DEVELOPMENT OF HYBRID META-HEURISTIC ALGORITHM FOR SOLVING NP-HARD COMBINATORIAL OFFICE SPACE ALLOCATION (OSA) PROBLEM IN A NIGERIA UNIVERSITY

AGBOOLA, OLADIRAN MARTINS
[91/027648]

MAY, 2018

DEVELOPMENT OF HYBRID META-HEURISTIC ALGORITHM FOR SOLVING NP-HARD COMBINATORIAL OFFICE SPACE ALLOCATION (OSA) PROBLEM IN A NIGERIA UNIVERSITY

By

AGBOOLA, OLADIRAN MARTINS
(91/027648)
B.Sc (1997), M.Sc (2009) (UNILORIN)

A Thesis Submitted to the Department of Computer Science, Faculty of Communication and Information Sciences in Partial Fulfilment of the Requirements for the Award of the Degree of Doctor of Philosophy in Computer Science

Department of Computer Science
University of Ilorin, Ilorin, Nigeria

MAY, 2018

CERTIFICATION

This certifies that this thesis has been read and approved as meeting the requirements of the Department of Computer Science, University of Ilorin, Ilorin, Nigeria for the award of Ph.D degree.

Prof. J. S. Sadiku (Supervisor)	Date
Dr. D.R. Aremu (Head of Department)	Date
Dr. Tinuke O. Oladele (PG Coordinator)	Date
(External Examiner)	Date

DEDICATION

This work is dedicated to God almighty and my parents late Alhaji Atidade Bello Agboola and late Deaconess Omorinola Aweke Ruth Agboola. May their souls rest in peace. Amen.

DECLARATION

I Agboola Oladiran Martins, hereby declare that this thesis entitled Development of Hybrid Metaheuristic Algorithm for Solving NP-Hard Combinatorial Office Space Allocation problem in a Nigeria University is a record of my research work. It has neither been presented nor accepted in any previous application for a higher degree. All sources of information have been specifically acknowledged.

In addition, the research work has been ethically approved by the University Ethical Review Committee.

_____ _____

Agboola, O. M. Date

ACKNOWLEDGEMENTS

I give thanks to the Almighty God that make this wish a reality, thank him for seeing me through to him alone I return all glory.

My unquestionable appreciation goes to my supervisor Professor J.S Sadiku for his constructive criticism and correction of my research work and also the Head of Department Dr. D.R. Aremu.

I also use this medium to appreciate the Ag.Dean of Faculty of Communication and Information Sciences, Dr. R.A Jimoh, the Departmental Post-graduate coordinator Dr. (Mrs) C. Oladele, the immediate past Post-graduate coordinator Dr. (Mrs) C. O. Abikoye , Dr. Oladele, Dr. A.Baje (Young Prof.), Dr. Ameen and all other staff of the Department.

Special appreciation goes to Prof. Gbadeyan J.A of Department of Mathematic, University of Ilorin, for his encouragement since the inception (under graduate). Dr. Azeez and Dr. Issa of Library Department University of Ilorin, I thank you for your support, I also want to thank all my siblings, most especially Mr Agboola Olatunde for his tireless effort throughout the Programme.

My appreciation also goes to my wife, Mrs Agboola, Adenike Dorcas, my children, Agboola Fogofoluwa Daniel, Agboola Fowofoluwa Ruth and Agboola Fopefoluwa Abraham, for their endurance during the research work.

Appreciation goes to everyone that had been part of this success, the likes of Dr. Ajibola S. A of Osun state University, Osogbo and Dr. Asaju Laro, Mr Ayeni, J.T, Mr Seyi Fadeyi, Mr Samson Samprog, Mr Sayo, Mr Atolagbe, all staff and students of Computer Science Department Kwara State Polytechnic, Ilorin and many others. God bless you all.

ABSTRACT

Office Space Allocation (OSA) is a major problem in higher institutions of learning. As a result of this problem, most of the demanding entities (staff) are wrongly allocated. The problem of OSA is considered to be Non-Polynomial (NP)-Hard combinatorial optimization problem which has been attended to by different researchers in the field of Artificial Intelligence (AI) and Operations Research (OR). Due to its combinatorial nature, several methods have been proposed, which include mathematical, heuristic and meta-heuristic methods. Considering the various methods available, meta-heuristic algorithms in their combinatorial forms need to be developed and tested for solving OSA in Nigeria Universities. Since the hybridization of the meta-heuristic algorithms considered in this research is not yet in existence, this study aimed at developing a hybrid meta-heuristic algorithms of Tabu search and Artificial Bee Colony in solving OSA problems using University of Ilorin as a case study. The objectives of the study were to; (i) formulate a mathematical objective function model for OSA problem and calculate penalty weight; (ii) adapt the algorithms to the problem of OSA; (iii) hybridize Artificial Bee Colony (ABC) algorithm with Tabu Search algorithm to solve OSA problem; and (iv) evaluate the algorithms using Halstead's complexity measures.

The research adopted a five-phase method. These phases included collection of dataset from the Faculty of Communication and Information Sciences, University of Ilorin, as a sample for mathematical modelling for solving OSA problem in terms of the objective function and the constraints. The methodology phases were adaptation of Artificial Bee Colony, Genetic and Tabu search meta-heuristic algorithms for the OSA problem, hybridization of ABC and Tabu Search algorithms to enhance the performance of the allocation, and a comparative study of the hybrid algorithms using halstead's complexity measures.

The findings of the study were that:

i. *the ABCgave lower penalty weight of 1678.3 when compared with 3885, 4036.6 and 1838.3 of hybrid, Tabu and genetic algorithms respectively;*
ii. *when Tabu, ABC and genetic algorithms were adapted to the problem of OSA, the Tabu gave better result in term of time used. Tabu used 1231secs against 3114.8secs of ABC and 4256.3secs of genetic;*
iii. *the hybrid algorithm of Tabu and ABC gave better result when compared with the three algorithms in the second finding in term of time used to solve the OSA problem. The hybrid used 616.62secs against 1231s, 3114.8s and 4256.3s of Tabu, ABC and genetic respectively; and*
iv. *the halstead's complexity measure such as program vocabulary, program length, program volume, program intelligence and program difficulty were used to compare the performance of all the algorithms and the hybrid algorithm gave the best result.*

The hybridized meta-heuristic algorithm and mathematical model developed was effective in solving the OSA problem and the use of population based algorithm enhanced the performance in allocating all entities to their respective offices. The hybrid algorithm also outperformed other existing algorithms considering the time used and the penalty weight. The study recommended the use of more hybridized algorithms in solving the problem of OSA in Nigeria Universities.

TABLE OF CONTENTS

CHAPTER TWO: LITERATURE REVIEW

CHAPTER FIVE: CONCLUSION AND RECOMMENDATION

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

List of Abbreviations/Notations

OSA- Office Space Allocation

ABC- Artificial Bee Colony

(O/I)- Output / Input

ACO- Ant Colony Optimisation

NP- Non-Polynomial

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND TO THE STUDY

Space is an area occupied by or intended for a person or thing, which is regarded as one of the highly invaluable resources in any typical organization. The Office Space Allocation (OSA), therefore aims at making efficient and effective use of the spatial resources so that the misuse of space is minimized or eradicated. The duty of space allocation often include constraints as well as objectives on the preference of the specific organization.

In several institutions, a lot of people that use the available resources such as machines, rooms or spaces are organized into structural units for instance departments, where all the entities within the same organizational unit are expected to be placed close to one another (Adewumi and Ali, 2010). This is not an issue when such an organizational unit is large. However, in a typical university environment for instance, one will not expect that the rooms occupied by members of an engineering faculty to be within the same building with rooms occupied by members of the social sciences. This will enable the decomposition of the office space allocation problem or challengeinto smaller sub-problems (Ulker, 2013).

In the OSA problem, the main purpose is to optimize the efficient and effective use of space. There are usually at least two components of the misuse of spaces (Landa, 2003). The first component opines that each room is not expected to be used above its capacity. This is

because the problem of overuse occurs in an organization that has budget restrictions, whereby workers were asked to share the office space resources. This may force the employees to work or operate in limited space which can lead to reduced productivity.The second problem deals with the use of office space below its optimal capacity (Beyeronthy, 2009). Under-use of office space may constitute a serious financial burden as a result of unnecessary new building costs.

OSA and resources efficiency are of great importance in all institutions or organisations, from small organisations or companies to large multinational organizations (Awadallah, Khader, Al-Betar and Woon, 2012). Putting academic institutions into consideration, the distribution of the available space among staff and students as well as other resources such as labs, offices, lecture rooms and storage rooms is a process that has to be carried out constantly because of the changes that often occur in the environment for example office for new staff, or research students, new lecture rooms or labs, people leaving the institution, among others (Ulker, 2013).

The available office space is often restricted and an efficient as well as effective functioning of any academic institution depends a lot on, among other necessary factors, having a good distribution of the space (Burke, Cowling, Landa and Mecolumn, 2001). Efficient distribution should make sure that every demanding resource is given the minimum space required (Burke et.al, 2001). This ensures that the available space is utilized as efficient as possible so that the additional constraints ca be satisfied to a greater extent. An efficient utilization of space ensures that neither too much nor too less space is given to any particular resource than the required minimum. One of the constraints in space allocation requires that the available rooms meet certain conditions (i.e. senior lecturers and above must not share offices while research students should be allocated

close to their supervisor and lecture rooms should not be located close to a noisy area among others) (Landa, 2003).

OSA is regarded as space management problem encountered in many educational institutions (Beyeronthy, Burke, McCollum, McMullan, Landa and Parkes, 2009). The office space includes lecture halls and rooms dedicated for tutorials as well as seminars, workshops, and other purposes. The efficiency of OSA management is often measured by the utilization of spatial resources. Most of the time, the utilization is measured as the fraction of used office space over the total available space. Contrary to common perception, the utilization of office space in a lot of universities is quite low. The proportion of practical utilization was reported to be so low that it ranges between twenty to thirty percent in the developed world (Beyrouthy et al, 2009) while the percentage could not be ascertained in the developing world. A specific search on office space allocation lead to some principle as described by most universities in the developed countries (University of Michigan, 2012). There is no reference to any automated system, as it is mainly a manual process. In many universities, the governing body is responsible for the space allocation process placed under the works division. Requests for office space are usually made official and in a large office space restructuring request an extensive and bureaucratic assessement period is normally required. The governing body allocates or assigns the space to different faculties. It is then the responsibility of the faculty to allocate or assign space allocated to different departments under the faculty (Ulker, 2013).

The allocation or assignment of office space in any large organization/institution is normally a problematic issue, which normally requires a significant amount of time to perform manually (Varleys, 1998). The outcome of this allocation affects the life of whoever makes use of the space. The problem of space allocation affects almost everyone in some way or another, whether it is the size or layout of offices or work environment, limited parking space or even the organization of homes (Burke and Varleys, 1998).

This studywill use hybridized metaheuristic algorithm (TABC) to solve the problem of Office Space Allocation in a tertiary institution focusing on the time of allocation as area of interest.

1.2 Principles of Space Allocation

i. The space belongs to the institution and the provost is in charge of the allocation.

ii. Spaces are allocated based on the needs for various programme and priorities as determined by the dean.

iii. Certain quantitative metrics should be developed in order to evaluate the research space utilization and periodic checks should be put in place to examine the allocations.

iv. The space is allocated to research activities and not to individuals. Therefore it can be taken over by the university if there are changes in the research activity.

v. Vacant or under – utilized space should be re-claimed, re-assigned or re-purposed.

vi. Schools are allowed to subsidize research activities that do not generate sufficient costs related to the space usage.

vii. Optimal use of research space includes shared use of resources and facilities.

viii. Space allocations should be based on maximum utilization of the existing facilities.

ix. Space allocation should take into consideration health regulations and procedures. (Burke and Varleys, 1998)

1.3 Statement of the Problem

Studies have revealed that office space allocation is a major problem in most higher institutions of learning, to the extent that most of the demanding entities (staff) are wrongly allocated,some were not allocated, some offices were used above its capacitywhile some were used below its capacity. This research work intend to solve the problem of Office Space Allocation using three metaheuristic algorithms and hybridization

of two of these algorithmsto improve on each algorithm weakness of early convergence and infinite loop trap. Based on the past research work done in this OSAP area by different researchers, it is considered to be NP-Hard combinatorial optimization problem which has been attended to by different researchers in the field of Artificial Intelligence (AI) and Operations Research (Ulker, 2012). Due to its combinatorial nature, several methods have been proposed, which include mathematical approach (exact method), heuristic methods as well as meta-heuristic method (Landa et.al, 2012). Considering the various methods available, meta-heuristic algorithms are regarded as one of the best methods because of the shortcoming of heuristic methods involving early convergence (Landa, 2003).

Search on office space allocation resulting in fair amount of guidelines,which are described by most universities in the world (Ulker, 2013). It is unfortunate that most of these universities only have guidelines on an automated system since space allocations in several universities involve manual process shouldered on the various units to allocate space assigned to them (Frimping and Owusu, 2005).The manual office allocation takes weeks or months to be completed. However, this research work intend to use hybridize population based meta-heuristic algorithm (Artificial Bee Colony) and local search based meta-heuristic algorithm (TABU Search) to solve OSA problem faster than manual and heuristic methods.

Landa (2003) suggested the fully automated system of OSA, that should be tested with a good range of data sets and also suggested further implementation of OSA using populated based metaheuristic. Ulker (2013) suggested a good mathematical model of the objective function and re-allocation of previous allocation due to modifications in entity, room structure and the constraints associated, which he recommended as future research work. These recommendations of Landa and Ulker form the bases of this study.

1.4 Aim and Objectives of the Study

This study aims at development of hybrid metaheuristic algorithm technique in solving NP-Hard combinatorial Office Space Allocation problems in a Nigeria University. The objectives of the study are to:-

i. formulate a mathematical model for OSA problem objective function and calculate penalty weight;
ii. Adapt ABC,Genetic and Tabu search algorithm to the problem of OSA;
iii. hybridize Artificial Bee Colony(ABC) algorithm with Tabu Searchalgorithm(TABU) to solve Office Space Allocation Problem;
iv. evaluatethe hybridized and the existing algorithms(TABU,GENETIC and ABC) using Halstead's complexity measure(program length, program volume, program intelligence, program effort, program size, execution time, Line of codes, number of bugs and program difficulty).

1.5 Justification of the Study

The research literatures reviewed shows that there are different methods used in solving the problem of OSA in higher institutions of learning, such as mathematical programming and metaheuristic algorithm (Ulker, 2013), integer programming (Landa and Ulker, 2011), asynchronous cooperative local search(Burke et al. 2007), harmony search algorithm (Awadallah et al., 2013), modified harmony search algorithm (Al-Betar, 2013), genetic algorithm and tabu search (Landa, 2003), hybrid particle swarm optimization (Remi, 2009), particle swarm optimization (Andeep et al., 2013), hill climbing and simulated annealing(Burke et al., 1999), pattern search and particle search(Luke, 2013), and Multilevel genetic algorithm (Adewumi, 2010) among others.

Since the problem of office space allocation lies within the scope of NP- complete and P ≠ NP, then the problem belongs to the set NP-P. All Np-complete problems are intractable. The complexity of a problem and the complexity of an algorithm to solve the problem from a computational point of view shows that an exact algorithm only has the capacity to solve a particular instance of a combinatory problem to optimality.

The time complexity of some exact algorithm is bounded by an exponential function, which makes these algorithms inefficient. The interest and the actual implication of the concept of NP- complete problem lies in the popular belief that an effective and efficient algorithm for solving such problem does not exist and the algorithm that produces high quality(or near optimal) solution in a considerable amount of time is needed (Ulker and Landa, 2012). In view of this assertion, this research work is aimed at solving the problem of office space allocation in tertiary institutions using hybrid population based algorithm (Artificial Bee Colony algorithm), Genetic algorithm(Ga) and local search based algorithm (Tabu algorithm).The OSA model provides basis upon which the office allocation can be deployed.

