CREW ROSTERING PROBLEM: A RANDOM KEY GENETIC ALGORITHM WITH LOCAL SEARCH

THESIS

Presented in Partial Fulfillment of the Requirements for

the Master of Science in

Graduate School of The Ohio State University

By

Ravi Kanth Rachakonda, B. Tech

* * * * *

The Ohio State University

2008

Approved by

Dissertation Committee:

Dr. Theodore T. Allen, Adviser

Dr. Clark A. Mount-Campbell

______________________________

Adviser

Graduate Program in Industrial
And Systems Engineering
ABSTRACT

The Crew Rostering Problem is a part of the operational planning process of any airline industry. Like many other combinatorial optimization problems, the crew rostering problem is an integer program which is, in the formulations considered here, viewed as a black box optimization problem because of the many special constraints and weighted considerations. Specifically, the so called “18/15 day crew rostering problem” schedules specific kind of pilots who work only for either 18 days or 15 days in a month. The formulations considered here focus on the particular balance between economics and pilot satisfaction relevant to a major Midwestern airline company. The size of the problem (on order of 1000 or fewer decision variables) makes the application of a genetic algorithm (GA) computationally feasible, i.e., the overhead from the GA is not prohibitive. The key aspects of the proposed solutions methods are (1) the random keys approach used which guarantees many constraints are satisfied automatically and (2) a local search method proposed here. The computational performance of the proposed methods is compared with CPLEX branch and bound implementations. These computational comparisons motivate the application of both the random keys genetic algorithm and the local search adjustment methods.
Dedicated to my family.
ACKNOWLEDGEMENTS

First of all, I want to thank my advisor, Dr. Theodore T. Allen, who is not only my mentor, but also a good friend of mine. He provided me with great insights on this research. Some of the key research ideas contained here was developed jointly with him like encoding of the schedule, random keys genetic algorithm.

I want to thank David Benson, Operations Planning, NetJets, Inc for his cordial support and encouragement of the idea and providing the vital data. I want to specially thank Jack Keoshian, NetJets, Inc. for his extreme support and encouragement for the research since it was started. I also thank him for being a great mentor and a good friend while I was at NetJets, Inc. I thank all the people who helped me at NetJets. I want to specially thank my brothers Kranthi Kumar Rachakonda and Venu Rachakonda for all the help, encouragement and support throughout the research. You always made me feel lucky.

Finally, I would like to thank my all other friends for their patience and support during research.
VITA

September 10th, 1985…………………………………………Born – Hyderabad, India


Jawaharlal Nehru Technological
University, Hyderabad, India

2008 …………………………………………………………………..M.S Industrial and Systems Eng.

The Ohio State University,

Columbus, OH
FIELDS OF STUDY

Major Field: Operations Research
# TABLE OF CONTENTS

Abstract ......................................................................................................................... ii

Dedication ....................................................................................................................... iii

Acknowledgements ......................................................................................................... iv

Vita ....................................................................................................................................... v

List of Tables .................................................................................................................... x

List of Figures .................................................................................................................. xi

Chapters

1. Introduction ................................................................................................................. 1

   1.1 Crew Rostering Problem ..................................................................................... 2

   1.2 Mathematical Formulation ................................................................................... 6

2. Discussion of black box optimization methods ......................................................... 20

   2.1 Overview .............................................................................................................. 20

   2.2 Genetic algorithms for the bus driver scheduling problem .............................. 21

   2.3 Self-Adjusting Algorithm for Driver Scheduling ........................................... 23
2.4 A dynamic neighborhood based tabu search algorithm .......................... 25

3. Random Keys Genetic Algorithm With Local Search .......................... 27
   3.1 General Presentation ........................................................................ 27
   3.2 Random keys Genetic algorithms (RKGA) ........................................ 29
      3.2.1 Random Keys Coding Scheme .................................................. 29
      3.2.2 Random Keys Genetic Algorithm (RKGA) Operators .................. 30
   3.3 Pseudo code for Random Keys Genetic Algorithm (RKGA) ............. 33
   3.4 Solution Enhancer (Local Search) ................................................... 34
   3.5 Pseudo code for Adaptive Local Search Random Keys Genetic Algorithm .... 35

4. Computational Results ........................................................................... 37
   4.1 Real crew rostering test problem ..................................................... 39
   4.2 RKGA Vs LSKRKGA Vs ALSRKGA ................................................. 41
   4.3 Two sample t-test for RKGA and ALSRKGA .................................. 42
   4.4 Two sample t-test for LSRKGA and ALSRKGA .............................. 42
4.5 Two sample t-test for RKGA and LSRKGA ........................................ 43

4.6 One sample t-test for RKGA and Branch and Bound method ................. 43

5. Conclusions.................................................................................................. 45

References......................................................................................................... 47
LIST OF TABLES

Table 1. 1: Predefined tour pairs ........................................................................................................ 5

Table 4. 1: Objective value at 200th iteration .................................................................................. 42

Table 4. 2 One Sample test data ........................................................................................................ 43
LIST OF FIGURES

Figure 1. 1: Piecewise Linear Function for Crew Score ......................................................... 17

Figure 2. 1: Solution with fourteen pieces of work ................................................................. 21

Figure 3. 1: Crossover of two parents ...................................................................................... 28

Figure 3. 2: Random Keys Coding Scheme ............................................................................. 30

Figure 4. 1: GA Objective value for A4 .................................................................................. 37

Figure 4. 2: Crews on duty on a day for the month of Nov 08 .................................................. 40

Figure 4. 3: Schedule for the month of Nov 08 ...................................................................... 41

Figure 4. 4: RKGA Vs LSRKGA Vs ALSRKGA seed 1 .............................................................. 41

Figure 4. 5: Boxplot for all the algorithms ............................................................................. 41
CHAPTER 1

INTRODUCTION

The Crew Rostering Problem (CRP) is part of the operational planning process of airline industries that has been studied by many operations researchers (Bryan A. Norman & James C. Bean, (1998), Milind G. Sohoni, T. Glenn Bailey & Kristi G. Martin (2003)). The goals of crew scheduling is to determine which days each pilot or crew member is working and on how long of a tour. The varied nature of the particular characteristics of each company has lead to a variety of techniques applied (T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002), Jingpeng Li & Raymond S. Kwan (2005)). Here we focus on the needs of a particular Midwest air line company relating to balancing financial goals and the subjective desires of pilots and their union.

In the terminology here, the process of crew rostering consists of constructing a set of tours, which together with training and pilot preferences cover all the forecasted crew demand. Crew rostering is subject to constraints that are specific to each company (Jeng, C; Liu, T; Chang, Y. (2005)). These constraints generally arise from government legislation, union work rules and some operational rules like tour sizes that are settled between pilots and the airline industry. The planning process starts by the generation of crew schedules, aiming at maximizing the number of pilots available on any required day, complying with the requests and constraints. Predefined tour pairs are allotted to each pilot
to make up the pilot’s schedule known as “string”. A set of strings for all pilots in a fleet make up a feasible schedule, i.e. a schedule for all the pilots in a fleet for a period of one month. An optimal solution for the crew rostering problem is a set of feasible strings that hopefully exceed the forecasted daily crew target, while respecting union work rules, individual pilot vacation and training periods, and achieving a high level of satisfaction of the pilot specified individual preferences for the pilots. The problem is further complicated by various preferences, hard and soft constraints. In the next section, we describe the overall structure of the crew rostering problem. Next, we discuss the many heuristics that could be applied to black box scheduling. The proposed heuristic based on the genetic algorithm in Hadj Alouane and Bean (1997) is the described (Bryan A. Norman & James C. Bean, (1998)). Computational results follow showing the significant advantages of the proposed methods for the Midwest manufacturer. The results are based on the real problems with all their complexities as described fully in an on-line supplement. The last section closes with conclusions about the benefits of the proposed methods.

