reema Patel

Submitted Tuesday December 15, 2009

0. Introduction and Background

Our interest in the *Nurse Scheduling Problem (NSP)* as a research project for *Operations Research II* originated from the desire to optimize the shift schedule for nurses at the *University of Pittsburgh Medical Center (UPMC)*. Our original intention, though altruistic, was unfortunately *infeasible*. We chose to contact *UPMC Shadyside* as they are one of the smaller hospitals in the Pittsburgh area. Our goal was to have real data to help create the data set for testing our model. We made contact with the Director of Nurses to ascertain how hospitals *actually* deal with the *NSP* as we thought it would be interesting to see how someone with a minimal background in mathematics would set about solving a complex optimization problem like this one. Our contact with *UPMC Shadyside* resulted in us finding out *exactly* how a non-mathematician handles the problem; ***they simply avoid it*** (e.g. the nurses schedule themselves).

As we began researching and reading papers we found out that the *Nurse Scheduling Problem (NSP)* is a well studied problem in mathematical optimization [2] of known complexity ($\mathcal{NP}$)-*Hard*. Consequently we found two solution methods offered; a method by *cyclic coordinate descent* [1] and a *hybrid genetic algorithm* [2]. We chose to investigate the *Genetic Algorithm (GA)* approach and implemented our model in `Java`.

We will begin our project by giving general information about genetic algorithms, followed by some notation, our formulation, implementation, and a *statistical* comparison of the crossovers we chose to construct.

1. A Word About Genetic Algorithms

The central ideas of *Genetic Algorithms (GAs)* rely on evolutionary biology. At the most basic level, the genetic data for an organism is stored in its *chromosomes*. In biology, two *chromosomes* of a *population*, referred to as *parents*, *crossover* (or *mate*) to produce *offspring* which may be subject to *random genetic mutation*. The strongest of these offspring, or most *fit*, survive to form future *populations*. *GAs* are no different than these simple ideas. For the *GA* the population is comprised of *solutions to the NSP*. The parents are *randomly selected solutions*, the crossover is a *constructed operator on a solution* [2], the mutation is a *random change in the solution*, and we apply a *fitness evaluator* to determine which solutions should move on between generations.

The basic outline of the *NSP GA* is given below from [2] and explained later when discussing our implementation:

```
Genetic Algorithm (NSP)

Initialize a Population of Individuals
While Stop Criterion not met
   Selection of Individuals to Combine
   Application of Crossover Operator
   Application of Mutation Operator
   Application of Local Search Heuristics
   Evaluation of Fitness of the Newly Created Individuals
   Update Population
Endwhile
```

The first attempts at creating a genetics-motivated algorithm were in the 1950s and exclusively featured the *mutation* of the genes of an individual *chromosome* (or *solution*). In the mid-1960s computer scientist John Holland discovered that *mating solutions* was a crucial component of genetic algorithms and proposed the technique as a solution method [6].

2. Notation

Before presenting the general model we will introduce some notation and conventions to simplify the problem statement.

- Let there be $\mathcal{I}$ *nurses* scheduled, $i = 1, \ldots, \mathcal{I}$
- Let the *scheduling period (or horizon)* be for $\mathcal{K}$ *days*, $k = 1, \ldots, \mathcal{K}$ (where we tacitly assume that the solution schedule is repeated)
- Let each day have four (4) *shifts* denoted by $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$
- Define s_1, s_2, s_3 to be *working shifts (morning, afternoon, night respectively)* and let s_4 represent a *free shift (or day off)*
- $\mathcal{D}_{i,k,s} \in \{0, 1\}$ for each *nurse* i, *day* k, *shift* s – that is, nurse i is either working the shift or not.