1.6 Definition of Operational Terms

Space Allocation – Allocation of available spaces (offices) to entities

Meta –heuristic – Optimization Algorithm for Exploration and Exploitation

Hard Constraint – Rules that must be satisfied at all time

Soft constraint - Rules that can be violated with penalty

Hybrid meta-heuristic – Combination of heuristic and metaheuristicalgorithms

Global optimum - Maximum solution in the search space

Minimal optimum - Minimum solution in the search space

Combinational Optimization - Non-polynomial optimization

Artificial Bee Colony- Population Based meta-heuristic algorithm

Genetic- Population Based meta-heuristic algorithm

Tabu search - Local search based algorithm

Heuristic - Optimization Algorithm for Exploitation

Office space allocation - Allocation of available office to entities

Halstead Complexity Measure- Established measure of comparison

1.7 Thesis Layout

The remaing parts of this study is as follows:

Chapter two discusses the literature review of the research work while, chapter three discusses the methodology used in carrying out the research work. Chapter four is the discussion of the result and the output while chapter five include the summary, conclusion and recommendations for future research work on the Office Space Allocation problem.

CHAPTER TWO

REVIEW OF RELATED LITERATURE

2.1 Introduction

Optimization is an important field of study because of its relevance to many areas of life. It has gained a lot of attention from people in academics, expecially in recent years. Fields like Computer Science, Mathematics, Economics and many more. The focus of optimization is to use its techniques in order to determine realistic solutions to optimization problem, the optimal problem objectives are determined by the solution. Feasible or realistic solutions are found from a set of feasible solutions that can be found within the solution space, which are subject to the constraints associated with the problem. Seeking optimal solutions are always desired of the reseachers in solving real-world optimization problems. For instance, consider the importance of finding an optimal solution in running an organisation or a business. In running a business, the business owner will focus on maximising profits, minimizing costs and be able to do so in the least amount of time possible. Therefore finding an optimal solution to this problem permits the business owner to manage the business effectively.

Definition 2.1 (Ariyo, 2013) contains a formal definition of optimisation

Definition 2.1.1: Let $f: A \rightarrow \mathbb{R}$represent an objective function. $A \subset \mathbb{R}$ represents a set of feasible solutions that lie within a solution space of real numbers $\mathbb{R}$. Let $x_0 \in A$. The goal of an optimization problem is to find $x_0 \in A \subset \mathbb{R}$, such that;

$$f(x_0) \leq f(x) \text{ for all } x \in A. \text{ Minimum solution}$$

$$f(x_0) \geq f(x) \text{ for all } x \in A. \text{ Maximum solution}$$

$f(x_0)$ that either minimizes or maximizes f is called an optimal solution.

Optimal solutions are often found within neighbourhood structures of a solution space.

The definition 2.2 (Blum and Roli, 2003), therefore gives the definition of a neighbourhood structure.

Definition 2.1.2: Let $:S \rightarrow 2^S$ represent a neighbourhood structure. If $s \in S$, then $(s) \subseteq S$ represents the neighbourhood of s.

Optimal solutions which are found within local neighbourhood structures of a solution space often stand for the local optima. These local optima can either be the local minimum or maximum solutions. A solution space consist of several local optimal solutions, andthe best local optimal solution among them is a global optimum. Global optimum solutions are local optimal solutions, but local optimal solutions are not necessarily global optimum solutions. The definitions of a local minimum, local maximum, global minimum and global maximum solution are given in definitions 2.3, 2.4, 2.5 and 2.6 respectively (Ariyo, 2013).

Definition 2.1.3: Let f represent an objective function and $\mathbb{R}$a solution space of real numbers. A local minimum exists at a point $x^* \in \mathbb{R}$, if there exists some value X > 0 such that

$$f(x^*) \leq f(x) \text{ subject to } |x - x^*| < X \text{ for all } x \in \mathbb{R}$$

Definition 2.1.4: Let f represent an objective function and $\mathbb{R}$a solution space of real numbers. A local maximum exists at a point $x^* \in \mathbb{R}$, if there exists some value > 0 such that

$$f(x^*) \geq f(x) \text{ subject to } |x - x^*| < X \text{ for all } x \in \mathbb{R}$$

Definition 2.1.5: Let f represent an objective function and $\mathbb{R}$a solution space of real numbers. A global minimum exists at a point $x^* \in \mathbb{R}$ such that,

$$f(x^*) \leq f(x) \text{ for all } x \in \mathbb{R}$$

Definition 2.1.6: Let *f* represent an objective function and ℝa solution space of real numbers. A global maximum exists at a point $x^* \in \mathbb{R}$ such that,

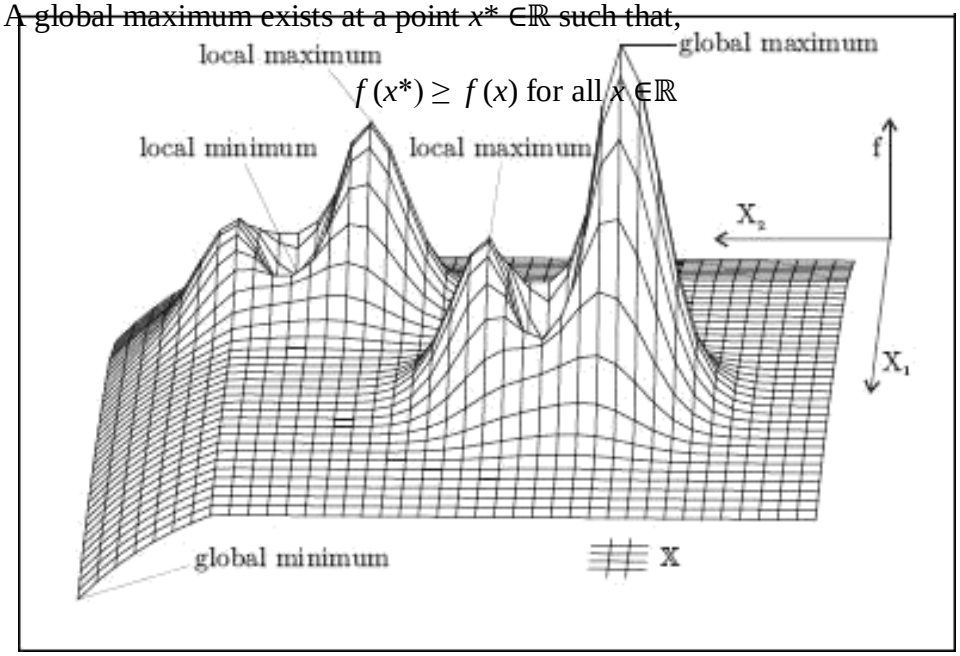

Figure 2.1: Local and Global optimal solutions of a two-dimensional function (Weise, 2009).

Figure 2.1 illustrates optimal solution points, found within the neighbourhood structures of a solution space. Global optimum solutions are the extremum solutions within the neighbourhood structure. Local optimum solutions are the best known solutions within the local neighbourhood structure of the solution space. Possible solutions to optimization problems are determined by evaluating the problems objective function *f*, subject to the restrictions of the problems constraints and controlling the decision variables as the inputs to *f*.

Global optimal solutions have been discovered for several real-world problems. The solution demands an exhaustive search of the solution space. If an algorithm exists that finds a global optimum solution, within polynomial time (***P***), the solution will be considered deterministic and the solution will be traceable. Deterministic solutions have direct relationships with the objective functions decision variables as well as the results obtained, i.e. specific inputs to the objective function will produce the exact same solutions every time.

Time complexity is not usually considered important if global optimal solutions are found within ***P***. On the contrary, if the time complexity increases exponentially as a result of the increase in the complexity of finding the global optimal solution, then time complexity will becomes a tangible factor. If only exponential time algorithms exist for determining the global optimal solution, the problem is considered intractable and is non-deterministic polynomial (***NP***) (Silva, 2003). When performing a comprehensive search of the solution space, solution is not feasible, accepting estimated or approximated near-optimal solutions is accepted. Near-optimal solutions are found within ***P***.

Time complexity determines the techniques employed to find solutions to optimization problems. There are two main methods used in solving the problem. The two types are the exact method and the heuristic algorithm method. Both types of methods find solutions to optimization problems within ***P***.Exact method algorithms exhaustively search the solution spaceso as to find global optimal solutions. Exact method algorithms don't take into cosederation the computational complexity involved in performing an exhaustive search. Thus, they are not suitable for ***NP*** problems and cannot provide solutions to many real-world problems. Examples of exact methods include linear programming, dynamic programming, branch and bound techniques.

Heuristic otherwise referred to as approximatemethod algorithms present near-optimal solutions to optimization problem. Near-optimal solutions are accepted in a situation where are no exact method algorithm in existence to provide global optimal solutions within ***P.*** Near-optimal solutions are somewhat inferior solutions, that trade accuracy for a significant reduction in computational time complexity. They are decision algorithms and they make use trial and error techniques for determiningpracticable solutions. Heuristic algorithms was used to provide near-optimal solutions to an NP-Hard Combinatorial Optimization Problem (COP), which is the subject of this research. It uses either local and

global heuristic search methods or techniques.

The rest of this section gives more detailsabout certain important background information concerned with addressing optimization problems. The information includes discussing the categorisations of optimization problems, heuristic and metaheuristic methods or techniques as well as local and global search heuristic algorithms.

2.2. Classification of Optimization Problem

There are many ways of classifying optimization techniques. To start with it can be classified according to the problem constraints, the nature of the equations and the number of objective functions. It may also be classified according to the deterministic nature of the problem and the type of decision variables used, amongst others (Kumar, 2011). There is no single optimization technique that can solve all types of optimization problems. This shows that some classifications of optimization techniques may perform effectively for some classes of optimization problems yet perform poorly (or may not even be applicable) for others. The classifications are briefly described and discussed below.

2.2.1 Classification Based on Constraints

Constraints can be reffered to as the restrictions applied to an objective function *f*. They define the bounds within which feasible solutions are found. There are both hard constraints and soft constraints. On the one hand, the hard constraints are constraints that must be enforced and cannot be broken. On the other hand, soft constraints are those which may be compromised. In order to find feasible solutions, all hard constraints must be satisfied and as many soft constraints as possible need to be satisfied as well. Grouping based on constraints fall into two categories. These are unconstrained optimization problems and constraint optimization problems.

i. **Unconstrained Optimization Problems**: If there are no constraints governing the evaluation of *f*, the problem is considered an unconstrained optimization problem. If *k*

equal the number of constraints then $k = 0$.

ii. **Constrained Optimization Problems**: If there are constraints governing the evaluation of *f*, the problem is considered a constrained optimization problem. If *k* equals the number ofconstraint then $k > = 1$. Most real-world optimization problems are multi-constrained.

2.2.2 Nature of the Equations Involved

Based on the nature of the equations of the objective function *f* and its constraints, optimization problems are classified as linear, non-linear, geometric or quadratic programming problems (Kumar, 2011).

i. **Linear Programming Problem (LPP):** If *f* and the constraints governing it are linear functions of non-negative design variables, then the problem is an LPP. LPP is

mathematically represented as follows (Kumar, 2011):

Let,

f = objective function

$h_i(x)$ = equality constraints

$g_i(x)$ = inequality constraints

$x = \{x_1 x_2, \dots, x_n\}$ be the design variables

which maximizes

$$f(x) = \sum_{i=1}^{n} c(i)x\,(i)$$

subject to;

$\sum_{i=1}^{n} a(ij)x(i) = b(j) for\ j = 1,2,\dots,m$

$x(i) = \geq 0 \qquad for\ j = 1,2,\dots,m$

where c(i), *a(ij)* and $b(j)$ are constants

ii. **Non-Linear Programming (NLP):** If *f*or one or more constraints governing *f* are non-linear functions of the design variables, then the problem is NLP (Jain, 2003). NLP are the most common programming problems encountered. NLP is mathematically represented as follows (Jain, 2003).

f = objective function

$h_i(x)$ = equality constraints

$g_i(x)$ = inequality constraints

X = $\{x_1 x_2, \dots, x_n\}$ be the design variables

Then

Optimize

f(x)

subject to;

$h_i(x) = 0$ for i = *1,2, ...,m*

$g_i(x) \leq 0$ for i = *(m+1), ...,p*

iii. **Geometric Programming Problem (GMP):** If *function* and constraints governing it are expressed as polynomials of *x*, then the problem is a GMP (Kumar, 2011).

iv. **Quadratic Programming Problem (QPP):** These are maximization type NLP problems which have 'concaved' objective functions and linear constraints (Kumar, 2011).

2.2.3. Types of Optimisation Objective Function

Optimization problems may have single or multiple objectives that will need to be satisfied in order to obtain feasible solutions.

i. **Single-objective programming problem:** This type of problem occurswhere there

is only a single objective that needs to be evaluated, i.e. $l = 1$. An example can be finding the sum of two integers.

ii. **Multi-objective programming problem:** These are problems where more than one objective function needs to be simultaneously evaluated, i.e. $l > 1$. Most real world problems are multi-objective. According to

Kumar, (2011) the mathematical formulation is as follows:

Let

f = objective function

$g_j(x)$ = inequality constraints

Find x which simultaneously optimizes

$$fi(x) \qquad \text{for } i = 1,...,k$$

subject to;

$$g_i(x) \leq 0 \qquad \text{for i} = 1, 2, ...,m$$

2.2.4. Deterministic Nature of the Problem

The deterministic nature of optimization problems related to the computational time complexity involved in finding feasible solutions. Deterministic or non-deterministic algorithms can be used to find solution to optimisation problems.

i. **Deterministic Algorithms:** These are exact method algorithms. Examples include Divide and Conquer; Branch and Bound, amongst others.

ii. **Non-deterministic Algorithms**: These are heuristic algorithms, such as Hill Climbing (HC), Simulated Annealing (SA) and Tabu Search (TS), Artificial Bee Colony(ABC).

2.2.5 Type of Decision Variables Used

Decision variables are divided into two.The first refersto values taken from a real number system of values while the second deals with discrete values. Discrete values are

independent values which are found within a set of possible inputs. Optimization problems can be categorised as continuous or combinatorial optimization problems based on the decision variables used.

i. **Continuous Optimization Problems:** This category make use of subsets of real numbered values as the inputs to *f*. The decision variables used must satisfy the constraints associated with the problem.

ii. **Combinatorial Optimization Problem (COP):** This category make use of discrete values as inputs to *f*. These inputs also need to satisfy the constraints associated with the problem. COP's consist of many NP-Hard problems and near-optimal solutions are accepted by techniques such as heuristic techniques. Strategies used to find solutions to COP include 'constructive' and 'improvement' (local search) heuristic methods.

a. **Constructive Heuristics**: This techniques starts off with an empty solution set and iteratively adds solution elements to the set in a systematic way until a complete solution is determined. Constructive heuristic methods are very fast compared to improvement heuristic techniques. On the contrary, they generally provide inferior solutions compared to improvement heuristic techniques (Syam and Al-Harkam, 2010).

b. **Improvement (Local Search) Heuristics**: This technique starts off with an initial random solution, and therefore attempts to improve on this solution. The improvement is done by using trial and error techniques to search the solution space for improved solutions. Current solutions are often replaced by improved solutions during an iterative process.