1.1 Crew Rostering problem
In this section, the details of the crew rostering problem of interest in this thesis are described. The complexity will be used to motivate the choice of black box optimization methods described in subsequent sections. The relevant Midwestern manufacturer large number of pilots and wide variety of fleets. Each fleet has a different kind of airplanes and requires pilots with special qualifications. Pilots in a Midwest air line company are categorized into 3 groups:
Pilots are classified on the number of days they work. 7 & 7 pilots works for 7 days in a row followed by 7 days off. These pilots are scheduled by different scheduler. 18 day pilots works for maximum of 18 days in a month. Schedule for the next month needs to be provided by 15th of every month. So the schedule for these pilots need to be done carefully depending upon the demand forecast. 15 day pilots work for a maximum of 15 days. These pilots can be given notice of work as less as 3 days. 15 day pilots cannot work a tour more than 5 days unless it is a training tour. In this crew rostering problem, we are concerned with 18 day and 15 day pilots only. We need to generate a schedule for 44 days. While generating the schedule for next month, last 8 days of the current month are taken into consideration and the schedule is generated in accordance with the last 8 days. 13-14 days of extra schedule is made every time to make the monthly schedule more continuous. For the scheduling period assign duty tours to 18 day and 15 day schedule pilots such that good coverage is achieved w.r.t. the forecasted daily crew target, while respecting union work rules, individual pilot vacation and training periods, and achieving a high level of satisfaction of the pilot specified individual preferences for the 18 day pilots. There are predefined tour pairs like <8, 5>, <7, 4>, <6, 3>, <5, 3>, <4, 3>, <3, 3>. An eight day tour should be followed by a five days off and a seven day tour should be followed by a four
days off. Depending on the ranking pilots are classified into 6 types. 1) PIC 2) SIC 3) TR 4) CA 5) IP 6) NRFO.

Simplified Form of Objective function:

\[
\text{Maximize: CrewingScore + crewPreferenceScore}
\]

Where crewingScore represents the economics of how well the daily crewing for a given solution meets the forecast daily crew demand for this fleet. A large bonus is added to the objective function for each crew day added on a day when the crew total is under the forecast crew target, while only a small bonus is added to the objective for adding an additional crew on days when the crew total is over the forecast crew demand. Currently a piecewise linear function with three breakpoints is used to establish 4 different marginal bonuses to the objective for addition of a crew day. The breakpoints currently in use represent crews provided on any given day being 95\%, 1.05\% and 1.15 \% of the forecast crew demand.

CrewPreferenceScore quantifies how well any particular rostering plan satisfies the various preferences requested by the 18 day pilots in the current scheduling period. Currently each 18 day pilot may request preferences related to:

- Requested dates off
- Long/short tour length
- Balanced/unbalanced rest
The bonus to be applied for any particular preference requested by any particular pilot will depend on that pilot’s seniority number.

- DaysOffScore, shortTourScore, longTourScore, imbalancedTourScore, weekendScore, HolidayScore, balancedTourScore represent bonuses to the objective for meeting each pilot specified preference. All these scores make up CrewPreferenceScore.

- threeDayScore is a penalty imposed on each three day tour to reduce their occurrence.

<table>
<thead>
<tr>
<th>Tour Length</th>
<th>Rest Period</th>
<th>Availability</th>
<th>Tour pair Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>18,15</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>18,15</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>18,15</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1. 1: Predefined tour pairs

Some Constraints:

- No two tour pairs (i.e. work + rest period) can overlap for a pilot.

- Enforce maximum workday limit for each pilot; e.g. 18 – (days vacation + days training)

- Allow maximum of one 8 day tour per pilot this period and only for pilots who haven’t worked one previously.
- Limit total number of 8 day tours this month to be equal to MaxNum8dayTours or less. Note: An 8 day tour is considered to be "in" a month (or 4 month scheduling period) if it ends in that month (or that 4 month period).

- Implement the “defined” days for each pilot, that is, the days required to be on or off; e.g. seam transition days worked or off, vacation days or “hard” days off.

- Force some tour to be worked that starts on start day of training. If training 5 or more days, force tour assigned to be exactly same length as # of training days; i.e. don’t extend if training 5 or more days.

- Set $Y[p][d] = 1$ whenever pilot $p$ is working on day $d$, as indicated by the $X$ decision variable.

### 1.2 Mathematical Formulation

Maximize \( \text{TotalObjective} \)

Subject to:

\[
\text{TotalObjective} = \text{daysOffScore} \times \text{daysOffCoef} + \text{shortTourScore} \times \text{shortTourCoef} + \\
\text{longTourScore} \times \text{longTourCoef} \times \text{longTourCoef} + \text{balancedTourScore} \times \\
\text{balancedTourCoef} + \text{imbalancedTourScore} \times \text{imbalancedTourCoef} + \\
\text{weekendOnScore} + \text{weekendOffScore} + \text{holidayOnScore} + \text{holidayOffScore} + \\
\text{crewingScore} \times \text{crewingCoef} + \text{threeDayScore} + \text{pilotBonusPeriodScore} + \\
\text{fleetBonusPeriodScore} + \text{pilotWorkDayScore}
\]
TotalObjective is a summation of different scores like DaysOffScore, threeDayScore attained by granting preferences or satisfying constraints.

\[ Y[p][d] = \sum_{tp \in TP} \sum_{d \in Pd} X[p][d][tp] \quad (1.1) \]

\[ \forall \ p \in P, d \in Pd, d2 \leq d \leq d2 + tp.lengthOn - 1 \]

\[ Y[p][d] = 1 \] whenever pilot p is working on day d.

Pilot productivity varies depending on the day of the tour. Pilot productivity is not considered as 100% on starting day and ending day of the tour. Pilot Productivity for Primary In Charge (PIC) is around 0.3 on starting day and 0.2 on ending day of the tour.

\[ NumPICFirstDays[d] = \sum_{tp \in TP} \sum_{p \in P} X[p][d][tp] - \sum_{p \in P} \sum_{tr \in TE} 1 \quad (1.2) \]

\[ \forall \ d \in MP, p.rank = 1, tr.cmi = p.cmi, tr.start = d \]

\[ NumPICLastDays[d] = \sum_{tp \in TP} \sum_{p \in P} \sum_{d \in Pd} X[p][d][tp] - \sum_{p \in P} \sum_{tr \in TE} \sum_{tp \in TP} \sum_{d \in Pd} X[p][d2][tp] \quad (1.3) \]

\[ \forall d \in MP, p.rank = 1, d2 + tp.lengthOn - 1 = d, tr.cmi = p.cmi, tr.start + tr.duration - 1 = d, d2 + tp.lengthOn - 1 = d \]

\[ NumPICFirstDays[d] \] and \[ NumPICLastDays[d] \] are the count of first days and last days of PIC pilots.