For simplicity we will refer to specific instances of the *NSP* by the *3-tuple*

$$(\mathcal{I}, \mathcal{K}, \mathcal{S}).$$

3. Formulation

I. Simplifying Assumptions

The following comprise a list of simplifying assumptions that we made throughout our project:

- To simplify the problem we assume that there are *no substitutions* available amongst different classes of nurses. For instance, it is not possible to replace an *RN* by some number of any other nurse, say 10 *LPNs* - even if in theory they are able to perform the same tasks. We use this assumption to assert that we may schedule each type of nurse seperately.
- To limit the problem size we will restrict the *aversion factors* of the nurses to be chosen from $\mathcal{A} = \{1, 2, 3, 4\}$ where 4 indicates a *strong aversion* to the shift and 1 represents a *preference* to the shift.
- We make no differentiation or take any special note of holidays or weekends - that is, we treat *all days* the same way.
- *All shifts* are *not* equal, thus each will require a different *minimum* number of nurses depending on the *day/shift*.

II. Constraints

Hard constraints are those that **cannot be violated** and define the feasibility of solutions. Hard constraints are concerned with the **hospital's needs** as opposed to the nurses' preferences. Let $\mathcal{H}_{k,s}$ represent the *Hospital's Minimum Coverage Constraint* for *day* k, *shift* s, then our hard constraints are:

- For all *days* k and *shifts* s where $k = 1, \ldots, \mathcal{K}$ and $s = s_1, s_2, s_3, s_4$:

$$\sum_{i=1}^{\mathcal{I}} \mathcal{D}_{i,k,s} \geq \mathcal{H}_{k,s}$$

  that is, each schedule must satisfy the hospital's *minimum coverage constraint*.

- For all i, k where $k = 1, \ldots, \mathcal{K} - 1$, $i = 1, \ldots, \mathcal{I}$:

$$\mathcal{D}_{i,k,s_3} + \mathcal{D}_{i,k+1,s_1} \leq 1$$

  that is, no nurse may be scheduled to work a *night shift* (s_3) followed immediately by a *morning shift* (s_1).

- For all *nurses* i and *days* k:

$$\sum_{s \in \mathcal{S}} \mathcal{D}_{k,i,s} = 1$$

that is, each nurse must be scheduled for *exactly* one shift each day.

Soft constraints are those which **may be violated** but with an associated **penalty cost**. Constraints of this form contribute to the objective function and define optimality rather than feasibility. This notion of *constraint satisfaction* with respect to the **nurses' preferences** is a crucial component of the *NSP*. Our soft constraints are:

- For each nurse $i = 1, \ldots, \mathcal{I}$, and for $k = 1, \ldots, \mathcal{K} - 2$:

$$\mathcal{D}_{i,k,s_4} + \mathcal{D}_{i,k+1,s_4} + \mathcal{D}_{i,k+2,s_4} \leq 2$$

that is, each nurse may have *no more than three days off in a row.*

- For each nurse $i = 1, \ldots, \mathcal{I}$, and for $k = 1, \ldots, \mathcal{K} - 6$:

$$\sum_{j=k}^{k+6} \mathcal{D}_{i,j,s} \leq 7$$

that is, each nurse may work *no more than seven days in a row.*

III. Objective Function

Let $\mathcal{F} : \{0,1\}^{\mathcal{I} \times \mathcal{K}\mathcal{S}} \rightarrow \mathcal{Z}$ be a *linear function* that takes in a schedule matrix and outputs an integer *fitness value* based on the total of the *nurses' individual aversions to the schedule.* Similarly, let $\mathcal{G} : \{0,1\}^{\mathcal{I} \times \mathcal{K}\mathcal{S}} \rightarrow \mathcal{Z}$ be an *exponential function* for the *hospital's cost* due to over/under-staffing.

To formulate an objective function we shall *minimize* the *overall nurse aversion* to the schedule as well as the *hospital's schedule cost* subject to a *weight,* λ. Let $\lambda \in \mathcal{R}$ be such that $\lambda \in [0,1]$ and consider it a *weight in the interest of the nurses* (but decided on by the hopsital). To form the objective function we may choose λ to favor either the *interest of the nurses* ($\lambda > 0.5$), the *hospital* ($\lambda < 0.5$), or *both interests equally* ($\lambda = 0.5$).