COP is mathematically represented as follows Weise, (2009):

Let

f = objective function

$X = \{x_1, x_2, \ldots, x_n\}$

S = search space

D_i= variable domains for $i = 1, \ldots, n$

Then Optimize

f such that $f: D_1 x\ D_2 x \ldots x\ Dn \rightarrow R^+$

The set of all possible feasible solutions are;

$S = \{s = (xi, vi) \mid \forall\ vi \in Di\ \}$ for $i = 1, \ldots, n$

Subject to *s* satisfying all the problem constraints

2.3. Search Techniques

2.3.1 Local Search Techniques

Local search techniques are refer to the algorithms that exploit the local neighbourhood structures of a particular solution space in search for the local optimum solution. Thus start off with an initial random solution, and iteratively make local changes within the local neighbourhood structures of the solution space in order to find improved feasible solutions. Local search techniques often try to determine the best neighbour surrounding the present solution. The problem associated with local search heuristic algorithms has been said to have prematurely converges. 'Intelligent' local search metaheuristic algorithms have built in functionality that reduces the risk of prematurely converging. A typical example of a local search heuristic algorithm is HC while examples of local search metaheuristic algorithms are SA and TS.

2.3.2 Global Search Techniques

It is possible to have many local optima in a particular solution space. Global search technique algorithms focus on determining the single best local optimum solution.It is very difficult to get global optima solution. There are several real-world problems which exist

where finding the global optimal solution still remains practically impossible (Silva, 2003).There is no single global search technique algorithm in existence that guarantee finding global optimal solution to all types of optimization problems. Thus, global search heuristic algorithms only attempt to estimate the global optimal solution from a set of local optima. An example of a global search metaheuristic algorithm is GA, ABC, Ant Colony.

2.4 Space Allocation Problem(SAP)

It has been discovered that one of the extremely difficult COPs' to solveSAP's. It deals with distributing limited amounts of available space amongst the demanding set of entities requiring space utilization. SAP is a very important managerial responsibility. Mismanagement in the way limited amounts of space utilized which impacts on the total operations of an organization negatively. Its effects are inefficient use of the limited available space, which may affect the general costs involved in operating the organization, and other ones. Categories of SAP's consist of bin-packing, resource allocation, knapsack problems, among others.This section explains SAP's at tertiary institutions. It illustrates a number of the problems and the complexities which are related to the issue of addressing space utilization. This overview will provide a more enhanced understanding to SAP's at tertiary institutions. It is suitable in order to introduce the OSAP presented in the methodology. This section also gives the multiple-knapsack mathematical model that can be used to mathematically formulate the proposed OSAP.

2.4.1 Space Allocation Problems at Tertiary Institutions

Space utilization at tertiary institution is a very relevant problem. It focuses attention on allocating limited available space amongst demanding entities which require space utilization. Demanding entities can be categorized into staff members, lecture venues, students demanding on-campus accommodation, among others (Silva, 2003). Therefore, it is not an easy task. Space allocation has to be done in ways that provide the highest level

of satisfaction to all demanding entities involved. In the process of doing this, all hard constraints must be satisfied and as many soft constraints and objectives must also be satisfied as much as possible. Example of allocating space include allocating adequate room spaces to members of staff, allocating adequate room spaces for lectures, and allocating as many eligible students on-campus accommodation from those that require accommodation, amongst others.

Mismanagement of available spaces at tertiary institutions negatively impacts on the overall cost andoperation of the institution.This is very important that space be utilized effectively. However, it is difficult to find optimal solutions in the way space is utilized.According to Silva(2003), the problems can get further complicated when considering the dynamic nature in which these organizations are managed. For example, entities can be added or removed.

In determining solutions to the problems of space allocation the convenience of the entities must also be considered. For instance, departments in a faculty should be allocated as close as possible to lecture venues.In addition to this, the health conditions of physically challenged students as well as other issues should be taken into consideration, before decisions are made.

These problems make space allocation to become a very important managerial responsibility. In order to establish effective solutions, automated systems need to be employed. Automated tools are time efficient and they provide more accurate solutions. However, a lot of institutions, especially in developing countries, still rely on using manual processes in dealing with space allocation at tertiary institutions.

i. Manual Approach to Space Allocation at Tertiary Institutions

The manual approach of space allocation at higher or tertiary institutions is done in the following ways: (Burke and Valley, 1998): First approach is that, top level management will decide on how the available space will be allocated to the different demanding entities. Such demanding entitiesconsist of various departments, faculties, and more. The moment the allocation is done, management within these various entities also decide on how to distribute the assigned space,and the space will be distributed amongst the various demanding entities that fall under them. This distribution should be done in a fairly ways, and it should take into consideration the many constraints or restrictions that could be associated with allocating space. Some of the constraints or restrictions include taking into account issues like the different sizes of the available space, the locations of the space, among others. Allocations should be done in a way that is as convenient as possible to all demanding entities involved.

The procedure for allocation allocation is not easy, as this may be attempted a number of times before a final solution will be determined. Futhermore, the final solution may not be an optimal solution. On the other hand, space allocation may be determined relatively quickly in smaller organisation. While for larger organizations it may be quite more complicated. This is as a result of the larger sizes of the input data sets, and the complexities of the constraints and objectives associated with providing solutions.

Mathematical models that can be used to model SAP's include bin-packing, resource allocation and knapsack modelling. These models are used to provide more accurate solutions to SAP's mathematically. This research employs a branch of a knapsack model called 0-1 multiple-knapsack. It will be used mathematically to model the various stages of the OSAP as presented in methodology.

2.4.2 Office Space Allocation

Office space allocation problem (OSAP) is a combinatorial optimization problem, solving many versions of office space allocation system may consume heavy amount of processing power. However, they can be proficiently implemented in terms of memory requirements. The time complexity concept includes decision and optimization problem which include the classes of P, NP, NP-complete, NP-hard and no free-lunch theorem.

Theoretically, problems associated with office space allocation are multi-dimensional Knapsack, bin packing and generalized assignment.

2.5.Related work

Bolaji, Micheal and Shola (2017) presented an adaptation of ABC algorithm in solving Office Space Allocation problem. The adaptation involved integration of three neighbourhood operators with the components of the ABC algorithm. The researcher used benchmark instances established by University of Nottingham dataset to evaluate the proposed ABC algorithm. ABC produced a good quality solution in comparison with the state-of-the-art methods. The program iterations can be trapped in an infinite loop.

Ulker (2013) proposed solving office space allocation problem using mathematical programming and meta-heuristic algorithm which present recent, efficient and effective research work in OSA. The objective of this work is to investigate the proposed solution method that can be used in automated Office Space Allocation Problem.

The work analyse the nature of the problem from the perspective of space misuse constraints and general objective function. The work also developed binary integer programming model based on types of constraints or restraints and weighted objective function. It investigates the possible benefit of utilizing room and floor relationship while developing the model.

The aim of his study is to combine mathematical programming model using integer linear programming based on definition of different constraints and additional variables and meta- heuristic algorithm.

The contributions of this research to OSA are the analysis of the OSA problem, formulation of a new parameterized data instance generator, mathematical programming model and meta-heuristic approach. These are used in order to extend the state of art in this area. The work only based its comparison on linear programming and one metaheuristic algorithm.

Al-Betar et al. (2013) used a modified Harmony Search Algorithm (MHSA) technique in Solving Space Allocation Problem. The research focused on modifying the Harmony Search (HS) algorithm and adapted it for the office space allocation. The modification includes two harmony search operators which are:

(i) memory consideration, where the global – best concept of particle swam optimization is borrowed and employed ; and

(ii) the pitch adjustment was designed to be a local search agent with three effective neighbourhood activities.

The modified harmony search algorithm was evaluated using three dataset from Nottingham and Wolverthampton Universities, where two new best results were obtained and a comparable result for the third dataset that was out preferred to the techniques of integer programming and mathematical programming, earlier proposed by Ulker and Landa (2010, 2011). The MHSA is an extension of the HS earlier proposed by the same researcher. This research work will experience early convergence because of the local search agent used.

According to Ulker and Landa (2012), a Local search algorithm was presented to solve the problem of office space allocation. The evolutionary component of the algorithm include standard crossover and mutation operators and a relatively small population of individuals where the offspring produced by the evolutionary operators that are subjected to a short but intensive local search process. The algorithm was designed to produce a solution for the difficult and highly constrained combinatorial optimization problem (OSA). The research observed that the mutation rate (m) and the number of local search iterations (h) turned out to be the performance affecting factors of the algorithm low mutation rate and a small number of local search in iterations greatly improve the performance of the algorithm. The researchers concluded that the evolutionary local search algorithm is highly competitive to the mathematical programming model that was presented earlier for solving the office space allocation problem. When they were compared for more than half of instances, the evolutionary local search algorithm yielded better result for the same execution time. The work got stucked in local optimal because of the local search algorithm used.

Awadallah et al. (2012) made use of Harmony Search Algorithm (HSA) in solving office space allocation problem (OSA) subjected to the two constraints (hard and soft constraints). The harmony search algorithm (HSA) is a population based meta-heuristic inspired by a musical improvisation process where three (3) operators were used to generate the new harmony at each iteration (i.e. memory consideration, random consideration and pitch adjustment). This paper modifies the memory consideration operator to select from the best solution in the population during the search. The harmony search algorithm used in this research was evaluated using three datasets. The algorithm used has weak local search ability, therefore will be trapped in a local optimum.

Ulker and Landa (2011) designed complex office space allocation problem instances with mathematics programming approach which was developed to model and generate test instances for the difficult combinational optimization problem. An Output /Input (O/I)integer programming model was developed and a commercial integer linear programming (ILP) solver was applied. Based on this model, a test instance generator was also developed to further investigate the difficulty of OSA problem through systematic experimentation. The objective of this research is to minimize the rate of space misuse which can be overuse or underuse and the soft constraints violation through the effect of four constraints with different parameters of the Slack Amount (P) and Violation Rate (v). It was observed that important factors affecting the optimality proof difficult of the test instance, which was the differences between the Negative slack (N) and the positive slack (P) amount. This adjusts the overuse/ underuse of rooms respectively in the generated test instances. Although the study raises the slack space rate (s) and violation rate (v) which increases the percentage gaps but concluded that the effect was less prominent than the effect of N and P. This research is linear in nature and it has a single objective function.

Retyal (2010) studied the performance of a greedy search algorithm and a tabu search algorithm for generating high quality solutions to the office space allocation problem. The objectives are to maximize synergies in the organization, minimize the over usage of limited space and maximize the number of buildings and rooms that can be completely closed. The computational experiments showed that a tabu search algorithm generated higher quality solution than a greedy local search algorithm with the same computational budget. The result of the research experience early convergence.

Ulker and Landa (2010) proposed 0/l integer programming model to solve office space allocation problem with the goal of optimizing the space utilization while satisfying a set of additional requirements. The research ascertains whether setting some constraints (hard

or soft) has considerable impact on the difficulty of the OSA problem. The problem is that of having a set of rooms (Office, hall), set of entities (people, machine) and then allocate each of the entities to a room.

Each room has a capacity while each entity has a size, the research optimizes the space utilization by minimizing the space wastage and ensures satisfaction of specific conditions that establish prerequisitess for the way in which entities are to be allocated to rooms. The research considered six constraints. All allocation, same room, not sharing, nearby,capacity, away from and any of the constraints can be set as hard (must be satisfied) or soft (desirable to satisfy) in their formulation. When constraint is set as soft, minimizing its violation becomes an objective in the problem formulation. An 0/l integer programming model was developed (exact method) and solve the problem using PLEX. The researcher concluded that the most difficult soft constraint to optimize is the same room constraint and that setting all constraints as hard makes solving the problem unrealisable. The research work make use of exact method which did not put time into consideration.

Burke et al. (2007) researched into the use of asynchronous Cooperative Local Search concept in solving Office Space Allocation Problem in the universities and other organizations. Their objective was to use certain number of entities such as students, staff and equipment, which must be allocated into specific rooms, utilize the space to the maximum while satisfying number of hard and soft constraints. An asynchronous cooperative local search approach was developed, in which a population of local search threads cooperate asynchronously to find better solutions. This approach integrates a cooperation mechanism in which a pool of genes (part of solutions) is shared to improve the global search strategy. The research was implemented by extending four single solutions meta-heuristics (Hill – climbing, Simulated Annealing, Tabu Search and a hybrid

meta-heuristic) to population based variant, using asynchronous cooperative mechanism. In each case, the population-based approach performs better than the single solution, using comparable computation time. The research concludes that the Asynchronous Cooperative meta-heuristics developed, improve upper known results for a number of test instances. The research work experienced early convergence.

Landa (2003) Metaheuristic and Multi-objective approaches for Office space allocation. The research presented an investigation on application of metaheuristic techniques for solving the office space allocation problem in academic Institutions using four well known metaheuristic, Iterative Improvement, Simulated Annealing, Tabu search and Genetic algorithms. The researcher concluded that the metaheuristic algorithm perform better than other methods and proposed the use of populated algorithm for future research work. The research work will be trapped locally and will also experience an infinite loop.

Tanzila Islam and Zunayed Kauyl (2016) Design an Automated university time table generator using Tabu-search algorithm. This algorithm helps to generate a course schedule for university by analyzing the search space and averts in-essential exploration. It optimise this solution and keeps the list of recently visited area in the tabulist .The objective is to derive a suitable time tabling system for courses and exams with proper requirements to minimize violation of constraints and find a feasible solution. Tabu-search algorithm helps the proposed system to solve the problem within a reasonable time frame and gives a better solution than the manual system.

Frank and Alexandra (2015) proposed linear programming model to solve allocation of classroom spaces to various student groups. The research was aimed at using linear programming to solve both the problem of over-allocation and that of under-allocation considering the Premier Nurse's training College, Kumasi. The objective of the work is to find out how classroom space are allocated to the students of the college based on the

various programs and courses offered and also to develop a linear programming model to allocate the space for various programs and courses offered and also to develop a linear programming model to allocate the space for various programmers to ensure optimal use of the classroom.

Sandeep et al. (2013) discussed automated timetable generator using Particle Swarm Optimization (PSO) to solve the problem of course timetabling. The research has two objectives: to give a detail introduction to the topic of timetabling, particle Swarm Optimization (their methods and variations); the second objective is the application of Pso to the problem of course timetabling. The research work based its modelling on four main factors (teacher, courses, time solution and classroom) together with other teaching facilities. The combination of these four factors was defined as the particle position and each particle represents a solution group. The researcher stated that Eberchart (1998) proposed the inertial weight value concept and added an inertial weight value (w) to the original PSO algorithm. The inertial weight was used to balance the global search ability in order to boost the capability to locate the optional solution and convergence rate. The researcher discussed the application of inertial weight factor to type of PSO to solving the problem of timetabling. The work was used to reduce the computational compulsion of the timetabling and also designed particle concluding on the basis of time slot.

Luke (2013) applied three optimization algorithm to campus parking space allocation problem in the university, the research formulated and modelled solution for solving the car parking space allocation (CPSA) problem by applying heuristic and meta-heuristic techniques such as Genetic algorithm (GA), pattern search (PS) and particle swam pattern search. The research compared the result obtained by the optimization algorithm to the result obtained from the CPLEX software and an exact method used in solving CPSA problem. It was stated that the new technique is better. The research model caters for the

reserved and unreserved parking policies in the campus. An investigation of the mathematical model formulated was done to cater for the parking space policies of the institution. The researcher generated some variants of real world data which were used to evaluate the optimization model formulated and was concluded that optimization algorithm can effectively provide solution to the car park space allocation.

Ayachi (2010) presented a research work on Genetic Algorithm (GA) to solve space container storage problem in the port. The study included Regular container, Open side, open top, empty and refrigerated containers. The objective was to determine an optimal container arrangement which obeys customers' delivery deadlines, minimizes the re-handle operations of containers and reduces the stop time of container ship. The adaptation of the genetic algorithm to the container storage problem was detailed and the proposed approach was compared to a last-In-first-Out (LIFO) and was applied to the same problem and the proposed method yielded a better result.