\[ NumSICFirstDays[d] = \sum_{tp \in TP} \sum_{p \in P} X[p][d][tp] - \sum_{p \in P} \sum_{tr \in TE} 1 \quad (1.4) \]

\[ \forall d \in MP, p.rank = 2, tr.cmi = p.cmi, tr.start = d \]

\[ NumSICLastDays[d] = \sum_{tp \in TP} \sum_{p \in P} \sum_{d \in Pd} X[p][d][tp] - \sum_{p \in P} \sum_{tr \in TE} \sum_{tp \in TP} \sum_{d \in Pd} X[p][d2][tp] \quad (1.5) \]

\[ \forall d \in MP, p.rank = 2, d2 + tp.lengthOn - 1 = d, tr.cmi = p.cmi, \]
\[\text{tr. start} + \text{tr. duration} - 1 = d, d2 + \text{tp. length}0n - 1 = d\]

\[\text{NumTRFirstDays}[d] = \sum_{\text{tp} \in \text{TP}} \sum_{p \in P} X[p][d][tp] - \sum_{p \in P} \sum_{\text{tr} \in \text{TE}} 1\]  \hspace{1cm} (1.6)

\[\forall \ d \in MP, p. \text{rank} = 3, d2 + \text{tp. length}0n - 1 = d, \text{tr. cmi} = p. \text{cmi}, \text{tr. start} = d\]

\[\text{NumTRLastDays}[d] = \sum_{\text{tp} \in \text{TP}} \sum_{p \in P} \sum_{d2 \in \text{EP}d} X[p][d2][tp] - \sum_{p \in P} \sum_{\text{tr} \in \text{TE}} \sum_{\text{tp} \in \text{TP}} \sum_{d2 \in \text{EP}d} X[p][d2][tp]\]  \hspace{1cm} (1.7)

\[\forall \ d \in MP, p. \text{rank} = 3, d2 + \text{tp. length}0n - 1 = d, \text{tr. cmi} = p. \text{cmi}, \text{tr. start} + \text{tr. duration} - 1 = d, d2 + \text{tp. length}0n - 1 = d\]

\[\text{NumCAFFirstDays}[d] = \sum_{\text{tp} \in \text{TP}} \sum_{p \in P} X[p][d][tp] - \sum_{p \in P} \sum_{\text{tr} \in \text{TE}} 1\]  \hspace{1cm} (1.8)

\[\forall \ d \in MP, p. \text{rank} = 4, \text{tr. cmi} = p. \text{cmi}, \text{tr. start} = d\]

\[\text{NumCALastDays}[d] = \sum_{\text{tp} \in \text{TP}} \sum_{p \in P} \sum_{d2 \in \text{EP}d} X[p][d2][tp] - \sum_{p \in P} \sum_{\text{tr} \in \text{TE}} \sum_{\text{tp} \in \text{TP}} \sum_{d2 \in \text{EP}d} X[p][d2][tp]\]  \hspace{1cm} (1.9)

\[\forall d \in MP, p. \text{rank} = 4, d2 + \text{tp. length}0n - 1 = d, \text{tr. cmi} = p. \text{cmi}, \text{tr. start} + \text{tr. duration} - 1 = d, d2 + \text{tp. length}0n - 1 = d\]

\[\text{NumIPFirstDays}[d] = \sum_{\text{tp} \in \text{TP}} \sum_{p \in P} X[p][d][tp] - \sum_{p \in P} \sum_{\text{tr} \in \text{TE}} 1\]  \hspace{1cm} (1.10)

\[\forall \ d \in MP, p. \text{rank} = 5, \text{tr. cmi} = p. \text{cmi}, \text{tr. start} = d\]

\[\text{NumIPLastDays}[d] = \sum_{\text{tp} \in \text{TP}} \sum_{p \in P} \sum_{d2 \in \text{EP}d} X[p][d2][tp] - \sum_{p \in P} \sum_{\text{tr} \in \text{TE}} \sum_{\text{tp} \in \text{TP}} \sum_{d2 \in \text{EP}d} X[p][d2][tp]\]  \hspace{1cm} (1.11)

\[\forall \ d \in MP, p. \text{rank} = 5, d2 + \text{tp. length}0n - 1 = d, \text{tr. cmi} = p. \text{cmi}, \text{tr. start} + \text{tr. duration} - 1 = d, d2 + \text{tp. length}0n - 1 = d\]
\[ NumNRFOFirstDays[d] = \sum_{tpeTP} \sum_{p \in P} X[p][d][tp] - \sum_{p \in P} \sum_{tr \in TE} 1 \]  
\[ \forall \ d \in MP, p.rank = 6, tr.cmi = p.cmi, tr.start = d \]  

\[ NumNRFOLastDays[d] = \sum_{tpeTP} \sum_{p \in P} \sum_{d2 \in PD} X[p][d2][tp] - \sum_{p \in P} \sum_{tr \in TE} \sum_{tpeTP} \sum_{d2 \in PD} X[p][d2][tp] \]  
\[ \forall \ d \in MP, p.rank = 6, d2 + tp.lengthOn - 1 = d, tr.cmi = p.cmi, \]  
\[ tr.start + tr.duration - 1 = d, d2 + tp.lengthOn - 1 = d \]

\[ PIConDuty[d] \] is the effective number of 18/15 type PIC pilots working on day ‘d’.

\[ PIConDuty[d] = PICs77[d] - TrainingPIC[d] + \sum_{p \in P} Y[p][d] \]
\[ - (1 - FirstDayProductivity) \times NumPICFirstDays[d] \]
\[ - (1 - LastDayProductivity) \times NumPICLastDays[d] \]  
\[ \forall \ d \in MP, p.rank = 1 \]  

\[ SIConDuty[d] \] is the effective number of 18/15 type SIC pilots working on day ‘d’.

\[ SIConDuty[d] = SICs77[d] - TrainingSIC[d] + \sum_{p \in P} Y[p][d] \]
\[ - (1 - FirstDayProductivity) \times NumSICFirstDays[d] \]
\[ - (1 - LastDayProductivity) \times NumSICLastDays[d] \]  
\[ \forall \ d \in MP, p.rank = 2 \]  

\[ TRonDuty[d] \] is the effective number of 18/15 type TR pilots working on day ‘d’.
\( T_{onDuty}[d] = TRs77[d] - TrainingTR[d] + \sum_{p \in P} Y[p][d] \)

\[ (1.16) \]

\( - (1 - FirstDayProductivity) \times NumTRFirstDays[d] \)

\( - (1 - LastDayProductivity) \times NumTRLastDays[d] \)

\( \forall \ d \in MP, p.rank = 3 \)

\( CA_{onDuty}[d] \) is the effective number of 18/15 type CA pilots working on day ‘d’.

\( CA_{onDuty}[d] = CAS77[d] - TrainingCA[d] + \sum_{p \in P} Y[p][d] \)

\[ (1.17) \]

\( - (1 - FirstDayProductivity) \times NumCAFirstDays[d] \)

\( - (1 - LastDayProductivity) \times NumCALastDays[d] \)

\( \forall \ d \in MP, p.rank = 4 \)

\( IP_{onDuty}[d] \) is the effective number of 18/15 type IP pilots working on day ‘d’.

\( IP_{onDuty}[d] = IPS77[d] - TrainingIP[d] + \sum_{p \in P} Y[p][d] \)

\[ (1.18) \]

\( - (1 - FirstDayProductivity) \times NumIPFirstDays[d] \)

\( - (1 - LastDayProductivity) \times NumIPLastDays[d] \)

\( \forall \ d \in MP, p.rank = 5 \)

\( NRFO_{onDuty}[d] \) is the effective number of 18/15 type NRFO pilots working on day ‘d’.