Since these functions consider **bad things** (e.g. *aversion*, *cost*), it follows that the objective of the *NSP* is to *minimize*:

$$\lambda * \mathcal{F}(\sum_{i=1}^{\mathcal{I}} \sum_{k=1}^{\mathcal{K}} \sum_{s=1}^{\mathcal{S}} \mathcal{D}_{i,k,s}) + (1-\lambda) * \mathcal{G}(\sum_{i=1}^{\mathcal{I}} \sum_{k=1}^{\mathcal{K}} \sum_{s=1}^{\mathcal{S}} \mathcal{D}_{i,k,s})$$

4. Implementation Notes

To explain our implementation we begin by presenting sample input for our `Java` program. We stored the instances from the *NSPLib* [7] in `.txt` files and used the `Java Scanner Class` to import our data. Below is a sample instance corresponding to $(\mathcal{I}, \mathcal{K}, \mathcal{S}) = (8, 3, 4)$:

```
8  3  4
3  3  2  0
0  1  2  0
3  3  1  0
3  4  3  3  1  4  4  4  3  4  3  3
1  1  1  1  1  4  4  4  1  1  1  1
2  2  2  2  1  1  1  1  2  2  2  2
1  3  1  1  1  2  2  2  1  3  1  1
3  1  3  3  1  4  4  4  3  1  3  3
3  1  3  3  4  3  3  3  3  1  3  3
4  2  4  4  3  3  3  3  4  2  4  4
3  1  3  3  2  4  4  4  3  1  3  3
```

The first three values are $\mathcal{I}$, $\mathcal{K}$, and $\mathcal{S}$ respectively. The following matrix, $\mathcal{H}$, of dimension $\mathcal{K} \times \mathcal{S}$ represents the *hospital's minimum coverage requirements*. The input is structured so that each row (*day*) represents the shift requirements (s_1, s_2, s_3, s_4) for days $k = 1, \ldots, \mathcal{K}$. Recall that the fourth shift, s_4 is the *off* shift − representing that a nurse is *not* working that day. It follows that the fourth shift coverage requirement is zero, that is $s_4 = 0$ always. The remaining matrix, $\mathcal{P}$, of dimension $\mathcal{I} \times \mathcal{K}\mathcal{S}$ is the *nurse preference matrix*. $\mathcal{P}$ is such that row i of $\mathcal{P}$ represents the preferences of nurse i, $i = 1, \ldots, \mathcal{I}$, where the entries of each row represent the aversion of the nurse i to specific shift s on a given day k for each of the $\mathcal{K} \times \mathcal{S}$ shifts of the scheduling horizon.

A solution schedule, $\mathcal{D}^\star$, is a 0/1 matrix of dimension $\mathcal{I} \times \mathcal{K}\mathcal{S}$. Each row i of $\mathcal{D}^\star$ represents the schedule of the i^{th} nurse, $i = 1, \ldots, \mathcal{I}$. Observe that:

$$\mathcal{D}^\star_{i,k,s} = \begin{cases} 1 & \textit{if nurse } i \textit{ is working shift } s \textit{ on day } k \\ 0 & \textit{else} \end{cases}$$

And it follows that since each nurse will have only one shift per day, each nurse will have *exactly* $\mathcal{K}$ 1s in their respective row of $\mathcal{D}^{\star}$.

The program begins by randomly generating schedules. These schedules are immediately tested for feasibility and are discarded if they are infeasible, that is the solution violates at least one hard constraint. The feasibility check merely confirms that all *hard constraints* are *satisfied*.

If a schedule is feasible, then it is tested for *fitness* and assigned a corresponding *fitness value*. Recall that the fitness value measures how well the schedule satisfies the *soft constraints*. Thus a lower fitness value implies a better scheule. In our program we used the nurses' total aversion, the hospital cost, and the aforementioned *soft constraints*. We arbitrarily chose $\lambda = 0.46$ so that the constraints related to the hospital's cost were 54% of the total fitness value. To calculate the overall aversion of a schedule we simply sum over all nurses i, $i = 1, \ldots, \mathcal{I}$, after taking the dot product of their corresponding row i from $\mathcal{D}^{\star}$ with their respective row i from $\mathcal{P}$. For the hospital's cost, we found the difference between the total number of nurses scheduled to work a specific shift and the minimum coverage for that specific shift for each shift, squared the difference, and then added all of those values together. We chose to square the values, rather than have a linear function, to emphasize the severity of having too many or too few nurses working a specific shift.