Chieh-Yuan and Ming-Chung (2010) proposed a two stages Simulated Annealing (SA) algorithm to solve the problem of shelf space allocation in retailing stores. In retailing stores, divergent displaying strategies are said to directly influence customers purchasing decision and also the profit margin of a store. In most of the researches, items are allocated into shelf space based on product type similarity neglecting the affinity relationship between product categories, another argument for the need of the research work is that customer's purchase behaviour on product that is located at eye-level layer of shelf usually gets much more attention from customers than other layers.To solve this shelf allocation problem, a modified Simulated Annealing (SA) algorithm with better initial strategy was developed with the following objectives to:

i. Construct the product category affinity matrix ;
ii. Locate the shelf space of product category;
iii. Allocate the self space of product type ;

iv. Develop an optimum shelf allocation technique for retail store ; and

v. Adopt the Simulated Annealing (SA) algorithm.

The research aims at developing an efficient heuristic algorithm with initial solution setting that can increase the solution quality and reduce the convergence speed to solve multi-level self space allocation problem considering the affinity between product categories and important weight of shelf spaces.

Adewumi (2010) studied space allocation problem using a multi-level heuristic driven by genetic algorithm (GA) to solve the hostel space allocation problem under domain specific constraints. The research examined the sensitivity analysis of various genetic algorithm operators in order to establish the baseline for practical deployment. The study was based on a real-life multi-stage case of hostel space allocation problem with large data set. The simulation was performed using the data set. The paper's major concern was the allocation of various categories of students into hostels in order to maximize bed space utilization. The research result provided a firm foundation for decision making by relevant authority in the university on the allocation of hostel space that achieved a four-point three objectives of transparency, reliability, efficiency and effectiveness for hostel allocation.

Asharm (2009) asserted that a mathematical program that solves the problem of determining the optimal allocation of limited resources needed to meet a specific objective is the linear programming.The researcher uses optimization problem where both the objective function to be optimized and all constraints are linear as regards decision variables. It determines the way to accomplish the best result such as maximizing profit and minimizing cost in a particular mathematical model given some list of requirements, its considered as linear equation. POM-QM window 4 (software for Quantitative methods, production and operation management by Haward J. Weiss) was employed based on the simplex algorithm to achieve optimal solution. The analysis of the result indicated that six (50%) of the twelve classroom could be used to produce the highest classroom space of six

hundred and forty students. It was observed that the management could use two hundred and eighty (280) extra spaces to increase the student's intake and can also use the research work to reduce the number of classroom used by 50%.

Remi Safai (2009) developed a hybrid particle swam optimization to automate the design of the university course timetabling problem (UCTP) by proposing three algorithm which include (i) hybrid particular swam optimization constraints – based – reasoning (PSO-CBR), (ii) Hybrid particle swam optimization Local search (PSO-LS) and (iii) standard particle swam optimization (PSO) to solve the UCTP. The researcher stated that time-tabling problem is computationally an NP – Complete problem and has been model as constraint – satisfaction problem (CSP) making it very difficult to solve, using conventional optimization technique. In their study, a hybrid approach of PSO and other related constraints handling techniques were tested to solve UCTP. The research compared the hybrid approach with the other approaches and concluded that the hybrid particle swarm optimization algorithm provides feasible solution but to have a near optimal one with acceptable computational time, hybrid PSO-CBR is more promising and it shows that the convergence of optimizing UCTP is efficient due to the CBR ability to significantly reduce the search space.

According to Burke et al. (1999), the study proposed the review of the algorithms to solve the problem of space allocation or distribution (Hill climbing Simulated Annealing and genetic algorithm) using three different data sets from three different universities. It was stated that the space allocation problem within UK universities is highly constrained, and has multiple objectives, extensively among various institutions which require frequent modifications. The research revealed that application of local search, meta- heuristics and evolutionary algorithm to OSA problem can yield better result and a extensive comparison between all the three techniques are presented using real test data. The objectives of the

research is to examine or investigate the application of the three algorithms to the variants of the space allocation problem, comparing the advantages and disadvantages to obtain a better understanding of the problem and proposed future hybridization of such methods and additional methods. The three algorithms try to find the global optimum in the solution space.

The Hill Climbing algorithm has been referred to as a heuristic search algorithm which may become stuck in local optima, but simulated annealing and genetic algorithm avoid this local optimal by performing a broader exploration of the solution space. This paper applied three moves to modify allocation and explore the search space. The moves are ALLOCATE, RELOCATE and SWAP during the construction of the crucial solution and space exploration, the following parameters were investigated to determine the appropriate neighborhood exploration in each algorithm resources search, room search, space deviation and termination criteria.

2.6Appraisal of Literature Review

Office space allocation problem is considered a Non-deterministic polynomial combinatorial problem. Because of its combinatorial nature, several solution methods were proposed, such as exact method that gives the actual solution to same set of problem. Heuristic method such as hill climbing and metaheuristic solution such as Artificial Bee Colony, ant colony, genetic algorithm, variable neighborhood search algorithm, simulated annealing. The heuristic and metaheuristic algorithms provide near optimal solution to OSA problem within a reasonable time, the accuracy is trade off for time.

The computational complexity of a problem is normally determined by the best algorithm which can be discovered to solve the problem. From a extensive perspective, the efficiency of an algorithm is examined in term of the resources that are needed to execute the

algorithm and this include time and space (Johnson 1979). There are two categories of classifying the problem, the

polynomial (P) and Non-deterministic Polynomial (NP). The complexity of a problem and the complexity of an algorithm to solve a problem give a sign of how difficult it is to solve the problem from a computational point of view.

An exact algorithm is capable of solving a giving instance of a combinatorial optimisation problem to optimality. Nevertheless, the time complexity of exact solution algorithm is inefficient. The interest and actual significance of the concept of NP-complete problem lies in the popular belief that an efficient algorithm for solving such problem does not exist and that the algorithm which produces high quality (or near optimal) solution in a reasonable amount of time are then needed. Al-Harkeem(2010) said decision algorithm which uses trial by error techniques in deciding on the next improved solution to exploit within the local neighborhood structure of a solution space and also suffer from premature convergence.

This review shows that several researchers has applied several methods and algorithms to solving space allocation problem which ranges from mathematical and metaheuristic algorithm(Ozgur 2013), Integer Programming(landa & Ozgur 2011), harmony search algorithm(Awadallah et al 2013), Multilevel genetic algorithm(Adewumi 2010), Particle swarm optimization(Remi, 2009) among others.

2.7 Types of Complexity

2.7.1 Algorithms Complexity

The theory of algorithm complexity is concerned with the identification of problems which are computationally easy to solve and problems that are computationally hard to solve (Garey & Johnson, 1979; Rayward-Smith, 1986). This theory also deals with identifying

those algorithms that are efficient and those that are inefficient from a computational point of view. From a general perspective, the efficiency or effectiveness of an algorithm is examined in terms of the computing resources that are necessary to execute the algorithm and this includes execution time and space. The execution time is the number of steps that the algorithm takes to process the input and give an output. The space indicates the amount of memory that will be needed to run the algorithm. Nevertheless, in the theory of algorithms complexity, the efficiency of algorithms is always expressed in terms of its time complexity.

The time complexity is depicted by a function of the size of the input, which partains to the size of the problem instance. More particularly, the time complexity for an algorithm is described by its worst-case behaviour, which is the highest number of basic operations that the algorithm is expected to perform for an input of size n. The time complexity of an algorithm is expressed using the notation O (g(n)) which is defined as follows: A function f(n) is said to be O (g(n)) if there is a constant k such that $|f(n)| \leq k \cdot |g(n)|$ for $n \geq 0$. In other words, O (g(n)) refers to functions that do not grow faster than g(n) and the O (g(n)) notation indicates that the algorithm's worst-case time complexity is bounded by g(n).

Algorithms that possess a time complexity explained by a polynomial function (e.g. O (4n), O (n3), etc.) are considered efficient because they can be run in reasonable amount of time for inputs of considerable size. On the contrary, if the time complexity of the algorithm is explained by an exponential function (e.g. O (3n), O (nlog n), etc.), then the algorithm is considered to be inefficient because it can be run in a reasonable amount of time only for inputs of small length, but for larger inputs running the algorithm becomes impractical. The dissimilarity between polynomial time algorithms and exponential time algorithms deals with the rate at which their computational time complexity increases, given an increase in the size of the input (n). Remember that the time complexity of an

algorithm means to the worst-case performance. There exist some polynomial time algorithms which are not quite useful in practice because n is typically large in practical instances. Also, there exist some exponential time algorithms considered as useful due to the fact that they can run quickly in practice as a result of small values of n encountered in practical instances.

2.7.2 Problem Complexity – The P and NP Classes

The computational complexity of a problem is determined by the best algorithm which can be found to solve the problem (Garey and Johnson, 1979). At a high level of abstraction, if a polynomial time algorithm can be found for a given or particular problem, then the problem is considered tractable or not so hard. However, if no such algorithm is available for the problem, i.e. only exponential time algorithms can be constructed, the problem is considered intractable or very hard even if the problem is solvable. The major consideration for the development of the theory of computational complexity is decision problem. Most optimisation problems can be considered as a decision problem. A decision problem is a problem for which the answer is 'yes' or 'no' based on whether the input satisfies the particular conditions in the problem. Some examples of decision problems are presented below: EVEN. Given a natural number n, is n an even number? The answer is 'yes' if n is even or 'no' if n is odd. PRIME. Given a natural number n, is n a prime number? The answer is 'yes' if n is prime or 'no' if n is composite.

Satisfiability. Given a Boolean expression f (x1, x2,…xn), can the variables x1, x2,…xn be fixed to values that make the value of f true? The answer is 'yes' if there is a setting of the variables that makes f true and 'no' otherwise.

Hamiltonian cycle. Given a graph G(V,E) with N nodes, is there a cycle of edges in G that includes each of the N nodes? The answer is 'yes' if such cycle is in existence and 'no' otherwise. The space allocation problem described in the Literature review can also be stated as a decision problem:

Space allocation. Given n entities and m available rooms, is it possible to construct an allocation of the n entities to the m rooms in such a way that all existing constraints (hard and soft) are satisfied and the space misuse is minimized. The answer is 'yes' if such an allocation exists and 'no' otherwise.

Problems are often classified into two classes. According to Garey and Johnson (1979), and Rayward Smith (1986), the two classes are: P and NP classes. The class P includes all those

problems for which an efficient (polynomial time) deterministic algorithm has been found. On the contrary, the class NP includes all those problems for which a non-deterministic polynomial time algorithm is known to solve the problem (NP stands for non-deterministic polynomial). In the decription of a non-deterministic algorithm there are two stages involved.Guessing a structure for the problemtakes place at the first stage. What is done at the second stage is to verify whetherthe given structure is or is not a solution to the problem. Then, the algorithm is said to be a non-deterministic polynomial time algorithm if for each instance of the problem there is a guess that can be verified by the deterministic phase for answer 'yes' in a polynomial time. Then, if P are problems solved in polynomial time by deterministic algorithms and NP are problems solved in polynomial time by non-deterministic algorithms, the question is whether $P = NP$ or $P \neq NP$. In fact, this is the most important open question in computational complexity theory. It is clear that $P \subseteq NP$, which means that non-deterministic algorithms are more powerful than deterministic algorithms. If there is a deterministic algorithm for a problem, a non-deterministic one can be constructed by simply not using the guessing stage. No efficient algorithm has been found for numerous problems which are in the NP class. This strengthens the belief that $P \neq NP$ but this conjecture has not been proved. There existseveral problems considered to be in NP for which no efficient algorithm has been discovered and these problems are considered NP-hard in the strong sense. The multiple knapsack problem and the

generalized assignment problems constitute examples of these problems and it is generally believed that no efficient algorithm exists for these (and all other NP-hard) combinatorial problems, i.e. they are intractable. If it is true that P ≠ NP, then the problems in the set NP − P are intractable. As a result of this, when tackling a particular problem, it is important to confirm whether the problem belongs to the class of tractable or intractable problems. One important way to confirm this is to determine if the problem of interest is or not related to another problem that is already known to be tractable or intractable. Reducing one problem to another is the method used to determine whether there is a relationship between the two problems or not. Reduction is to provide a

transformation that permits to map one instance of the first problem into one instance of the second problem. This transformation permits to convert one algorithm that solves one problem into an algorithm that solves the other problem. There is a significant class of problems in NP, this is the class NP-complete. The first work towards the theory of NP-completeness was reported by Cook in 1971 (Cook, 1971). Cook (1971) proved that any problem in NP can be reduced to the satisfiability problem. This means that if there is an efficient algorithm to solve the satisfiability problem, it means that any problem in NP can also be solved by an efficient algorithm. These problems are said to be NP-complete and are considered the hardest NP in a way. This is because if there is no single NP-complete problem that has an efficient algorithm to solve it, then none of them has an efficient algorithm and they are all intractable (Landa, 2003). Many problems have been proven to be NP-complete (or reduced to the satisfiability problem) but it has not been proved that these problems are intractable. Nevertheless, it is widely assumed that finding an efficient algorithm for any problem in NP-complete is unlikely (Ulker, 2013). Then, if a problem is NP-complete and P ≠ NP then the problem belongs to the set NP − P. In other words, the problem (and all in NP-complete) might belong to P only if P = NP. Then, if it is assumed that NP-complete problems are intractable, i.e. P ≠ NP, then when a problem is known to

be NP-complete, the focus should not be on finding efficient algorithms. Instead one should aim to design algorithms that produce high-quality solutions with no guaranteed optimality, for example design useful algorithms to solve the problem in practice.

2.8Approaches to Solve Optimisation Problems

The complexity of a problem and the complexity of an algorithm in solving problem gives an indication of how hard it is to solve the problem from a computational view point. An exact algorithm is capable of solving a given instance of a combinatorial optimisation problem to optimality. However, the time complexity of some exact algorithms is bounded by an exponential function, which makes these algorithms inefficient. The interest and practical

significance of the concept of NP-complete problems lies in the general belief that an efficient algorithm for solving such problems does not exist and that algorithms that produce high quality (or near-optimal) solutions in a reasonable amount of time are then needed. Such a heuristic is defined byReeves(1995) as a "technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to assure either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is". Constructive algorithms are examples of heuristics (also known as greedy methods). These are very simple heuristics that construct the solution in a series of steps based on the strategy of making the best decision (based on a certain criterion) at every step. Another example of heuristic methodology is local search (which can also be reffered to as neighbourhood search) where neighbouring solutions are explored in an attempt to improve the solution (although worse solutions can be accepted as an interim step). In the last couple of decades, more advanced heuristic approaches reffered to as metaheuristics have been widely developed and applied to a variety of optimisation problems (for instance, Glover and Kochenberger, 2003; Voss et al., 1999;

Aarts and Lenstra, 1997; Osman and Kelly, 1996; Osman and Laporte, 1996; Rayward-Smith et al., 1996; Reeves, 1995). A metaheuristic is described in Voss et al., (1999 P ix) as “an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It has the capacity to manipulate a complete (or incomplete) single solution or a collection of solutions at every iteration. The subordinate heuristics can be high (or low) level procedures, or a simple local search, or just a construction method”. When solving combinatorial optimisation problems, there are exact algorithms that, given enough time, can guarantee finding an optimal solution. There are also very specialised heuristics that exploit knowledge of the problem domain and produce solutions of good quality. There also existsome metaheuristics that are not actually designed for a particular problem but are considered general approaches that can be tuned for any problem. While certain metaheuristics may need tuning, others may act as a black box since they can be implemented

with none or very little information about the problem being solved. An example of such black-box approach is random search, which can be used to compare the performance of other algorithms.

2.9Heuristic and Meta-heuristic Algorithm

Heuristic algorithms provide near-optimal solutions to non-deterministic optimization problems in ***P***. They are decision algorithms, which employ trial and error techniques in deciding on the next improved solution to exploit within the local neighbourhood structures of a solution space. Heuristic algorithms are iterative algorithms and normally stop when the initial number of iterations complete. Heuristic algorithms may suffer from premature convergence. Premature convergence is the possibility of an algorithm being stuck in local optima. This can give a largely inferior feasible solution compared to the global optimal solution. An example of a heuristic algorithm is HC (Ariyo, 2013).