\( NRFO_{onDuty}[d] = NRFOs77[d] - TrainingNRFO[d] + \sum_{p \in P} Y[p][d] \)

\[ (1.19) \]

\( - (1 - FirstDayProductivity) \times NumNRFOFirstDays[d] \)

\( - (1 - LastDayProductivity) \times NumNRFOLastDays[d] \)

\( \forall \ d \in MP, p.rank = 6 \)
PilotWorkDays[\(p\)] is the number of days, pilot \(p\) is working for particular month ‘M’.

\[
PilotWorkDays[\(p\)] = \sum_{d \in M} Y[\(p\)][d]
\]  
(1.20)

\(\forall \ p \in P\)

\(PilotWorkDays[\(p\)] \leq p.\maxDays\)  
(1.21)

\(p.\maxDays\) is the maximum number of days a pilot can work in a schedule

\[
\sum_{p \in P} \sum_{d \in PP} Y[\(p\)][d] \leq \left(\frac{nPostDays}{30}\right) \times 18 \times \left(\sum_{p \in P} 1\right) \times postPeriodWorkDayLimit
\]  
(1.22)

\(\forall \ p.\ scheduleType = 18\)

\[
\sum_{p \in P} \sum_{d \in PP} Y[\(p\)][d] \leq \left(\frac{nPostDays}{30}\right) \times 15 \times \left(\sum_{p \in P} 1\right) \times postPeriodWorkDayLimit
\]  
(1.23)

\(\forall \ p.\ scheduleType = 15\)

These constraints control the number of days schedule after the month. Number of days scheduled after month end is forced to under some range.

\[
\sum_{tp \in TP} \sum_{d2 \in Pd} X[\(p\)][d2][tp] \leq 1
\]  
(1.24)

\(\forall \ p \in P, d \in Pd, d2 \leq d \leq d2 + tp.lengthOn + tp.lengthOff - 1\)

\[
\sum_{d = -6}^{nDaysThisMonth+nPostDays} \sum_{tp \in TP} X[\(p\)][d][tp] \leq (1 - p.\workedBDayTourAlready)
\]  
(1.25)

\(\forall \ p \in P, tp.\ lengthOn = 8\)
A pilot can work only one 8 day tour in 4 month period. So this constraint takes care of 8 day tours. No pilot can have two 8 day tours in a schedule.

\[
\sum_{d=-6}^{0} \sum_{tp \in TP} X[p][d][tp] \leq \text{allow8DayTourIn Seam} \tag{1.26}
\]

\[
\forall p \in P, \text{tp.lengthOn} = 8
\]

\[
\sum_{p \in P} \sum_{d=(nPrevDays-1)}^{nDaysThisMonth} \sum_{tp \in TP} X[p][d][tp] \leq \text{Max8DayPercentAllowed} \tag{1.27}
\]

\[
\forall \text{tp.lengthOn} = 8, d + 8 - 1 \leq nDaysThisMonth, p.\text{scheduleType} = 18
\]

\[
\sum_{p \in P} Y[p][d] \leq \text{Max15DayPercentAllowed} \times \sum_{p \in P} 1 \tag{1.28}
\]

\[
\forall d \in MP, p.\text{scheduleType} = 15
\]

\[
X[p][\text{startDay}][tp] = 0 \tag{1.29}
\]

\[
\forall p \in P, tp \in TP, (\text{cmi}, \text{startDay}, \text{Duration}) \in ET, p.\text{cmi} = \text{cmi}, \text{tp.lengthOn} = \text{duration}
\]

\[
\sum_{tp \in TP} X[p][\text{startDay}][tp] = 1 \tag{1.30}
\]

\[
\forall (\text{cmi}, \text{startDay}, \text{Duration}) \in RT, p.\text{cmi} = \text{cmi}, \text{tp.lengthOn} = \text{duration}
\]

\[
Y[p][d] = \text{OnOrOff} \tag{1.31}
\]

\[
\forall p \in P, d \in Pd, (\text{cmi}, \text{startDay}, \text{Duration}) \in DD, p.\text{cmi} = \text{cmi}, d = \text{day}
\]

\[
Y[p][d] = 1 \forall tr \in TE, p \in P, d \in Pd, p.\text{cmi} = tr.\text{cmi}, tr.\text{.start} \leq d \leq tr.\text{start} + tr.\text{duration} - 1
\]

12
DaysOffScore is the measure of the requested day off’s granted by the scheduler.

$$\sum_{tp \in TP} X[p][tr.\ start][tp] \geq 1$$ \hspace{1cm} (1.33)

$$\forall tr \in TE, p \in P, p.\ cmi = tr.\ cmi, tp.\ lengthOn \geq tr.\ duration$$

$$X[p][tr.\ start][tp] = 1$$ \hspace{1cm} (1.34)

$$\forall tr \in TE, p \in P, p.\ cmi = tr.\ cmi, tp.\ lengthOn = tr.\ duration, tr.\ duration = 5$$

$$daysOffGranted[req] \leq 1 - \left(\frac{\sum_{d=0}^{\text{req.\ duration} - 1} y[p][\text{req.\ start} + d]}{\text{req.\ duration}}\right)$$ \hspace{1cm} (1.35)

$$\forall \text{req} \in OR, p \in P, \text{req.\ cmi} = p.\ cmi$$

$$DaysOffScore = \sum_{\text{req} \in OR} \sum_{p \in P} daysOffGranted[\text{req}] \times \text{req.\ score}$$ \hspace{1cm} (1.36)

$$\forall \text{req.\ cmi} = p.\ cmi$$

Short tours are not very productive. Tour starting day and ending days are considered as a fraction of a day. Tours below 6 days are considered to be short. In order to reduce the number of short tours, a penalty is added for each day in a short tour. ShortTourScore is the sum of penalties for all the short tours in the schedule.

$$\text{Sum} = \sum_{d \in Pd} \sum_{tp \in TP} x[p][d][tp] - \sum_{tr \in TE} 1$$ \hspace{1cm} (1.37)

$$\forall 1 \leq d + tp.\ lengthOn - 1, d < nDaysThisMonth, tp.\ lengthOn > 6, tr.\ duration > 6,$$

$$1 \leq tr.\ start + tr.\ duration - 1, nDaysThisMonth - tr.\ start - 1 > 6$$

$$\text{shortTourScore} = \sum_{tr \in LR} \sum_{p \in P} (\text{Sum} \geq 1) \times (-\text{lr.\ score})$$ \hspace{1cm} (1.38)

$$\forall \text{lr.\ cmi} = p.\ cmi, \text{lr.\ lengthSpec} = 1$$
Long tours are more productive than short tours. Tours which are 6 days or longer are considered to be long tours. So in order to reduce short tours and increase long tours, a bonus is added whenever a long tour is scheduled. LongTourScore is the sum of the bonuses added for scheduling long tours.

\[
LongTourScore = \sum_{lr \in LR} \sum_{p \in P} (SumLong1 \geq 1 \& SumLong2 = 0) \times lr.\ score
\]

\[
\forall lr.cmi = p.cmi, lr.lengthSpec = 2
\]

\[
SumLong1 = \sum_{d \in Pd} \sum_{tp \in TP} X[p][d][tp]
\]

\[
\forall 0 \leq d + tp.lengthOn - 1 < nDaysThisMonth, tp.lengthOn = 6
\]

\[
SumLong2 = \sum_{d \in Pd} \sum_{tp \in TP} X[p][d][tp] = 0
\]

\[
\forall 0 \leq d + tp.lengthOn - 1 < nDaysThisMonth, tp.lengthOn < 6
\]

\[
longPeriodOff[p][d] \leq \sum_{tp \in TP} \sum_{d2 \in Pd} X[p][d2][tp]
\]

\[
\forall br \in BR, p \in P, d \in MP, br.cmi = p.cmi, d2 + tp.lengthOn = d
\]

\[
longPeriodOff[p][d] \geq \sum_{tp \in TP} \sum_{d2 \in Pd} X[p][d2][tp] - \sum_{tp \in TP} \sum_{d2 \in EMP} X[p][d2][tp]
\]

\[
\forall br \in BR, p \in P, d \in MP, br.cmi = p.cmi, d \leq d2 \leq d + 5
\]

\[
balancedTourScore = \sum_{br \in BR} \sum_{p \in P} \sum_{d \in EMP} (-longPeriodOff[p][d]) \times br.\ score
\]

\[
\forall br \in BR, p \in P, d \in MP, br.cmi = p.cmi, d \leq d2 \leq d + 5
\]
Balanced tour is a preference opted by pilot if he wants no span of day off’s greater than 5 days. If the pilot prefers long day off spans, then he opts for imbalanced tour. balancedTourScore is the penalty added for not granting the balanced tour preference. imbalancedTourScore is the bonus added for scheduling dayoff’s of more than 5 day span.