Once a schedule has been determined feasible and given a corresponding fitness value, we then created a `Java Object` called a `Chromosome`. The `Chromosome` contains two data fields; a solution schedule and the schedule's corresponding fitness value. A population limit of 15 was established and we continue to make `Chromosome`s until we have a full population; at which point we may apply the crossover step. Given a full population, we determine which chromosomes will be used as the parents in the first generation of the algorithm. To decide, we randomly choose two chromosomes from the population, examine their fitness values and choose the more fit chromosome. This process is repeated until there is a set of four parents. Once the set of parents are chosen, we randomly choose 2 parents to apply the crossover operator to, in order to create a *child* chromosome.

We implemented three crossover operators using ideas from [2] which we briefly describe:

- *Crossover 1* − Takes half of the solution (columns) from parent $\mathcal{P}_1$ and half of the solution from parent $\mathcal{P}_2$ to create a child $\mathcal{C}_{\mathcal{P}_1\mathcal{P}_2}$

- *Crossover 2* − Uses a randomly generated crossover point, x, taking the first x days of parent solution $\mathcal{P}_1$ (columns) and the remaining $\mathcal{K}-x$ days from $\mathcal{P}_2$ to create a child solution $\mathcal{C}_{\mathcal{P}_1\mathcal{P}_2}$
- *Crossover 3* − Uses a randomly generated crossover point, i, taking the first i nurses of parent solution $\mathcal{P}_1$ (rows) and the remaining $\mathcal{I}-i$ rows from $\mathcal{P}_2$ to create a child solution $\mathcal{C}_{\mathcal{P}_1\mathcal{P}_2}$

Further, *GAs* require random mutation so we implemented a mutation that would occur roughly 10% of the time. Mutations are crucial for genetic algorithms because they allow for a more diverse population and aid in finding a global optimal solution. For our mutation, when a child schedule was produced, there was a 10% chance a mutation would be applied. The mutation randomly chooses a shift for a randomly chosen nurse to change the given entry of the solution schedule. Note that we maintained the solution by fixing the remaining entries of the day effected by the mutation.

As soon as the child is produced it is checked for feasibility. If the child solution is feasible, its fitness value is calculated and the child solution becomes a new chromosome. If the child solution is not feasible it is discarded. The crossover process continues until there are eight child chromosomes. The child chromosomes are then compared to the current population and the best children chromosomes replace the worst original chromosomes until a new, (hopefully) improved, population of size 15 is generated. This new population is treated as the initial population and the process repeats. For our program we repeated this process for 50 generations. In the last generation, the most fit schedule is chosen as the solution.