Metaheuristic algorithms refer to the improved heuristic algorithms. They build on heuristic algorithm methods and are more effective and efficient in searching the solution space. These algorithms employ ‘advanced’ methods in order to prevent premature convergence. This gives room for the possibility of finding more improved feasible solutions by performing a wider or more extensive search of the local neighbourhood structures. Some advanced methods include using memory ability, learning from other decision variables as well as randomly ‘jumping’ to other local neighbourhood structures, amongst others. Metaheuristics algorithms are modelled using real life sequences of events. Examples include the way ants and bees search for food and the annealing process associated with cooling heated metal. Metaheuristic algorithms may not be problem specific. Examples of metaheuristic algorithms include SA, TS, GA, ABC, Ant colony etc. Metaheuristic algorithms need to find a good balance between exploring and exploiting the local neighbourhood structures of the solution space (Syam and Al-Harkam,

2010).

Exploration involves looking for more ‘promising’ local neighbourhood structures. These ‘promising’ areas may contain improved solutions. Exploitation involves searching within a local neighborhood search area in order to find its local optimum. Looking for a good balance between exploration and exploitation means that an algorithm should quickly determine promising local neighbourhood structures but must not spend too much time searching for local optima (Syam and Al-Harkam, 2010).

In Computer Science and Mathematical optimization, a metaheuristic optimization is a higher level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information. Metaheuristic is a method used to solve very general classes of problems. It combines objective functions of heuristic

in an abstract and efficient manner usually without utilizing deeper insight into their procedure, i.e. by treating them as black – box procedures. The word "Meta" means beyond or higher level. Metaheuristic algorithms can be classified as either local search algorithms or population based algorithms (Bum and Roli, 2001). Examples of local search algorithms are Simulated Annealing, Tabu Search, Greedy Randomized Adaptive Search, Hill Climbing and Variable Neighbourhood Search, among others. On the other hand, examples of Population-Based Algorithms are Genetic Algorithm, Scatter Search, Ant Colony, Artificial Bee Colony, Particle Swarm Optimization and Memetic Algorithms, among others.

2.10. Descriptions of Local Based Algorithm

i. Simulated Annealing Algorithm

Simulated Annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a particularfunction in a broad search space. It is regularly employed when the search space is discrete. For certain

problems, simulated annealing may be more efficient than exhausted enumeration provided that the goal is only to find an acceptable good solution in a fixed amount of time rather than the best possible solution.

The name, an inspiration, comes from annealing in metalunary, a teaching involving heating and controlled cooling of a material to increase the size of the crystal and decrease their defect. Both attributes belong to the material that depend on its thermodynamic free energy. This notion of slow cooling is implemented in the simulated Annealing algorithm as a slow decrease in probability of accepting worse solution as it explores the solution space. Accepting worse solution is a fundamental property of metaheuristics because it

allows for a more-extensive search for the optimal solution. Simulated annelling searching for a maximum objective is to get the highest point.

ii. Tabu Search Algorithm

Tabu-search is a metaheuristic that attempts to guide the search in a systematic and intelligent way by using flexible and adaptive memory structure and some intensification and exploration strategies (Glovor et al. 1986 and 1993; Glovor and Laguna 1997; Hasen (1986). The major components of tabu search are: short term memory, long term memory and intensification and diversification strategies.

Short-Term Memory:- is used to forbid revisiting solution and then avoid cycling and being trapped in poor local optima.

Long-term Memory:- Is used as a type of learning procedure to generate intensification and diversification strategies. Long-term memory is employed to collect information during the overall search process that permits the identification of common properties in good visited solution and also to attempt to visit solution with diverse properties from already visited. The implementation of both short term and long term memory is based on four principles.

1. **Recency**:- is an indication of how recent it was that certain solution were visited.
2. **Frequency**:- is an indication of how often solution was visited.
3. **Quality**:- refers to keeping information about visited solution with good fitness values of identifying good solution component and simulate more intensive search in promising areas of the solution space.
4. **Influence**:- Is used to identify those change induced in the solution structure that has been proved to be more beneficial.

iii. Variable Neighbourhood Search

Maldenovic and Itensen (1997) Variable Neighborhood Search (VNS) is based on the idea of employing multiple neighbourhood structures in the process of local search. VNS algorithm tries to exploit the motion that a local optimum with respect to a single neighbourhood structure may not necessarily be optimal from the point at another neighbourhood structure. However, a global optimum of a problem is locally optimal in all neighbourhoods irrespective of the neighborhood structures used. In a VNS, a fair amount of different neighbourhood structures that represent different areas in the search space is desired. The intensification of the search can be satisfied via a local search within a single neighborhood while diversification may be achieved by systematically switching to different neighborhood structures.

The basic of VNS algorithm is usually divided into three (3) stages.

1. Shaking stage
2. Local search stage
3. Move stage

The Neighborhood structure list N (which consists of neighborhood N_k) is sorted in order from index K = 1 to K = Kmax.

- Shaking Stage:- A random solution is created by using the neighborhood structure N_k starting with K = 1. The solution in shaking stage is improved by local search operator using this neighborhood structure N_k. If the new local optimum solution is better than the incumbent solution, then this new solution is accepted.
- Move Stage**:-** If the solution cannot be improved with this neighborhood N_k the algorithm switches to the next neighbourhood $(N_k + 1)$ in the neighbourhood list. If none of the neighbourhood structure is able to improve the solution x, then neighbourhood index is reset back to k = 1, and the algorithm continues with a new random solution generated by the shaking stage. The VNS algorithm continues until a pre-determined termination criteria is met.

iv. Greedy Randomised Adaptive Search Procedure Algorithm

Greedy Randomized Adaptive Search Procedure (GRASP) (Resende Ribecro, 2003) is a multi start metaheuristic in which every iteration of the search comprises of two stages; which are construction stage and local search stage. A feasible solution is generated in the construction phase or stage and this solution is improved to a local optimal by means of a local search operator. The initial construction phase of every iteration may be much more complex than just randomly picking a new point in the search space. According to Resende, (1922), this was described as an iterative construction process where a particular element (gene) is "added" at a time where the elements to be added are chosen with respect to a greedy function. Here, not necessary the best possible all element is sent but one of the top candidates is selected randomly. After the initial solution is generated this way, a local search is applied to refine it.

v. Hill Climbing Algorithm

Hill climbing (HC) (1780) is an old and simple search and optimization algorithm for single objective function F. In principle, hill climbing algorithms perform a loop in which the currently known best solution individual p* is used to produce one offspring P-new. If this new individual is better than its parent, it replaces it, then the cycle starts all over again in this sense. It is similar to an evolutionary algorithm with a population size of 1. Although the search space g and the problem space x are most often the same in hill climbing, we differentiate them in algorithm for the purpose of generality. Furthermore, Hill climbing usually employs a parameter less search operation to create the first solution candidate and from there on, unary operation to generate the offspring, without losing its generality. This makes use of the reproduction operation from evolutionary algorithm.The major problem associated with Hill climbing is premature convergence, for example it gets easily stuck on a local optimum. It make use of the best known solution candidate x* to

look for new point in the problem space x. Hill climbing uses a unary reproduction operation similar to mutation in evolutionary algorithm. It should be noted that Hill climbing can be implemented using deterministic approach if the neighbour sets in search space G.

2.11Population Based Algorithms

i. Genetic Algorithm (GA)

Genetic Algorithm (GA) were in essence suggested by Holland in his book, Adaptation in Natural and Artificial System (Holland, 1975). Nevertheless, the ideas of using evolution and recombination for optimization were suggested even earlier by Bremerman, (1962). A genetic algorithm is a population based techniques that is based on principles of natural evolution (Goldberg, 1989; Man et al.1999; Michalewlez, 1999). The major concept in genetic algorithm is to generate a population of individuals and then, during a number of iterations (generations) to evolve this population by means of self-adaptation and recombination.

1. Initialize population
2. Find fitness of population
3. While (termination criteria is reached) doparent selection
4. Crossover with probability pc
5. Mutation with probability pm
6. Decode and fitness calculation
7. Survivor selection
8. Find best
9. Return best

iii. Particle Swarm Optimization Algorithm

\Particle swarm optimization algorithm (PSO) is a biologically inspired computational search and optimization technique developed in 1995 by Eberhart and Kennedy based on the social behaviour of birds flocking or first schooling. Normally, a flock of animals that have no leaders will find first fort by random, follow one of the members of the group that has the closest position to a food source (potential solution). The flock achieves its best condition simultaneously through communication among members who already have a more beneficial situation. Animal that has a better condition will inform its flock and the others will simultaneously move to the place. This would happen repeatedly until the best condition or a food source is discovered. The procedure of PSO algorithm in discovering optimal value follows the work of this animal society. Particle swarm comprises of a swarm of particles, where particle represents a feasible solution.

iii. Scatter Search Algorithm

Scatter Search (SS) was first introduced in Glover (1977) as a heuristic for integer programming. In the original proposal, solutions are purposely (i.e. non-randomly) generated to take account of characteristics in different parts of the solution space. Scatter search orients is explorations systematically relative to a set of reference points that normally comprises of good solutions obtained by previous problem solving attemps, where criteria for “good” are not restricted to objective function values, and may apply to sub collection of solutions rather than to a particular solution as in the case of solution that differs from one another according to particular specification. The scatter search template (Glover 1998) has served as the main reference for most of the scatter search implementations to date. The dispersion patterns created by these designs have been found useful in several application area. (Lagunna, 2003).

iv. Ant Colony Algorithm

Ant behaviour was the inspiration for the metaheuristic optimization technique, in computer science and operations research. The Ant colony optimization algorithm (A.C.O) is a probabilistic technique for solving computational problems which can be used in finding good paths through graph.It is a swarm intelligence optimization method. In the natural world, ants generally wander randomly, and upon finding food go back to their colony while laying down pheromones trails. If other ants find such path they are likely not to keep travelling at random but instead follow the trail, returning and reinforcing it if they finally find food. Overtime, however the pheromones trails start to evaporate, this reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A shortest path gets matched over more frequently and thus the pheromones density becomes higher on shorter path than longer ones.

Pheromones evaporation has the advantage of avoiding the convergence to a local optimal solution, if there were no evaporation at all, the path chosen by the first ant would tend to be excessively attractive to the following ones. In such a case, the exploration of the solution space would be constrained. Therefore, when one ant finds a good path from the colony to a food source, other ants are more likely to follow that path and (positive feedback) eventually leads to all the ants following a single path. The concept of the ant colony algorithm is to mimic this behaviour with “simulated ant” walking around the graph representing the problem solved.

v. Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) algorithm is a swarm based meta-heuristic algorithm that was introduced by Karaboga in 2005 for optimization of numerical problem. It was inspired by the intelligent foraging behaviour of honeybees. The algorithm is specially based on model for the

foraging behaviour of honey bee colonies.The models comprises of three important components, namely Employed and unemployed foraging bees, and food source. The first two components employed and unemployed foraging bee search for rich food source which is the third component, close to their hive. The model also defines two leading models of behaviours which are important for self-organizing and collective intelligence-recruitment of foragers to rich food source leading to a positive feedback, and abandonment of poor source by foragers causing negative feedback.

Artificial forager bees (agents) search for rich artificial food source (good solution for a given problem). To apply ABC, the considered optimization problem is first converted, the problem of finding the best parameter vector which minimizes an objective function, then the artificial bee randomly discovers a population of initial solution vectors and then iteratively improves them by employing the strategies moving towards better solution by means of a neighbourhood search mechanism while abandoning poor solution.

The ABC metaheuristic contains three groups of bees.

1. Employed Bee-: employed bee associated with specific good source
2. On looker Bee-: watching the dance of employed bees within the hive to choose a food source.
3. Scout-: searching for food source randomly both onlooker and scouts are called unemployed bees. Initially, all food source position are discovered by the scout bees, later on, the nectar of food source are exploited by employed bee and on looker bee and this continual exploration will finally cause them to become exhausted. The employed bee which was exploiting the exhausted food source

became a scout bee in search of further food source once again. In other words, the employed, whose food source had been exhausted, became a scout bee.

In ABC the position of food source represents a possible solution to the problem andthe nectar amount of food source corporate to the quality (fitness) of the associated solution, the number of employed bee is equal to the number of food source (solution) because every employed bee is associated with just only one food source.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

The research methodology involves five phases, through which the research objectives were achieved. These include:

i. Collection of Dataset for the Faculty of Communication and Information Sciences, this include the measurement of all the room to be allocated, total number and cadre of academic staff, amenities available in the offices including toilet facility.

ii. Mathematical Modelling: This stage mathematically modelled the office space allocation problem in terms of the objective function and the constraints. This mathematical model serves as one of the contributions to knowledge, of this research work.

iii. Adaptation of ABC, Genetic and Tabu search meta-heuristic algorithm for the office space allocation problem (OSAP). This stage adopted the Artificial Bee Colony, Genetic and Tabu search to the problem of OSA.

iv. Hybridized ABC and Tabu Search algorithms. This stage hybridized the features of these two algorithms to enhance the performance. Also, a comparative study of the hybrid, ABC, Genetic and Tabu Search(in iii above) was done in this phase.

v. Comparative evaluation of the hybridized and the existing algorithms (TABU, Genetic and ABC) using Halstead's complexity measure(Program vocabulary, Program length, Program volume, Program intelligence and program difficulty).

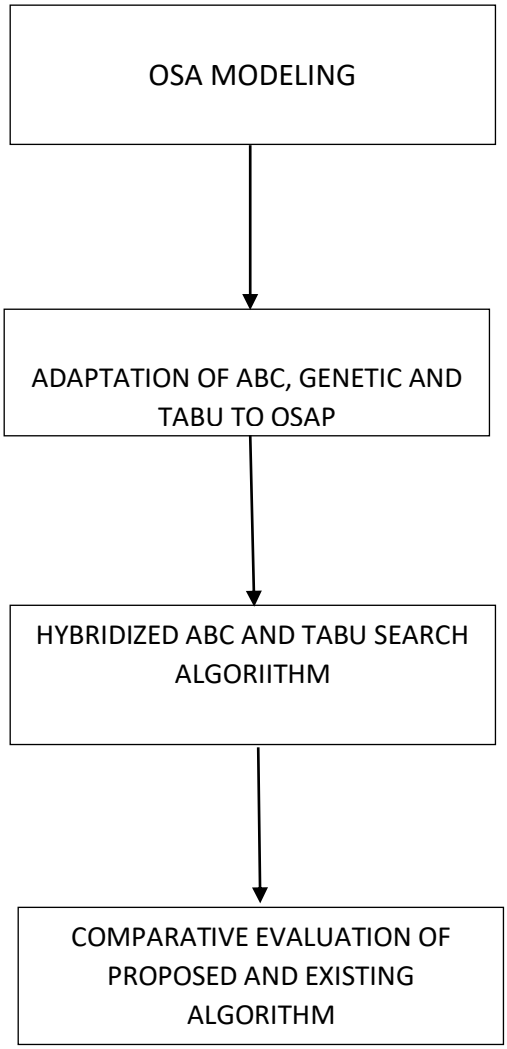

Figure 3.1: Block Diagram for Research Methodology

3.2The Mathematical Model of the Objective Function

Objective function = Usage penalty + Soft Constraint Penalty

Usage Penalty = Under-use + 2 x overuse

Objective Function = Under-use + 2overuse + Soft Constraint Penalty

$Max_z = \Sigma MaxCr - \Sigma merSe + 2\Sigma MerSe - Cr + \Sigma w^{ij} \Sigma m^{ij}$

$= \Sigma MaxCr + \Sigma MerSe - Cr + \Sigma w^{ij} \Sigma m^{ij}$

Max Cr= Maximum capacity of a room

Under Use = Addition of maximum capacity of a room – Addition of member of entity allocated to a room multiply by size of the entity.