\[
\text{imbalancedTourScore} = \sum_{br \in BR} \sum_{pep \in P} \sum_{dcmp} \text{longPeriodOff}[p][d] \times br.\text{score}
\]

\[
\forall br.\text{cmi} = p.\text{cmi}, br.\text{balance} = 2
\]

Weekend On/Off is a chosen by the pilot. If the pilot wants to work on weekends, then a bonus will be added to WeekendOnScore if the preference is granted, similarly if the pilot wants day off’s on weekends, then bonus is added to WeekendOffScore.

\[
\text{WeekendOnScore} = \sum_{we \in WE} \sum_{pep} (\text{SumWeekend1} \geq 2 \& \text{SumWeekend2} \geq 2) \times wr.\text{score}
\]

\[
\forall wr.\text{cmi} = p.\text{cmi}, wr.\text{code} = 2
\]
Holiday scores are much similar to the weekend on/off scores.

\[
\text{weekendOffScore} = \sum_{w \in \text{WR}} \sum_{p \in P} (\text{SumWeekend1} \geq 2 \& \text{SumWeekend2} \geq 2) \times \text{wr. score}
\]  
(1.52)

\[\forall \text{ wr. cmi} = \text{p. cmi, wr. code} = 2\]

\[
\text{holidayOnScore} = \sum_{h \in \text{HR}} \sum_{p \in P} Y[p][\text{hr. day}] \times \text{hr. score}
\]  
(1.53)

\[\forall \text{ hr. cmi} = \text{p. cmi, hr. score} \geq 0\]

\[
\text{SumHoliday} = \sum_{h \in \text{HR}} \sum_{p \in P} x 
\]
\[x = \begin{cases} 
1, & Y[p][\text{hr. day}] = 0 \\
0, & Y[p][\text{hr. day}] = 1 
\end{cases}
\]  
(1.54)

\[
\text{holidayOffScore} = \sum_{h \in \text{HR}} \sum_{p \in P} \text{SumHoliday} \times (-\text{hr. score})
\]  
(1.55)

\[\forall \text{ hr. cmi} = \text{p. cmi, hr. score} \leq 0\]

Even though the shortTourScore is used to reduce the number of short tours, still three day tours are heavily penalized. Penalty is added for all the three day tours in the schedule. So threeDayScore is the sum of penalty for scheduling 3 day tours.

\[
\text{threeDayScore} = \sum_{p \in P} \sum_{d \in \text{MP}} \sum_{tp \in \text{TP}} X[p][d][tp] 
\]  
(1.56)

\[\forall \text{ tp. lengthOn} = 3\]

Even though the shortTourScore is used to reduce the number of short tours, still three day tours are heavily penalized. Penalty is added for all the three day tours in the schedule. So threeDayScore is the sum of penalty for scheduling 3 day tours.

\[
\text{pilotBonusPeriodScore} = \sum_{pb \in \text{PBP}} \sum_{p \in P} \sum_{d \in \text{Pd}} \sum_{tp \in \text{TP}} \text{pilotBonusPeriodCoef} \times X[p][d][tp] 
\]  
(1.57)

\[\forall \text{ p. cmi} = \text{pbp. cmi, d \leq pbp. startDay and pbp. endDay \leq d + tp. lengthOn - 1}\]

\[
\text{fleetBonusPeriodScore} = \sum_{fb \in \text{FBP}} \sum_{p \in P} \sum_{d \in \text{Pd}} \sum_{tp \in \text{TP}} \text{fleetBonusPeriodCoef} \times X[p][d][tp] 
\]  
(1.58)

\[\forall \text{ d \leq fbp. startDay and fbp. endDay \leq d + tp. lengthOn - 1}\]
Crewing score is the major contributor to the total objective. Crewing score is a piece-wise function with 3 break points. The breakpoints currently in use represent crews provided on any given day being 95%, 1.05% and 1.15% of the forecast crew demand. A large bonus is added to the objective function for each crew day added on a day when the crew total is under the forecast crew target, while only a small bonus is added to the objective for adding an additional crew on days when the crew total is over the forecast crew demand.

Figure 1.1: Piecewise Linear Function for Crew Score

\[
SC = \begin{cases} 
  \text{CrewTarget}[d] + \text{Slope}[1] \times \text{CrewsOnDuty}[d], & \text{if } \text{CrewsOnDuty}[d] \leq \text{Breakpoint}(1) \times \text{CrewTarget}(d) \\
  \vdots \\
  \text{CrewTarget}[d] \left( 1 + \text{Slope}[1] \times \text{BreakPoint}[1] + \sum_{j=2}^{i-1} \text{slope}[j] \times (\text{Breakpoint}[j] - \text{BreakPoint}[j-1]) \right), \\
  \text{if } \text{Breakpoint}[i-1] \times \text{CrewTarget}[d] < \text{CrewsOnDuty}[d] < \text{Breakpoint}[i] \times \text{CrewTarget}[d] 
\end{cases}
\]
PilotWorkDayScore is just a score for all the working days allotted in the schedule.
This is the mathematical formulation is solved using branch and bound method. Quality of the solution is determined by a value known as Gap Value.

\[
\text{Gap Value} = \frac{\text{Best Node Value} - \text{Best Integer value}}{\text{Best Node Value}}
\]

Gap value below 0.0001 is considered to be good solution. ILOG’s Optimization Programming Language (OPL) is used formulating the problem and CPLEX engine is used to solve the problem.
CHAPTER 2

DISCUSSION OF BLACK BOX OPTIMIZATION METHODS

2.1 Overview

In this section, a brief review of black box optimization methods are described together with the key aspect of these methods which is random solution generation, here called mutation. The set covering problem is a paradigmatic NP-hard combinatorial optimization problem which is used as model in relevant applications, in particular crew scheduling in airline and mass-transit companies (Marchiori E, Steenbeek A (2000)). The complicated nature of the crew scheduling problem makes characterization of the preceding formulation difficult. It is fairly clear that the problem is NP-hard. Solving black box optimization problems generally consists of two major steps (Bryan A. Norman & James C. Bean, (1998), T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002))

1) Heuristic algorithm to search for the solution (Helena R. Lourenço, José P. Paixão & Rita Portugal (2001))

2) Function to evaluate fitness and determine the quality of the solution (T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002), Jingpeng Li & Raymond S. Kwan (2005))

Heuristics like Tabu search, simulated annealing, Ant colony optimization and genetic algorithm etc, are very useful solving scheduling problems, set covering/set partitioning problems, travelling salesman problem etc.
2.2 Genetic algorithms for the bus driver scheduling problem

Genetic algorithms for the bus driver scheduling problem discusses about the application of hybrid genetic algorithm to solve complex bus driver scheduling problem. Hybrid genetic algorithm used for bus drive scheduling has five operators.

- Initialization
- Parent selection scheme
- Cross over
- Mutation
- Population replacement scheme

Bus driver scheduling is basically composed of candidate duties and the set of pieces-of-work.

```
Gene: 1  2  3  4  5  6  7  8  9  10 11 12 13 14
      1 | 5 | 3 | 2 | 1 | 4 | 1 | 0 | 2 | 3 | 5 | 2 | 4 | 0
```

Figure 2.1: Solution with fourteen pieces of work

This coding always represents feasible relaxed partitioning solutions. A fitness function is developed to evaluate the fitness of the solution.