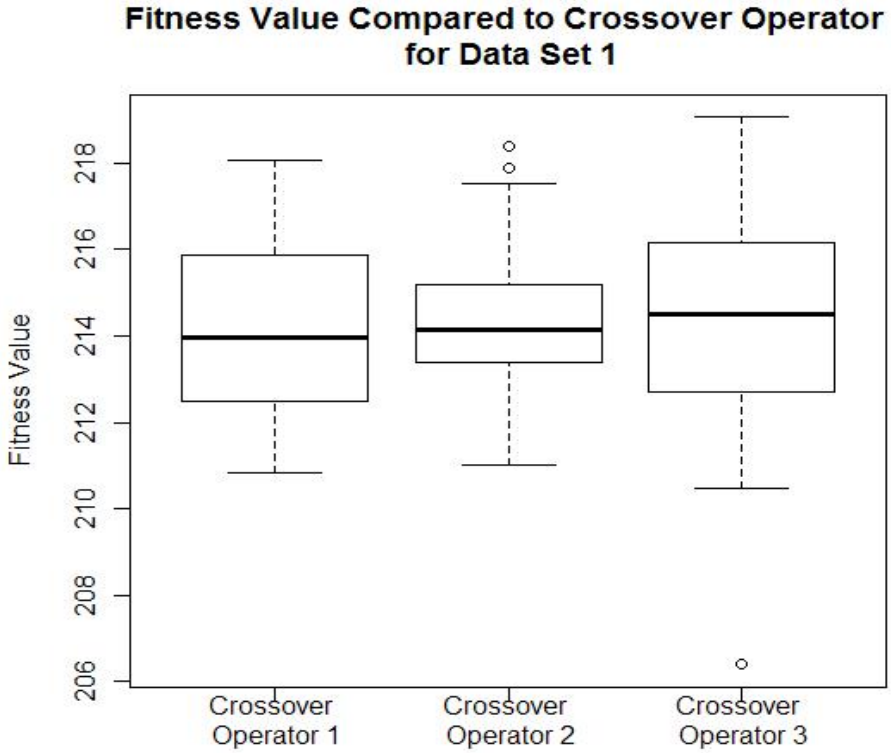
5. Analysis of a Sample Instance

We worked with two different data sets when we implemented our model in order to present a comparison study of the three crossover operators we implemented, labeled simply *Crossover 1, Crossover 2, Crossover 3*. *Data Set 1* is an instance of the form $(\mathcal{I}, \mathcal{K}, \mathcal{S}) = (25, 7, 4)$ and *Data Set 2* is an instance of the form $(\mathcal{I}, \mathcal{K}, \mathcal{S}) = (60, 28, 4)$. We ran our model using each of the three operators, for both of the data sets, 30 times each and obtained the following results:

Results for *Data Set 1*

The mean fitness values for *Crossover 1, Crossover 2*, and *Crossover 3* were 214.39, 214.50, and 214.34, respectively. The bloxplot below shows the distribution for each crossover. The overall shape of the distribution of each of the

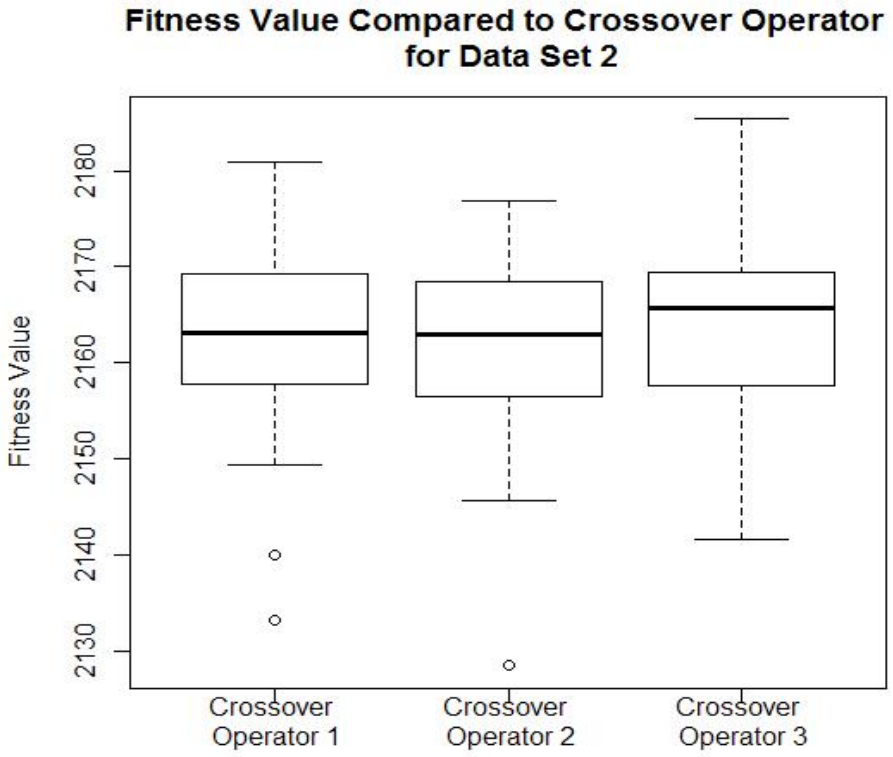
crossovers is fairly similar. They all have medians around a fitness value of 214 and each of the distributions is symmetric. An important distinction for *Crossover 2* is that its *IQR (Inter Quartile Range)*, where the middle 50% of the data lies, is much smaller than those of *Crossover 1* and *Crossover 3*. From the graph we see that the mean fitness values for the three crossover operators are about the same. To statistically verify that the means are the same, we ran an *Analysis of Variance test (ANOVA)* with the *null hypothesis* that *all three means are the same*, and the *alternative hypothesis* that *at least one of the means is different*. After running the test, we obtained $\mathcal{F} = 0.0422$ on 2, 56.6 degrees of freedom, which has a corresponding *p-value* of 0.9587. Since the *p-value* is so high, we can say that the means of the three crossover operators are equal.