Soft Constraint = 2 multiply (*) by addition of member of entity to a room multiply (*) by size of the entity minus (-) capacity of the room plus(+) addition of penalty weight for the violation of soft constraint in particular rows and colon by addition of member of entity in a particular row and particular colon.

Weight of Summation of Violated Constraint

$$\sum_{l=1}^{n}\sum_{j=1}^{n} Wi_j = Wi_1 + Wi_{2+} Wi_3 \dots\dots\dots Wi_n$$

All Allocated:

If each entity e ϵ E that is: must be allocated to exactly one room r ∈ R

Addition of member entity allocated to a room is equal to 1 and each room will represent a fraction for an entity.

$$\sum mer = 1 \; for \; every \; entity \; a \; member \forall \; r \in \; R$$

Near by:

If e_1 and e_2 have to be allocated to nearby room then

Member of entity 1 is less than addition of all other member to be allocated to nearby room is less than 1

Me_1r < Σme_2< 1 $\forall \; r \in$ R

Capacity:

Entities in a group have to be allocated on the basis of cadre.

$Mer_1 + \Sigma m_{Br} = 1 \quad \forall \; r \in$ R

$Mer_1 = 1 - \Sigma m_{Br} \forall \; r \in$ R

Member of entity in a room = one minus addition of all other staff (Br) to be allocated to a room.

Member allocated to a room is less than addition of all other members to be allocated to the other rooms is also less than one

Not Sharing

If entity e should not share a room with any other entity.

The Hard constraint states that Professor should not share a room.

This implied that:

$M_{er} = 1 \quad \forall r \in R$

$\Sigma m_{e1r} = 1$

That is all professor must occur have and only one room.

Away from:

Entities e_1 and e_2 have to be allocated in a room separately to a room away from each other. This implied that :

$Mer = 1 \rightarrow \Sigma m_{Br} = 0$

Same room

Entity e_1 and e_2 have to be allocated to the same room.

$Me_1r = 1 \rightarrow Me2r = 1 \forall r \in R$

Where

Soft Constraint = is the weighted sum of the Penalty due to the violation of the Constraint

Se = is the cadre of the entity e

Cr = is the capacity of room r

W^{ij} = Penalty weight for the violation of Soft Constant

R = Set of rooms

E = Set of entities

e_1 = element of E

Mer = member of entity allocated to room

M^{ij} = member of entity in a particular room.

$Max_z = \Sigma MaxCr + \Sigma MerSe - Cr + \Sigma w^{ij} \Sigma m^{ij}$

Subject to:

$\sum mer = 1$	All Allocated
$\Sigma me_2 < 1$	Near By
$Mer_1 + \Sigma m_{Br} = 1$	Capacity
Mer = 1	Not Sharing
$\Sigma m_{Br} = 0$	Away From
$Me_{2r} = 1$	Same Room

ID	TYPE OF CONSTRAINT	DESCRIPTION OF THE CONSRAINT	PENALTY WEIGHT
1	Allocation	All entity should be allocated	20
2	Same room	Entity e1 and e2 can be in a room	10
3	Not sharing	Entity e should not share a room	50
4	Nearby	Entity e1 and e2 should be in a nearby room	10
5	Away from	Entity e1 and e2 should be away from one another	10
6	Capacity	A room capacity must be maintain	10

Table 3.1: Penalty Weight for each constraint (Ozgur and Landa 2011)

3.3 Description of the Algorithm

3.3.1 Tabu Search algorithm

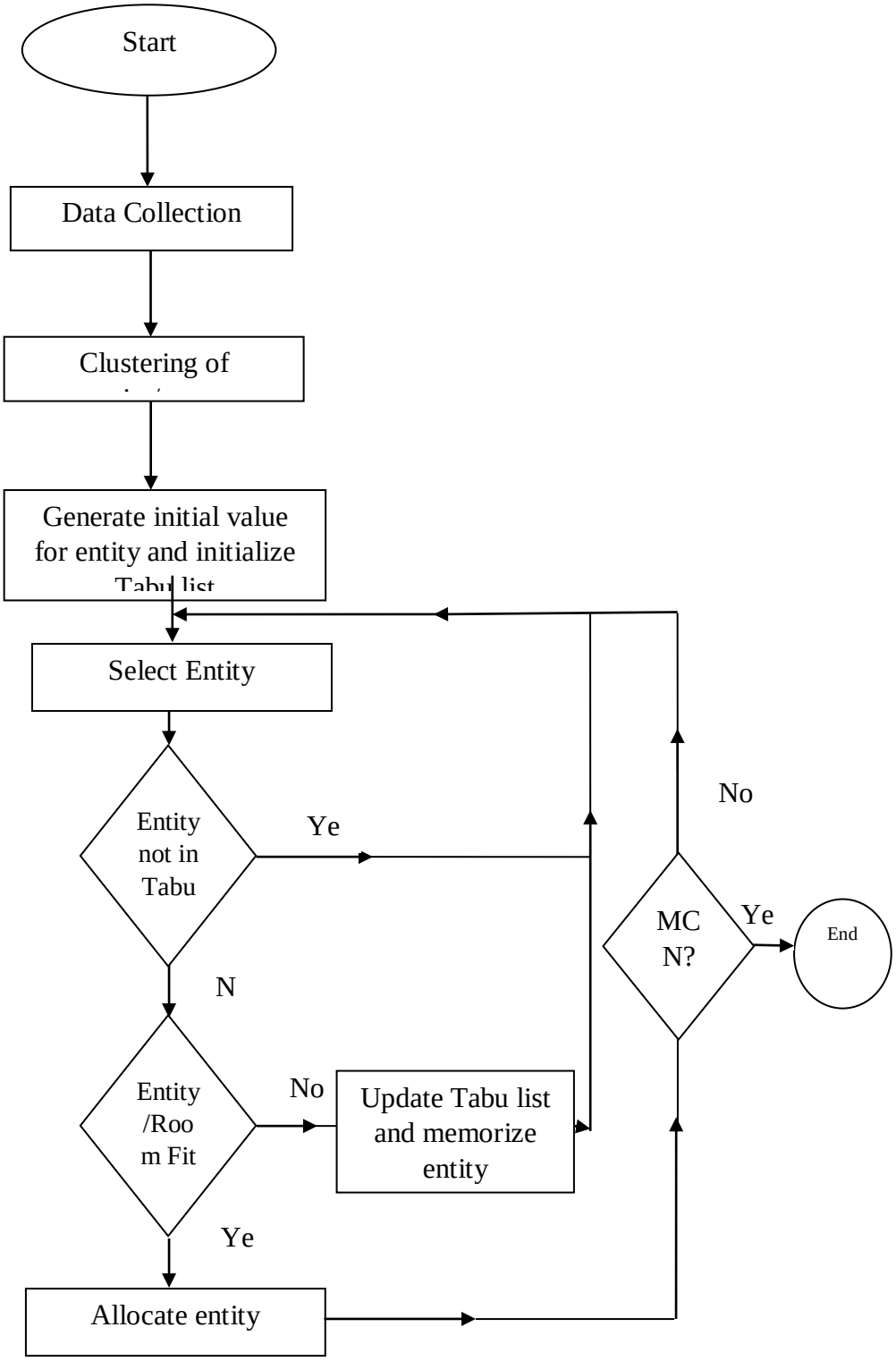

Figure: 3.2 Tabu search algorithm flowchart

Step 1: Prepare inputs

Step 2: Initialise tabuList

Step 3: Initialise program parameters

Step 4: Generate Initial Solution and initialise

Step 5: Calculate penalties for each solution

Step 6: Save best solution

Step 7: Do until maxCycle

Step 7.1 for each solution

Step 7.2 randomly select a neighbour solution and a staff

Step 7.3 DoWhile (combination of staff and neighbour allocation is not tabuList)

Step 7.3.1 randomly select a neighbour solution and a staff

Step 7.4 allocate staff to selected office

Step 7.5 calculate the penalty for the modified solution

Step 7.6 if new penalty < old penalty

Step 7.6.1 replace allocations in solution with the updated one

else

Step 7.6.2 add new allocation to tabuList

Step 7.6.3 reverse the change made

Step 7.7 Keep best "solution" in memory

Step 8 Output results

Figure 3.3 Tabu Search algorithm as used for the OSAP

3.3.2 ABC Algorithm

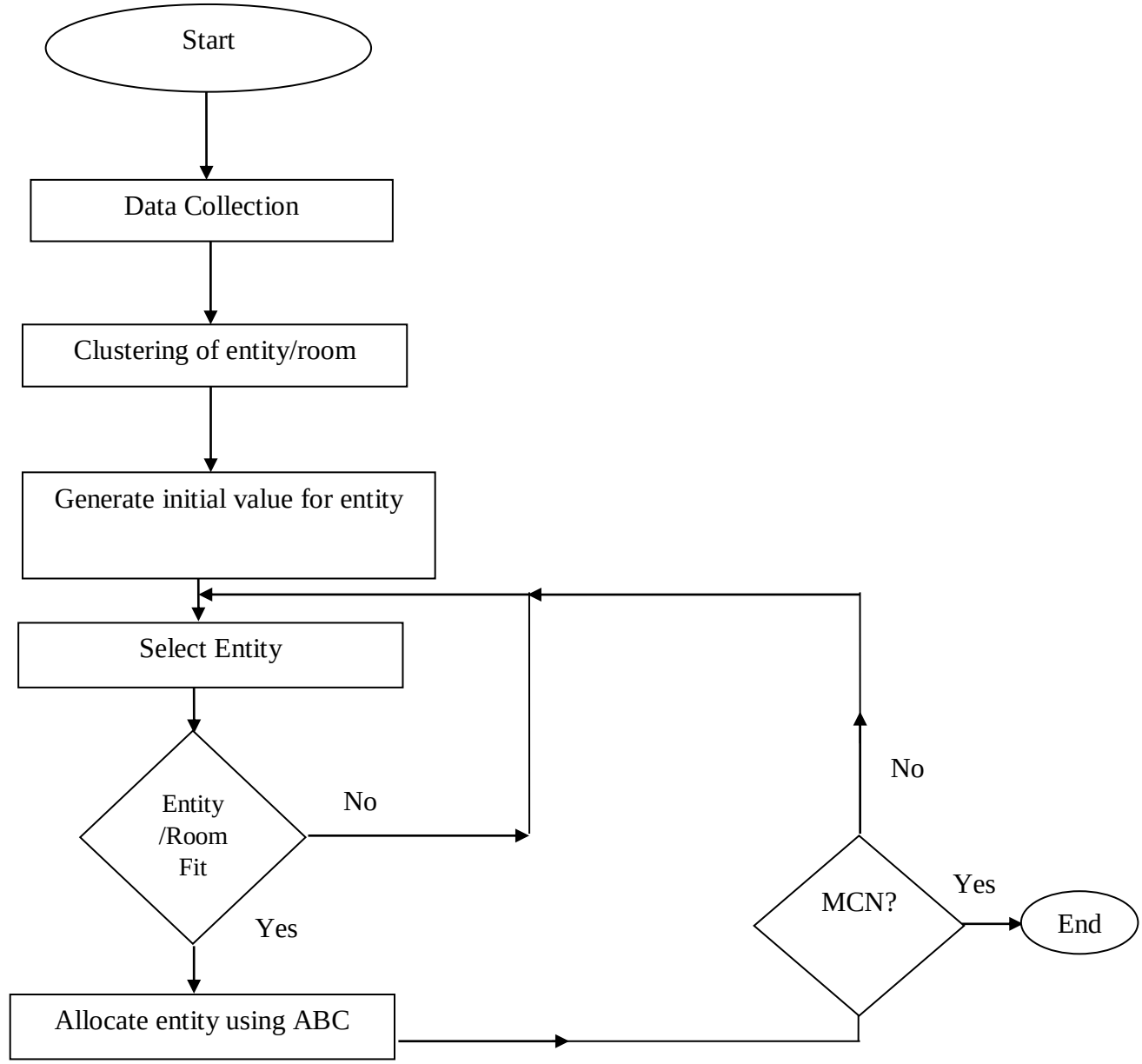

Figure 3.4: ABC algorithm flowchart

Step 1: Prepare inputs

Step 2: Initialise program parameters

Step 3: Generate Initial food source and initialise

Step 4: Calculate penalties for each food source

Step 5: Save best food source

Step 6: Do until maxCycle

Step 6.1 SendEmployedBees

Step 6.2 CalculateProbabilities

Step 6.3 SendOnlookerBees

Step 6.4 Save best food source

Step 6.5 SendScoutBees

Step 7 Output results

Figure 3.5 ABC ALGORITHM AS USED FOR THE OSAP

3.3.3 Genetic Algorithm

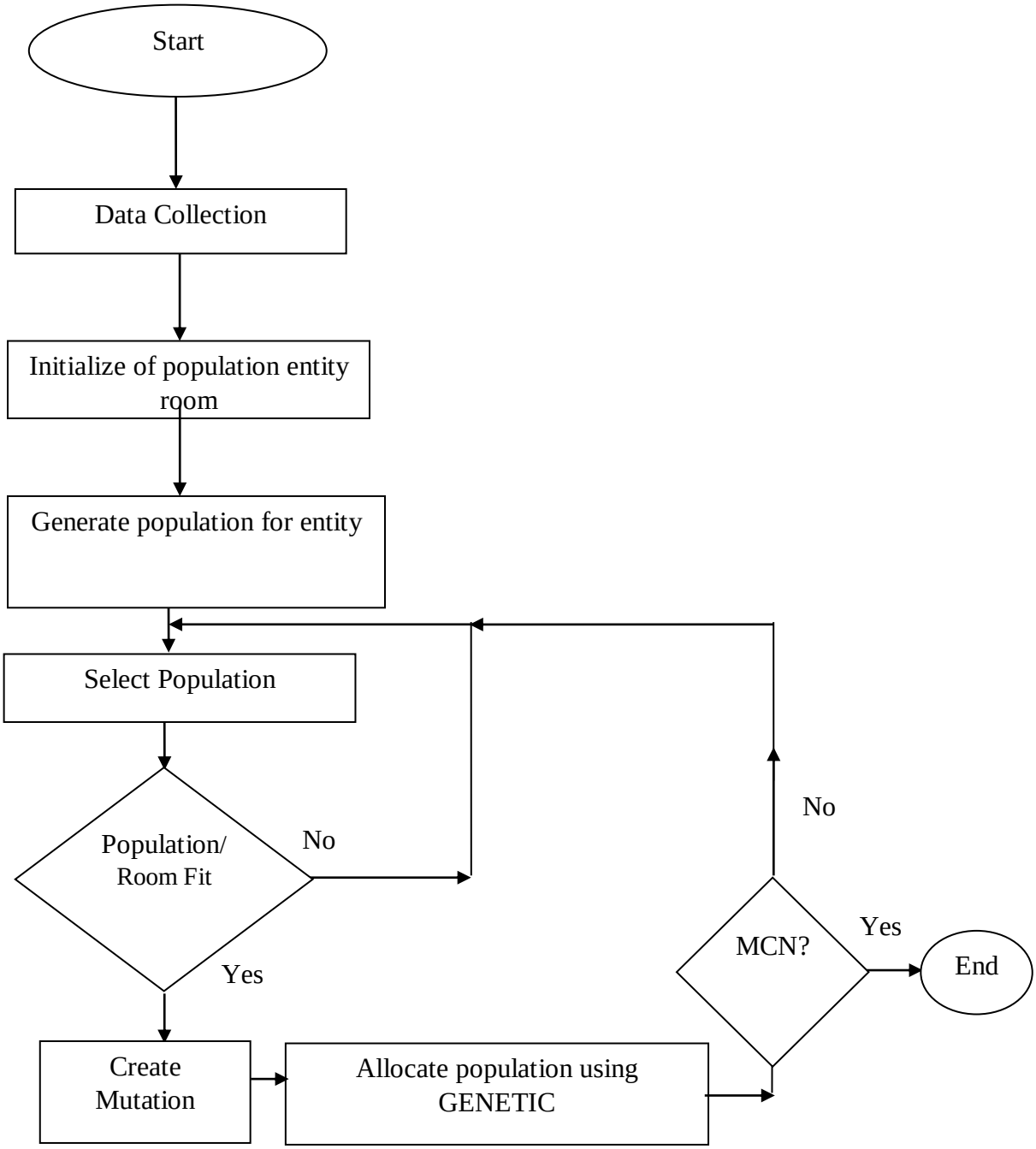

Figure 3.6: Genetic algorithm flow chart

Step 1: Prepare inputs

Step 2: Initialise program parameters

Step 3: Generate Populations

Step 4: Create Mutations

Step 5: Calculate penalties for each mutation created

Step 6: Save fitted mutation

Step 7: Do until maxCycle

Step 7.1 SelectionOperation

Step 7.2 CalcSelections;