Initialization of population is done using random number generator. Encoding of the solutions is done and the pool of solutions is generated. Parent selection is done using roulette-wheel selection.
The roulette wheel contains one slot for each population element. The size of each slot is directly proportional to its respective ps(i), and therefore population members with higher fitness values are likely to be selected more often than those with lower fitness values. The crossover operator is applied to pairs of selected chromosomes in order to generate one, two or more 'children'. The procedure is divided into two phases. In the first phase, a duty of one of the parents is randomly selected. If it is available, it is added to the chromosome under construction. The process is repeated until none of the duties in both parents is available. In the second phase the leftovers that still exist are reduced. Two types of mutation operators are developed.

- Basic Mutation
- Improve Mutation

Basic mutation is much like cross over operator. The process of selecting and inserting an available duty into the chromosome is repeated until there are no more available duties for that chromosome. In improve mutation operator, for each free piece-of-work; a duty is filled that also covers the same pieces-of-work of one of the adjacent duties. In population replacement operator, two types of replacement are developed. Generation replacement is the strategy adopted and the whole population is replaced. When the best element of the population is kept for the next generation, we get a generation replacement with elitism.

\[ ps(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \quad i = 1 \ldots n \]
This hybrid genetic algorithm has been very successful with airline crew scheduling problems (Jingpeng Li & Raymond S. Kwan (2005)).

2.3 Self-Adjusting Algorithm for Driver Scheduling

In paper Self-Adjusting Algorithm for Driver Scheduling, the major constraints and objective are:

- every piece of work is assigned to a shift
- the shifts must comply with all the operational constraints and labor rules;
- the total number of shifts is minimized
- the total cost is minimized

Now-a-days meta-heuristics are widely used for achieving near optimal solutions to NP-hard problems. Meta-heuristic are very efficient in searching solutions in a large pool of solution space. A new self adjusting algorithm was developed to solve the scheduling problems. In the first step, a greedy heuristic is used to evaluate all the possible legal shifts based on fuzzy subsets theory, and to decide which shift is going to be selected in the process of constructing a schedule. In this approach, each component has to prove continuously its worthiness to be a part of the solution (Jiefeng Xu, Milind Sohoni, Mike McCleery, & T. Glenn Bailey (2004)). So in every iteration, components are tested for worthiness and failure to have the worthiness discards the components from the solution. Discarded components are replaced with new ones. This replenishment of the components is done by a greedy heuristic, which deals with one component at a time, allowing those
components which make the maximum value to the objective. While replenishing the components, new components are always tested by a dynamic evaluation function, which checks for compliance of the new component with the already present components.

Self adjusting algorithms are more of a hybrid between population based search and local search (Jiefeng Xu, Milind Sohoni, Mike McCleery, & T. Glenn Bailey (2004)). Main features of these algorithms are the fitness-biased ‘hypermutation’, highly domain tailored algorithm which accounts for their success. The evolutionary element (i.e. the use of fitness-biased selection) comes mainly due to the mutation operator which enables highly disruptive mutations to occur. This helps in reducing the chances of convergence on to a local minimum and more movement towards global optimum. Self adjusting algorithm basically iterates through evaluation, selection, mutation and reconstruction. For one time at the beginning of execution an initialization step is performed. Initialization is the process of building a pool of feasible solutions similar to genetic algorithm. In evaluation, all the solutions are tested for fitness by a fitness evaluation function. Selection is the step in which components of the solution are tested for worthiness, and retained or discarded based on their worthiness. The goodness value is compared to \((p_s - p)\), where \(p_s\) is a random number in \([0, 1]\) and \(p\) is a constant. If goodness value is greater than \((p_s - p)\) then the component will survive the selection. Subtracting \(p\) from \(p_s\) improves the self adjusting algorithm’s capability of convergence (Jiefeng Xu, Milind Sohoni, Mike McCleery, & T. Glenn Bailey (2004)). Mutation is useful in avoiding the local optimal solutions. Unlike GA, mutation in self adjusting algorithm is done by random choosing the components and
replacing them, even after surviving selection process. Mutation needs to be maintained low in order to ensure convergence. In Reconstruction a partial schedule is taken as input and complete schedule is generated using refined greedy heuristics. Reconstruction acts more like a repair function which repairs the broken schedules by allotting some new components. If the components are not worthy enough, they will be automatically removed in selection step. Self adjusting algorithm was applied to a large scheduling problem of Regional Railways North East and successfully solved the schedule for a very good objective.

2.4 A dynamic neighborhood based tabu search algorithm for real-world flight instructor scheduling problems

A dynamic neighborhood based tabu search algorithm for real-world flight instructor scheduling problems (Helena R. Lourenço, JoséP. Paixão & Rita Portugal (2001)) is a paper dealing with an optimization problem to schedule the instructors to teach a set of pilot training events. Tabu search is used to solve this multi-objective flight instructor scheduling problem (MOFISP) at a major US air carrier that represents such a task. Tabu search is known to solve the hard combinatorial optimization problems with single objective function (Helena R. Lourenço, JoséP. Paixão & Rita Portugal (2001)). An innovative feature of tabu search known as Adaptive memory procedure is used to search the pool of solutions. This new feature restricts the search function from revisiting the solutions previously explored and thus avoiding local optimum solution. A new TS
algorithm based on the dynamic neighborhood search is developed employs some multiple mapping functions that are systematically interchanged during the search process (Helena R. Lourenço, José P. Paixão & Rita Portugal (2001)). One solution is generated using the random number generator, which is set as the current best solution. Dynamic neighborhood around the solution is analyzed by the mapping functions and the best distinct moves are noted. All the distinct moves are analyzed and the tabu memory is updated. New best solution is set to current best solution. Search is continued until a near optimal solution is reached or maximum number of iterations is reached. Tabu search is effective in solving multi objective scheduling problem.

One of the major advantages of genetic algorithms over these other algorithms is that it is parallel because they have multiple offspring thus making it ideal for large problems where evaluation of all possible solutions in serial would be too time taking, if not impossible (T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002)). Crew rostering problems for airlines are very large and have a large space of solutions. So algorithms like tabu search, ant colony optimization are slow in evaluating the solutions compared to genetic algorithm (Damon Cook (2000)). Genetic algorithms perform well in problems where the fitness function are complex, discontinuous, noisy, changes over time or has many local optima (Jiefeng Xu, Milind Sohoni, Mike McCleery, & T. Glenn Bailey (2004)). Genetic algorithms proceed without much understanding of the problem and are depended on the fitness function.
CHAPTER 3

RANDOM KEYS GENETIC ALGORITHM WITH LOCAL SEARCH

In this section, a method based on the elitist genetic algorithm with Bernoulli cross-over in Hadj-Alounane and Bean (1997) is described (Bryan A. Norman & James C. Bean, (1998)). This heuristic was selected with some admitted arbitrariness and it is motivated by the computational results that follow. The local search addition that is proposed here is also described in the context of a so-called “post tournament selection” phase which is optional.

3.1 General presentation

Genetic algorithms (GA) are a class of evolutionary algorithms that try to mimic biological ideas of evolution such as inheritance, mutation, selection and crossover (T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002)). GA was first introduced by Holland and since then a lot of research went in development of the algorithm. GA searches for solution based on a population of solutions unlike other heuristics like Tabu Search, Ant Colony Optimization etc (T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002)). GA is generally implemented in following steps: Initialization, Selection, Reproduction and Termination. GAs solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. These strings are called chromosomes. A GA starts with a schedule consisting of a string for each pilot. A pool of schedules are generated using random numbers which are later filtered for feasibility. All the feasible schedules are evaluated using fitness testing.
function sorts the solutions in the order of decreasing fitness. Parents are selected from the pool of solutions based on different methods like random selection, roulette wheel selection and tournament selection. Selected parents are crossed over to generate a second generation population of solutions. There are several ways to mate two schedules: One-point crossover, Two-point crossover, "Cut and splice", edge recombination, and more. Children generated from mating are evaluated for the fitness. Children with better fitness replace the parents for the next iteration pool. While crossover creates new individuals by recombining two or more parents, the mutation replaces certain percent of parents. Mutation pool is developed similar to the initial population. There are many ways of implementing selection, reproduction, cross-over and mutation. We utilized a genetic algorithm based on random keys representation, elitist reproduction, Bernoulli crossover and immigration type mutation for solving the crew rostering problem.
3.2 Random keys Genetic algorithms (RKGA)

Because of the difficulty in using branch-and-bound technique for crew rostering problem, we modified random keys genetic algorithm (RKGA) applied it to solve the crew rostering problem. The random keys genetic algorithm approach is extended to crew rostering problem by extending the encoding. Each pilot’s schedule is an encoded string.