Results for *Data Set 2*

The mean fitness values for *Crossover 1, Crossover 2, and Crossover 3* were 2162.12, 2161.92, and 2164.60, respectively. The boxplot below shows the distribution for each crossover. The overall shape of the distribution of each

crossover is symmetric. In addition, the size of the *IQR* for all three distributions is the same. Once again, we can graphically see that the mean fitness values for the three crossover operators are about the same. To statistically verify that the means are the same, we ran an *ANOVA* with the *null hypothesis* that *all three means are the same*, and the *alternative hypothesis* that *at least one of the means is different*. After running the test, we obtain $\mathcal{F} = 0.6295$ on 2, 57.97 degrees of freedom, which has a corresponding *p-value* of 0.5365. Therefore, we have strong statistical evidence that the means of the three crossover operators are equal.



Now that we have individually compared the crossover operators with respect to each of the data sets, we can make inferences comparing them side-by-side. Compared to *Data Set 1* (which contains 25 nurses), *Data Set 2* (which contains 60 nurses), has a much higher fitness value, which means that the nurses as a whole are less satisfied. This makes sense practically because when there are more people to take into consideration, there are more (soft) constraints, and it makes it harder to form a solution that satisfies as many constraints as possible. Another interesting observation between the

two data sets is that although within each data set the *IQRs* are about the same for all three crossover operators, the approximate size of the *IQR* for *Data Set 2* ($2157 < 2169$) is larger than the approximate size of the *IQR* for *Data Set 1* ($212 < 215$). This also makes sense practically because in general when there is a larger data set there are many more feasible solutions, some of which may be much more optimal than others. On the other hand, when there is a smaller data set, there may not be as many feasible solutions, and they will tend to have similar optimal values.

6. Conclusion

Spending a semester learning about the *NSP* outside of class was a valuable experience that exposed us to a difficult problem with a *slick* solution method − the genetic algorithm. In addition to spending time reading papers and refining our understanding of the problem and its solution method, we were able to experience a *group work* environment more like that found in the industry. Furthermore we had to implement the solution in code − a difficulty we often are able to avoid in classes. Finally, we were exposed to a *metaheuristic* technique not otherwise covered in our coursework and were able to successfully apply it independently to a problem that we chose.

7. Sources

[1] Miller, H., Pierskalla, W. & Rath, G. (1976). Nurse Scheduling Using Mathematical Programming. *Operations Research, Vol 24, No 5, Special Issue on Health Care, pp 857-870.*

[2] Maenhout, B. & Vanhoucke, M. (2007). Comparison and hybridization of crossover operators for the nurse scheduling problem. *Annals of Operations Research, Vol 159, pp 333-353.*

[3] Burke, Crowling, Causmaecker & Berghe (2001). A Memetic Approach to the Nurse Rostering Problem. *Applied Intelligence, Vol 15, Num 3,pp 119-214.*

[4] Dowsland, K. & Thompson, J. (2000).Solving a nurse scheduling problem with knapsacks, networks and tabu search. *Journal Of the Operations Research Society, Vol 51, pp 825-833.*

[5] Maenhout, B. & Vanhoucke, M. (2005). NSPLib A Nurse Scheduling Problem Library: A tool to evaluate (meta-)heuristic procedures. *ORAHS*

2005 Proceedings.

[6] Holland, J. (1992). Genetic Algorithms: Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. *Scientific American, July 1992, pp 66-72.*

[7] **Data Set** Maenhout, B. & Vanhoucke, M. (2005). Tutorial NSPLib benchmark dataset.