Step 7.3 CrossOverOperator

Step 7.4 MutationOperator

Step 7.5 Save fitted mutation

Step 7 Output results

Figure 3.7: Genetic Algorithm as used for the OSAP

3.4 Hybridized algorithm

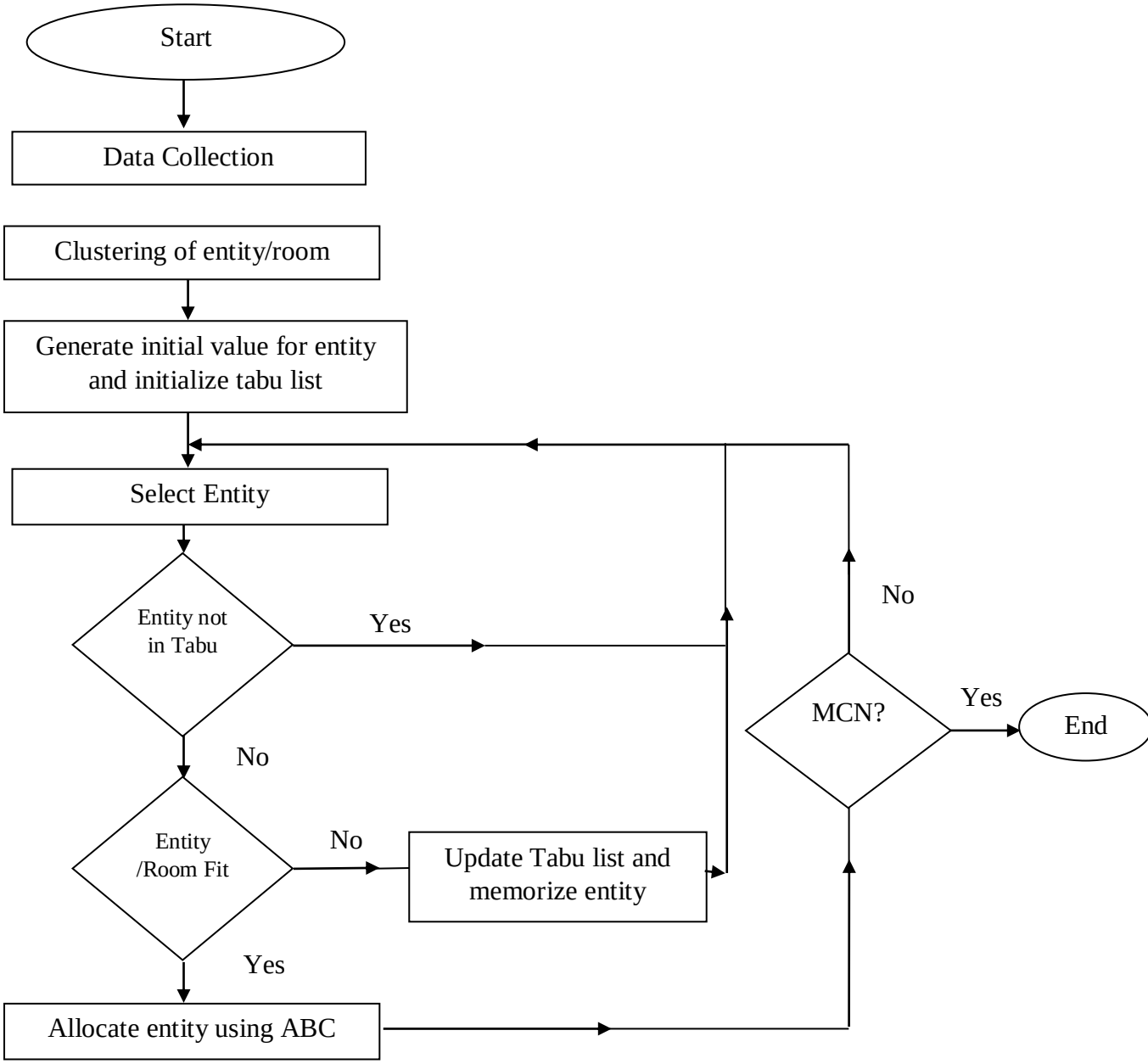

Figure 3.8: Hybridizes (tabc) flowchart

Step 1: Prepare inputs

Step 2: Initialise tabuList

Step 3: Initialise program parameters

Step 4: Generate Initial food source and initialise

Step 5: Calculate penalties for each food source

Step 6: Save best food source

Step 7: Do until maxCycle

Step 7.1 SendEmployedBees(tabuList)

Step 7.2 CalculateProbabilities

Step 7.3 SendOnlookerBees(tabuList)

Step 7.4 Save best food source

Step 7.5 SendScoutBees(tabuList)

Step 8 Output results

Figure 3.9: Hybridized (tabu-abc) Algorithm as used for the OSAP

```
1. //Prepare inputs
   input: entity, penalty, room, staff, n, MCN, maxTrialLimit
2. //Initialize parameters
   Tabulist = 0, X = (x1, x2, x3...xn), trialLimit = 0
3. //Iterate through n
   Do
      x = randomly generated solution entity/room from the solution space
      //Evaluate x using ABC and Iterate through MCN
      Do
         //Send Employee bees
         y = check for neighbour solution in x
            if y is not in solution x then
               x = change entity/room in the solution space
               //Calculate penalty
               penalty p = penalty
            else
               //increase trialLimit
               trialLimit = trialLimit + 1
         //Send Onlooker bees
         w = check for neighbour solution in x
            if w is not in solution x then
               x = change entity/room in the solution space
```

```
//Calculate penalty
            penalty p = penalty
        else
            //increase trialLimit
            trialLimit = trialLimit + 1
    //Get fitness
      Compute Fitness for x = x* (x1*, x2*, x3*...xn*)
       // Send Scout Bee
       if trialLimit >= maxTrialLimit
          //test for fitest solution in tabulist
          if x* = tabulist then
             //Perform tabulist update
                tabulist = x*
          else
             //Perfrom allocation for x
             x = x*
   Until MCN
 Until n
4. Stop
```

Figure 3.10: New Hybrid (TABU-ABC) Metaheuristic Algorihtm

3.5Halstead Complexity Measures

Halstead algorithms have measurable characteristics analogous to physical laws.The model is based on four different parameters:

n1: the number of distinct operators (instruction types, keyword) in a program, n2: the number of distinct operands (variables and constant).

N1: the total number of occurrences of the operators,

N2: the total number of occurrences of the operands.

these numbers, several measures can be calculated:

Program vocabulary $n = n1+ n2$ (3.1)

Program length $N = N1+ N2$ (3.2)

From the four counts, a number of useful measures can be obtained. The number of bit required to specify the program is called volume (V) of the program and is obtained through equation 3.3

$V = N*\log2n$ (3.3)

The program difficulty level (L). Which is the difficulty of understanding a program, is calculated by equation 3.4.

$L=(2n_2)/(n_1N_2)$ (3.4)

The intelligence content of a program (I) can be calculated by equation 3.5.

$I = L* V$ (3.5)

In an attempt to include the psychological aspects of complexity in the measures. Halstead studied the cognitive processes related to the perception and retention of simple stimuli. In his model, the number of discriminations made in the preparation of a program called effort (E) is given by equation 3.6

$E = V/L$ (3.6)

The memory used is referred to as program memory size which is been measure in kilobyte(KB) is given by equation 3.7

M = Total memory-(free+buffer+cache) (3.7)

The total execution time of the program is given in equation 3.8

T= E/18 (3.8)

The total number of bugs in the program is measure in equation 3.9

No of bugs = V/3000.

3.6 Data set for Faculty of Communication and Information sciences as used by OSAP

See APPENDIX E

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter discuss the implementation and result of the three metaheuristic algorithms together with the hybridized algorithm, the result was evaluated Using Halstead Complexity measure. The algorithms were implemented using C# programming language with MySQL database. The program consist of six class/module(Staff, Office, allocation, solution, MySQL database, input set).

The program randomly generate the solution and start comparing the solution to get the best solution, the weigth of the penalty violated is calculated by checking each allocation against each constraint to know whether or not the constraints are violated.

4.2.Implementation

See Appendix D

4.2. Experimental Results

4.2.1 Tabu Algorithm

Table 4.1: Result of TABU Algorithm for the six runs.

PARAMETER	RUN1	RUN2	RUN3	RUN4	RUN5	RUN6
RUNTIME /SEC	1106.40	1111.48	1094.55	1280.61	1263.57	1532.64
MEMORY USED (KB)	8951	17260KB	25602	33893	42162	50612
PENALTY WEIGHT(Count)	4030	3950	3880	4170	4070	4120

Considering the run time, memory used and the penalty weight of each of the runs. The third run gives the best result, in term of run time. It returned the least run time of 1094.55 seconds, 25602kilobyte and penalty weight of 3880count which is the least penalty in all the six runs. The sixth run returned the worse run time of 1532.64 seconds, 50612kb and 4120 penalty weight.

4.2.2 ABC Algorithm

Table 4.2: Result of ABC algorithm for six runs.

PARAMETER	RUN1	RUN2	RUN3	RUN4	RUN5	RUN6
RUNTIME /SEC	3039.25	3115.20	3235.94	3034.79	3096.62	3167.15
MEMORY USED	664KB	666KB	666KB	666KB	666KB	666KB
PENALTY	1640	1640	1460	1380	1460	2490

WEIGHT						

Considering the run time, memory used and the penalty weight of each of the runs. The fouth run gives the best result, in term of run time. It returned the least run time of 3034.79 seconds, 666 kilobyte and penalty weight of 1380count which is also the least penalty in all the six runs. While the third run returned the worse time of 3235.94 seconds,666kb and 1460 penalty weight.

4.2.3 Genetic Algorithm

Table 4.3: Result of Genetic Algorithm for six runs.

PARAMETER	**RUN1**	**RUN2**	**RUN3**	**RUN4**	**RUN5**	**RUN6**
RUNTIME /SEC	4368.06	4293.79	4306.61	4193.06	4185.75	4190.71
MEMORY USED(KB)	648	650	650	650	650	650
PENALTY WEIGHT	1760	1960	1890	1670	2020	1730

Considering the run time, memory used and the penalty weight of each of the runs. The fifth run gives the best result, in term of run time. It returned the least run time of 4185.75 seconds, 650kilobyte and penalty weight of 2020count. While first returned the worse run time of 4368.06seconds, 648kb and 1760 penalty weight.

4.2.4 Hybridized Algorithm

Table 4.4 Result of Hybrid Algorithm for six runs.

PARAMETER	**RUN1**	**RUN2**	**RUN3**	**RUN4**	**RUN5**	**RUN6**
RUNTIME /SEC	666.21	692.03	661.32	561.82	563.35	555.04
MEMORY USED(KB)	688	689	692	692	695	697
PENALTY WEIGHT(COUNT)	3830	3890	3980	3800	3850	3960

Considering the run time, memory used and the penalty weight of each of the runs. The sixth run gives the best result, in term of run time. It returned the least run time of 555.04 seconds, 697kilobyte and penalty weight of 3960. While run two returned worse time of 692.03 seconds, 689kb and 3890 penalty weight.

Considering all the best run for each of the adapted algorithms. The Hybridized algorithm returned the best run time (least) of 555.04 as against 1094.55 seconds of TABU search, 3034.79 seconds of ABC and 4185.75 seconds of Genetic algorithm. The Genetic algorithm has the highest run time of all the algorithms.

4.2 Comparision of the Results

Table 4.5: Result of the first run for all the algorithm.

RUN1	HYBRID(ABC-TABU)	ABC	TABU	GENETIC
RUN TIME (seconds)	666.21	3039.25	1106.40	4368.06
MEMORY USED(KB)	688	664	8951	648
PENALTY WEIGHT	3830	1640	4030	1760

At the first run, the hybridized algorithm outperforms the other algorithms in the processing time. While the hybridized used 666.21seconds which is 2373.04(64.04%) seconds less than ABC's time of 3039.25 seconds, 439.83(24.83%) less than TABU's time of 1106.40 seconds and 3701.85(73.53%) seconds less than genetic time of 4368.06 seconds. The time complexity is the major consideration of metaheuristic algorithm as discussed in Landa Silva 2003 PhD thesis and Ulker 2013 PhD thesis among other researchers.

Table 4.6: Result of the second run for all the Algorithm.

RUN2	**HYBRID(ABC-TABU)**	**ABC**	**TABU**	**GENETIC**
RUN TIME	692.03	3115.20	1111.48	4293.79

MEMORY USED	689KB	666KB	17260KB	650KB
PENALTY WEIGHT	3890	1640	3950	1960

At the second run, the hybridized algorithm outperforms the other algorithms in the processing time. While the hybridized used 692.03seconds which is 2423.17(63.65%) seconds less than ABC's time of 3115.20seconds, 419.45(23.26%) less than TABU's time of 1111.48 seconds and 3601.76(72.24%) seconds less than genetic time of 4293.79 seconds.

RUN3	**HYBRID(ABC-TABU)**	**ABC**	**TABU**	**GENETIC**
RUN TIME	661.32	3235.94	1094.55	4306.61
MEMORY USED	692KB	666KB	25602KB	650KB
PENALTY WEIGHT	3980	1460	3880	1890

Table 4.7 Result of the third run for all the algorithm.

At the third run, the hybridized algorithm outperforms the other algorithms in the processing time. While the hybridized used 661.32seconds which is 2574.46(66.06%) seconds less than ABC's time of 3235.94 seconds, 433.23(24.67%) less than TABU's time of 1094.55 seconds and 3645.29(73.38%) seconds less than genetic time of 4306.61 seconds.

RUN4	HYBRID(ABC-TABU)	ABC	TABU	GENETIC
RUN TIME	561.82	3034.79	1280.61	4193.06
MEMORY USED	692KB	666KB	33893KB	650KB
PENALTY WEIGHT	3800	1380	4170	1670

Table 4.8: Result of the fouth run for all the Algorithm.

At the fourth run, the hybridized algorithm outperforms the other algorithms in the processing time. While the hybridized used 561.82seconds which is 2472.97(68.76%) seconds less than ABC's time of 3034.79 seconds, 718.79(53.90%)less than TABU's time of 1280.61 seconds and 3631.24(76.37%) seconds less than genetic time of 4193.06 seconds.

RUN5	HYBRID(ABC-TABU)	ABC	TABU	GENETIC
RUN TIME	563.35	3096.62	1263.57	4185.75
MEMORY USED	695KB	666KB	42162KB	650KB
PENALTY WEIGHT	3850	1460	4070	2020

Table 4.9: Result of the fifth run for all the algorithm.

At the fifth run, the hybridized algorithm outperforms the other algorithms in the processing time. While the hybridized used 563.35seconds which is 2533.27(69.22%) seconds less than

ABC's time of 3096.62 seconds, 700.22(38.33%) less than TABU's time of 1263.57 seconds and 3622.24(76.28%) seconds less than genetic time of 4185.75seconds.