3.2.1 Random Keys Coding Scheme

The concept of “random keys” was apparently first described in (GIVE REFERENCE, possibly to Norman…). A set of day tours and day off’s for each pilot in a scheduling period is called a string. A string for each pilot in the fleet makes one schedule. Each string is encoded using two units, Day off’s number and Tour pair number. Day off’s number is the number of days following a tour and its rest period. Day off’s number is a number ranging from 0 to 6. Tour pair number is the number ranging in between 1 to 6, which gives the tour pairs that needs to be scheduled.

![Figure 3.2: Random Keys Coding Scheme](image)
In the above example ‘1’ represents the number of day off to follow after a preceding tour and its rest period. So one day off is followed after a 4 day tour. As ‘6’ represents three day tour, we schedule a three day tour after the day off.

\[
\text{Day off's Number} = ((6 - 0) \times \text{Random number} + 0)
\]

\[
\text{Tour Pair Number} = ((6 - 1) \times \text{Random number} + 1)
\]

First day off’s number and tour pair number are generally adjust to fit in accordance with the seam period data.

3.2.2 Random Keys Genetic Algorithm (RKGA) Operators

Our approach is based on the random keys encoding of the string. The random keys representation encodes a solution with random numbers, which can be decode later to get the solution. The operators in RKGA are

- Initialization
- Elitist reproduction
- Bernoulli crossover
- Immigration
- Post-tournament selection

**Initialization**

Suppose a fleet has ‘m’ 18/15 day pilots to be scheduled. We generate a pool of ‘n’ schedules randomly using the random number generator and encoding. Generated pool of
solutions are evaluated for fitness using a fitness evaluating function and ranked according to their fitness value.

**Elitist Reproduction**

Unlike roulette-wheel selection or tournament selection, Elitist reproduction is different way of selecting the population for next generation. In elitist reproduction, certain percent of the top fitness valued schedules are passed on to next generation. This may result in convergence of solution on to a local optimal value. Therefore certain percentage of Mutations are passed on to next generation. In the ‘n’ schedules we have, we chose the top ‘p’ schedules and pass them onto next generation of ‘n’ schedules. Out of the remaining n-p schedules, ‘r’ schedules with the lowest fitness value are replaced with mutations in the next generation. The left ‘n-p-r’ schedules are filled up using Bernoulli crossover. Elitist reproduction called to generate a population of feasible solutions for a next iteration.

**Bernoulli cross over**

Bernoulli crossover also known as parameterized uniform crossover, is different than classic methods like traditional single-point crossover or multiple-point crossover. Suppose we randomly chose two schedules using a random number generator. Each schedule consists of ‘m’ pilots. Let the parents be Parent1 = \{P_1, P_2, P_3, \ldots , P_m\} and Parent2 = \{Q_1, Q_2, Q_3, \ldots , Q_m\}, where \(P_1, Q_1, P_2, Q_2, P_3, Q_3, \ldots , P_m, Q_m\) are random key alleles in these two schedules. Let \(Z = (Z_1, Z_2, \ldots, Z_m)\) be \(m\) independent uniform (0, 1) variates. Let the
children generated be Child1 = \{C_1, C_2, C_3 \ldots C_m\} and Child2 = \{D_1, D_2, D_3 \ldots D_m\}.

Let the probability to crossover be P. Children are determined as follows:

\[
\begin{align*}
C_i &= P_i \text{ and } D_i = Q_i, \text{ if } Z_i < P \\
C_i &= Q_i \text{ and } D_i = P_i, \text{ if } Z_i \geq P
\end{align*}
\]

Probability of crossover ‘P’ is not necessarily 0.5, which can be used to bias for a parent with better fitness value.

**Immigration**

Though some immigrants infiltrate into the pool of next generation schedules, mutations are deliberately added to the pool of schedules. Certain population of low fitness value schedules are replaced with mutations to ensure non convergence at a local maximum.

**Post tournament selection**

Post tournament selection is type of tournament selection, used along with the crossover operator. Two parents P_1 and P_2 are randomly selected from the pool of previous generation schedules. These two parents are crossed over to generate two children C_1 and C_2. The two offsprings are evaluated for fitness and the offspring with better fitness value makes it into the pool of next generation schedules.
3.3 Pseudo code for Random Keys Genetic Algorithm (RKGA)

1) Initialize population ‘n’. Evaluate fitness and rank the schedules according to the fitness. Set Check=1 and Count_Iteration=0.

2) While check=1
   a) Select first ‘m’ sorted schedules and copy them to next generation solution pool.
   b) Count_Iteration = Count_Iteration + 1
   c) For j=1 to n-m-k
      - Select two parents randomly
      - Perform Bernoulli crossover and generate offsprings.
      - Evaluate the offsprings with the fitness testing function. The offspring with better fitness moves onto next generation schedules and other offspring is discarded.
   d) Select last ‘k’ sorted schedules and replace them with immigrants.
   e) Evaluate fitness and rank the population
   f) If the population converging then check=0
   g) If Count_Iteration=Maximum_Iterations, check=0

3) Output final solution

Based on the random keys genetic algorithm, an innovative new adaptive feature is added.
Owing to the large structures, Crew scheduling problem may have many local optimal values (T. G. Dias, J. P. de Sousa, & J. F. Cunha (2002), Damon Cook (2000)). Due to the local optimal solutions, premature convergence occurs. Therefore to solve this problem a new step is introduced into RKGA. Whenever the GA reaches a local maximum and all the parents become equal eventually, then the mutation population is increased to ‘m’ times the original. This increases the probability of finding a new solution, eventually forcing the algorithm to move towards a better solution. Once a new solution is found mutation population is set back to its original size. This GA is named as Adaptive Local Search Random Keys Genetic Algorithm (ALSRKGA).

### 3.4 Solution Enhancer (Local Search)

In the particular crew scheduling problem, long tours are more productive than short tours. So an algorithm is developed which watches out small training tours like 3-5 days long and increases them to 6-7 days. This increases the average pilot productivity over a scheduling period. Solution enhancer also fits the rest period of a tour with a day off is present on the path. Solution Enhancer acts more like a hill climbing algorithm but works simultaneously genetic algorithm. But unlike hill climbing algorithm, solution enhancer runs before genetic algorithm. All the random key coded solutions are generated into feasible solution and fitness is tested. While make the solutions feasible, solution enhancer acts on the encoded data and generated a feasible solution with better objective value. Solution enhancer acts more like a local search.
3.5 Pseudo code for Adaptive Local Search Random Keys Genetic Algorithm (ALSRKGA)

1) Initialize population ‘n’. Evaluate fitness and rank the schedules according to the fitness. Set Check=0 and Count_Iteration=0.

2) While check=1

h) Select first ‘m’ sorted schedules and copy them to next generation solution pool.

i) Count_Iteration = Count_Iteration + 1

j) For j=1 to n-m-k

   - Select two parents randomly
   - Perform Bernoulli crossover and generate offsprings.
   - Evaluate the offsprings with the fitness testing function. The offspring with better fitness moves onto next generation schedules and other offspring is discarded.

k) Select last ‘k’ sorted schedules and replace them with immigrants.

l) Evaluate fitness and rank the population

m) If the population converging then

   - If the fitness values over 5 generations are same then Increase the Mutations size 3 times the original
- Decrease the Cross over
- If Current best solution = new best solution then set mutation back to original size

n) If Count_Iteration=Maximum_Iterations, check=1

3) Output final solution
CHAPTER 4

COMPUTATIONAL RESULTS

The random keys genetic algorithm was coded in VB and the tests were performed on a 2.4 GHz Pentium with 2 GB RAM. The algorithm was initially tested on A4 problem to ensure the correctness of the algorithm.

Minimize  \[ \sum_{i=1}^{30} iX_i^4 \]

Subject to:  \[-1.28 \leq X_i \leq 1.28\]

We chose A4 problem to test our genetic algorithm as the minimum value for this problem is known. Crew rostering problem is much complex than the A4 problem.

![Graph showing GA Objective value for A4](image)

Figure 4.1: GA Objective value for A4
After testing the GA, we tested the code on crew rostering problem. The solutions provided by the RKGA are satisfying, as they are ahead of their competitor ILOG’s Optimization Programming Language (OPL). ILOG OPL solves the same crew rostering problem as a mixed integer programming problem by branch and bound method using a CPLEX engine. The fitness function is a set of some hard and soft constraints. The fitness function is defined as:

Maximize: TotalObjective

Where TotalObjective is given as:

TotalObjective = daysOffScore x daysOffCoef + shortTourScore x shortTourCoef +
                longTourScore x longTourCoef x longTourCoef + balancedTourScore x
                balancedTourCoef + imbalancedTourScore x imbalancedTourCoef +
                weekendOnScore + weekendOffScore + holidayOnScore + holidayOffScore +
                crewingScore x crewingCoef + threeDayScore + pilotBonusPeriodScore +
                fleetBonusPeriodScore + pilotWorkDayScore

TotalObjective is the fitness value, which is calculated from the individual scores like daysOffScore, threeDayScore etc. Each score is given a bonus or penalty depending on the satisfaction of a preference requested by a pilot. These scores keep the schedule from far from being just feasible. They enforce longer tours, add penalty for short tours etc.
4.1 Real crew rostering test problem

We have chosen few of the fleets solved by ILOG OPL using a CPLEX engine. We have applied RKGA to these sample files and generated the schedule in compliance with the rules. The criteria used to compare the solutions were the TotalObjective value and the average number of pilots per day. A 1900 iteration run by RKGA generated a schedule with higher TotalObjective and better average number of pilots per day.

![Graph showing crew duties on a day for Nov 08](image)

**Figure 4.2: Crews on duty on a day for the month of Nov 08**

The above figure compares the number of pilots scheduled on any day by OPL and GA. Schedule by GA & OPL has objective values of 1,664,805 and 1,641,030. GA was run for 200 iterations. OPL was run for 2 hours.
Figure 4.3: Schedule for the month of Nov 08
4.2 RKGA Vs LSRKGA Vs ALSRKGA

Figure 4. 4: RKGA Vs LSRKGA Vs ALSRKGA seed 0
4.3 Two sample t-test for RKGA and ALSRKGA

Two-Sample T-Test and CI: RKGA, ALSRKGA

<table>
<thead>
<tr>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>RKGA</td>
<td>2</td>
<td>1660857</td>
<td>158</td>
</tr>
<tr>
<td>ALSRKGA</td>
<td>2</td>
<td>1663591.50</td>
<td>2.12</td>
</tr>
</tbody>
</table>

Difference = mu (RKGA) - mu (ALSRKGA)
Estimate for difference: -2735
95% CI for difference: (-4158, -1311)
T-Test of difference = 0 (Vs. not =): T-Value = -24.41 P-Value = 0.026 DF = 1

Since the p-value is less than 0.05, we find the significance. It is significant with p-value = 0.026 which is almost close to bonferroni value of 0.025.

4.4 Two sample t-test for LSRKGA and ALSRKGA

Two-Sample T-Test and CI: LSRKGA, ALSRKGA

<table>
<thead>
<tr>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSRKGA</td>
<td>2</td>
<td>1663010.50</td>
<td>4.95</td>
</tr>
<tr>
<td>ALSRKGA</td>
<td>2</td>
<td>1663591.50</td>
<td>2.12</td>
</tr>
</tbody>
</table>

Difference = mu (LSRKGA) - mu (ALSRKGA)
Estimate for difference: -581.00
95% CI for difference: (-629.38, -532.62)
T-Test of difference = 0 (Vs. not =): T-Value = -152.58 P-Value = 0.004 DF = 1
Since the p-value = 0.004 < 0.05 we find significance with \( \alpha = 0.05 \) and even with bonferroni value of 0.025. So ALSRKGA is better than LSRKGA

### 4.5 Two sample t-test for RKGA and LSRKGA

Two-sample T for RKGA vs LSRKGA

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>RKGA</td>
<td>2</td>
<td>1660857</td>
<td>158</td>
<td>112</td>
</tr>
<tr>
<td>LSRKGA</td>
<td>2</td>
<td>1663010.50</td>
<td>4.95</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Difference = \( \mu (RKGA) - \mu (LSRKGA) \)

Estimate for difference: \(-2154\)

95% CI for difference: \((-3577, -730)\)

T-Test of difference = 0 (vs not =): T-Value = 19.22  P-Value = 0.033  DF = 1

Since the p-value < 0.05 we find significance with \( \alpha = 0.05 \) but not with bonferroni value.

LSRKGA is better than RKGA.

### 4.6 One sample t-test for ALSRKGA and Branch and Bound Method

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
<th>95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and Bound</td>
<td>2</td>
<td>19827</td>
<td>158</td>
<td>112</td>
<td>(18404, 21250)</td>
</tr>
</tbody>
</table>

Since the confidence interval does not contain zero in it, RKGA is better than branch and bound method.
So based on these tests we can state that ALSRKGA is better than LSRKGA which is better than RKGA which is better than branch and bound method.

Figure 4.5: Boxplot for all the GA algorithms and OPL
CHAPTER 5
CONCLUSIONS

In this research we modified and utilized a robust kind of genetic algorithms specially designed to solve the scheduling problems known as Random keys genetic algorithm (RKGA). RKGA uses a new coding scheme for generating a schedule for the pilots. The operators in RKGA are initialization, elitist reproduction, Bernoulli crossover, immigration and post-tournament selection. RKGA is accompanied with a local search function. Local search function performs those changes which can improve the fitness of the solution. Random key genetic algorithm generated a schedule for the month of Nov 2008 for fleet 800XP with objective value SOME points more than the competitor software ILOG OPL. The results show that Adaptive Local Search RKGA produces solutions higher average pilots per day than RKGA and Local Search RKG for 500 iterations on 800XP fleet. We have proven statements about the mean values or the average objective values with the following caveats:

- the results are relevant for the 800 XP fleet of mid west airline company
- the results are 500 iterations only
- the results are proven with p values at α =0.05

Results show that ALSRKGA reduced 1500 iterations to reach the solution generated by RKGA. Increased mutation size ensures ALSRKGA from premature convergence. These features have to be further explored. ALSRKGA needs to be tested on more fleets. Further
study needs to be done on the effects on increasing the mutations. Results of two sample t-tests show that all the three approaches are better at solving the crew rostering problem of the Midwest airline company.
REFERENCES


Honors Undergraduate Thesis

Marchiori E, Steenbeek A (2000) *An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling.*