RUN6	**HYBRID(ABC-TABU)**	**ABC**	**TABU**	**GENETIC**
RUN TIME	555.04	3167.15	1532.64	4190.71
MEMORY USED	697KB	666KB	50612KB	650KB
PENALTY WEIGHT	3960	2490	4120	1730

Table 4.10: Result of the sixth run for all the algorithm.

At the sixth run, the hybridized algorithm outperforms the other algorithms in the processing time. While the hybridized used 555.04seconds which is 2612.11(70.18%) seconds less than ABC's time of 3167.15 seconds, 977.60(46.83%) less than TABU's time of 1532.64 seconds and 3635.67(76.61%) seconds less than genetic time of 4190.71 seconds.

Hybridisation of algorithms brings together the strengths of the individual algorithms and also few of the weakness (Nicholas, 2012) as shown in the values returned by the algorithms discussed.

Furthermore, the parameters set for each of the algorithms as stated above were set only to generate preliminary results. For instance, the number that was set for foraging cycle for each run (100 in this case) is usually in hundreds. This will exponentially increase the execution time of the algorithms to hours, thereby also increasing the difference between the execution time of the hybridised algorithm and the execution time of other algorithms.

4.3 Halstead Complexity measures Result

Table 4.11: Parameters used for measuring the computational complexity

Parameter	**ABC**	**TS**	**GA**	**HYBRID**
No of distinct operators (n_1)	27	25	21	22
Total number of operators (N1)	1290	1143	1379	1107
No of distinct operands (n_2)	30	21	23	21
Total number of operands (N2)	638	625	719	539
Length of program (N= N1+N2)	1928	1768	2098	1646
Program Vocabulary (n= n_1+n2)	57	46	44	43

Table 4.11 shows the parameters considered and their values used in the computation of the four algorithms complexity.

Table 4.12: Summary of Data Obtained using Healstead Parameters.

PARAMETERS	ABC	TS	GA	HYBRID
Program Length N=N1+N2	1928	1768	2098	1646
MemoryRequirement(volume) V = N*log2n	11245	9765.5	11453	8931.6
Program Difficulty	287.1	372.02	328.23	282.33
Program Effort $L=(2n_2)/(n_1N_2)$	3228672	3633057	3789614	2521697
ProgramSize(KB) M=Total Memory-(free+buffers+ Cache)	667KB	650KB	29747KB	692KB
Execution Time	3114.83s	3601.00s	1039.54s	616.63s
Lines of Code T=E/18	658	739	869	602
Intelligence of a program	3228439.5	3632998.3	3759218.5	2521658.6
No of bugs B=V/3000	3.75	3.98	3.82	2.98

Table 4.12 shows the comparison of the four algorithms, the computed results are presented in the table. It should be noted that the four algorithms considered produced a feasible solutions as shown in the result table in appendix

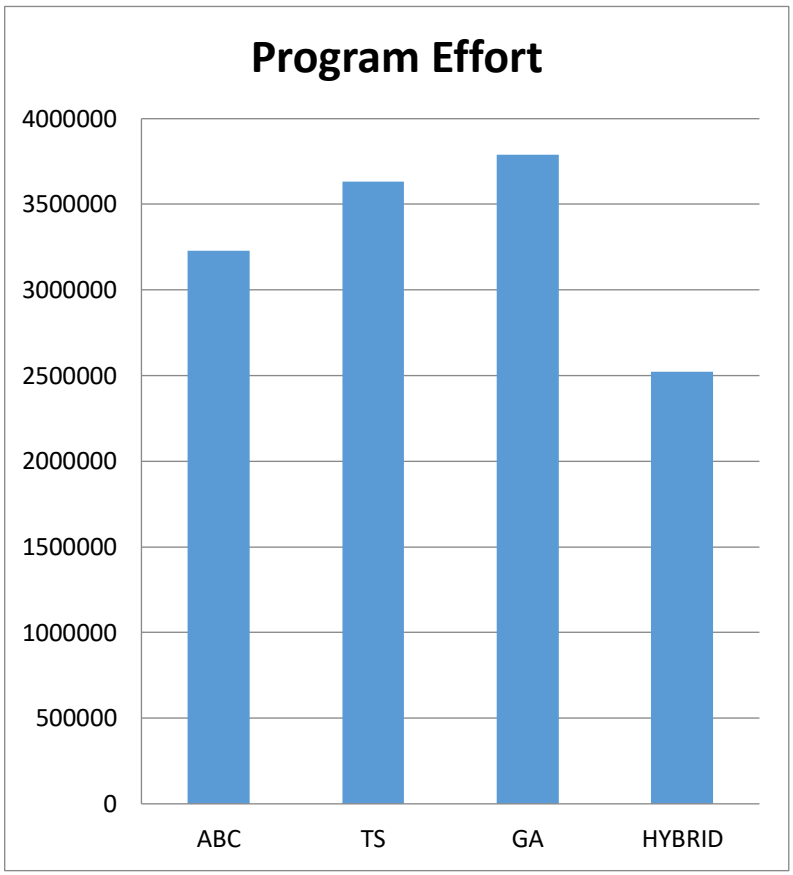

Figure 4.1: Bar chart showing Program effort of the four Algorithms

Figure 4.1 shows the graphical representation of the programming effort of all the algorithms and the hybrid has the least effort when compared to the Tabu, ABC and Genetic. Table 4.13 shows the program effort of ABC, TS, GA and the developed hybrid to be 3228672, 3633057, 3789614 and 2521697 respectively. This is the quantitative measure of the effort involved in the implementation of the four algorithms.

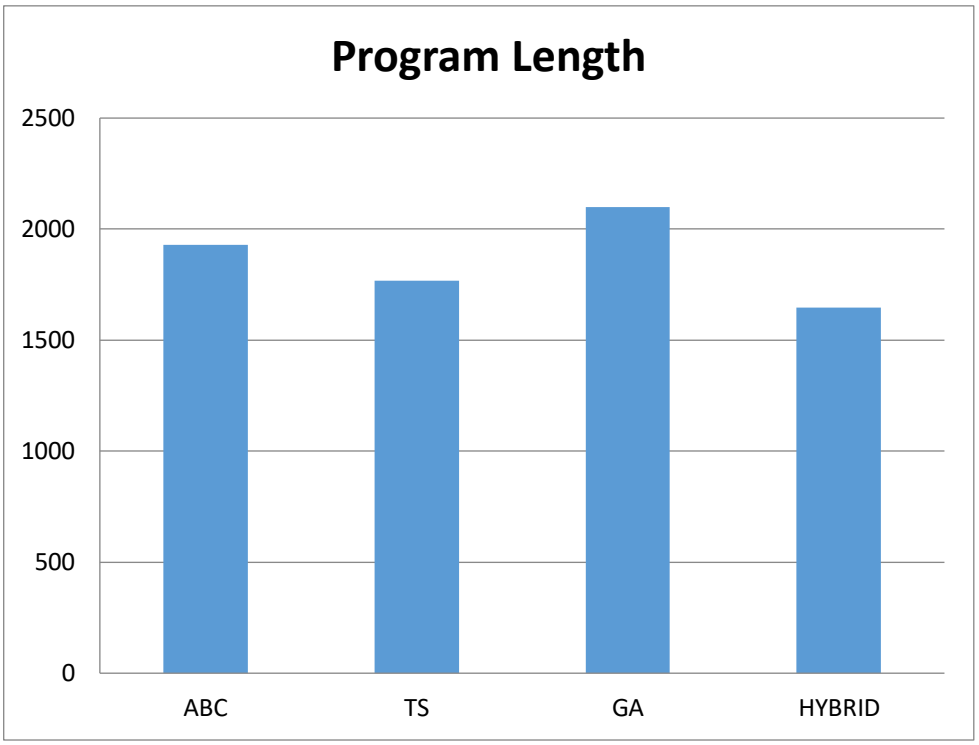

Figure 4.2: Bar chart showing Program length of the four Algorithms

This depicts that the program length for ABC, TS, GA and hybrid algorithms are 1928, 1768, 2098 and 1646respectively. The graph shows that the hybridised algorithm has the least lines of program for its allocation.

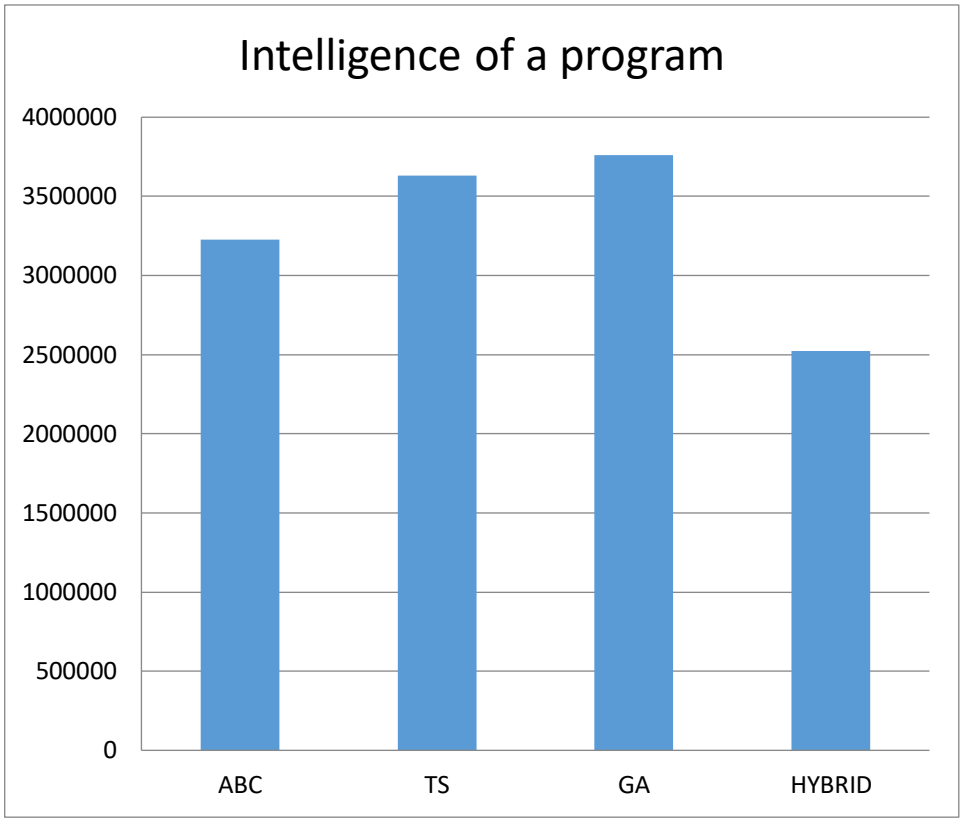

Figure 4.3: Bar chart showing Program length of the four Algorithms

This depicts that the program intelligence for ABC, TS, GA and hybrid algorithms are 3228439.5, 3632998.3, 3759218.5 and 2521658.6 respectively. The graph shows that the hybridised algorithm has the best program in term of intelligence for its allocation.

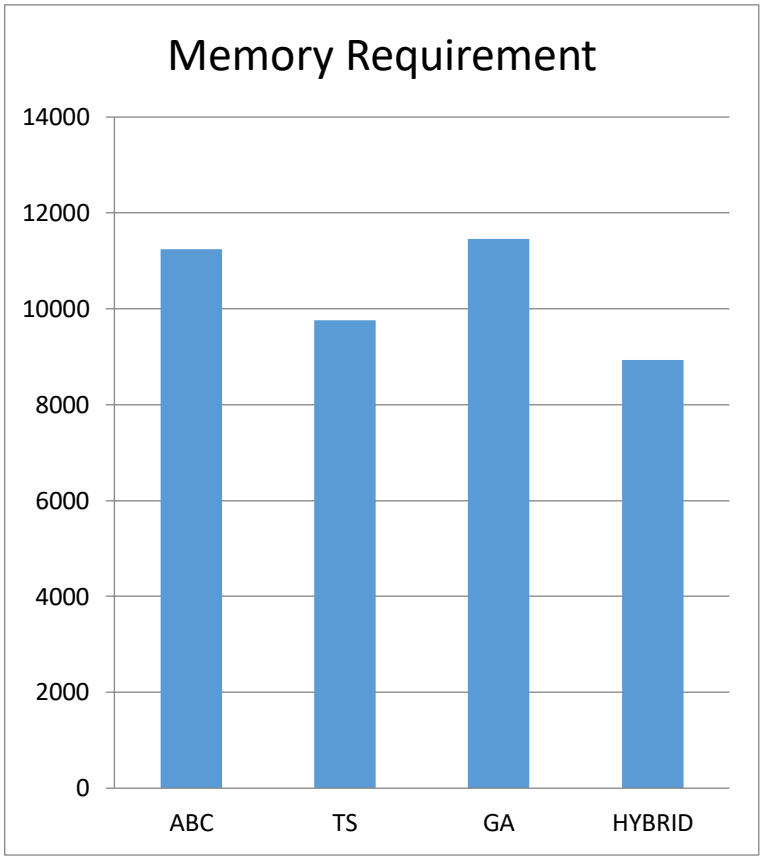

Figure 4.4: Bar chart showing memory requirements (Volume) of the four Algorithms

Figure 4.4 depicts the Average Memory Requirement of the four algorithms which shows that ABC, TS, GA and the developed hybrid takes 11245.0, 9765.5, 11453.0 and 8931.6. It indicates that the hybrid utilizes less average memory space in terms of memory requirement to generate the optimum solution.

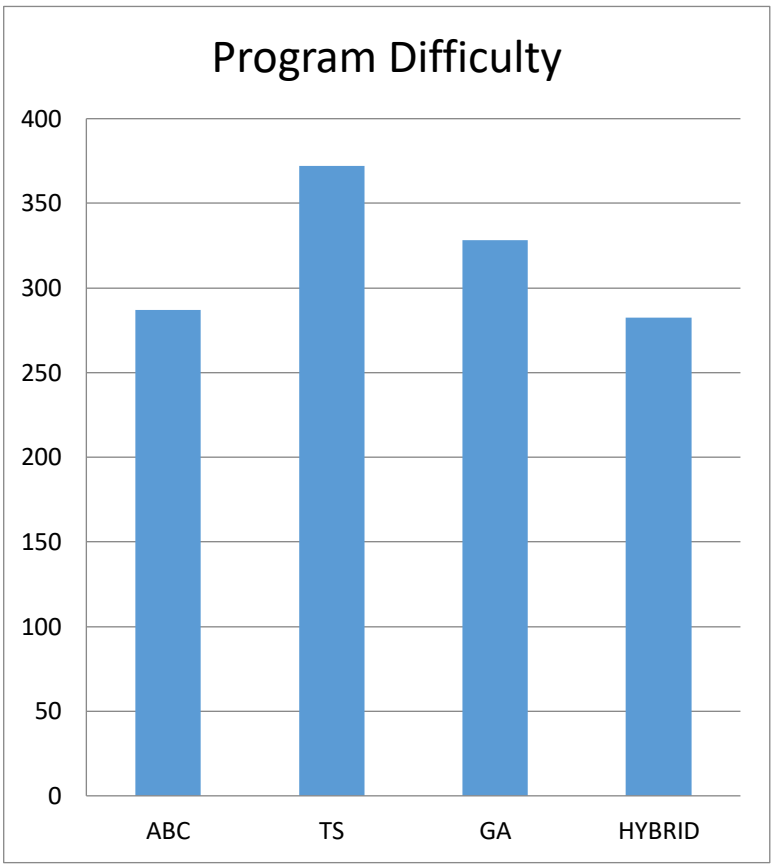

Figure 4.5: Bar chart showing Program difficulty of the four Algorithms

Difficulty of understanding the four programs are presented in figure 4.5 as 287.1, 372.02, 328.23 and 282.33 for ABC, TS, GA and the developed hybrid respectively.The most difficult algorithm is the Tabu search

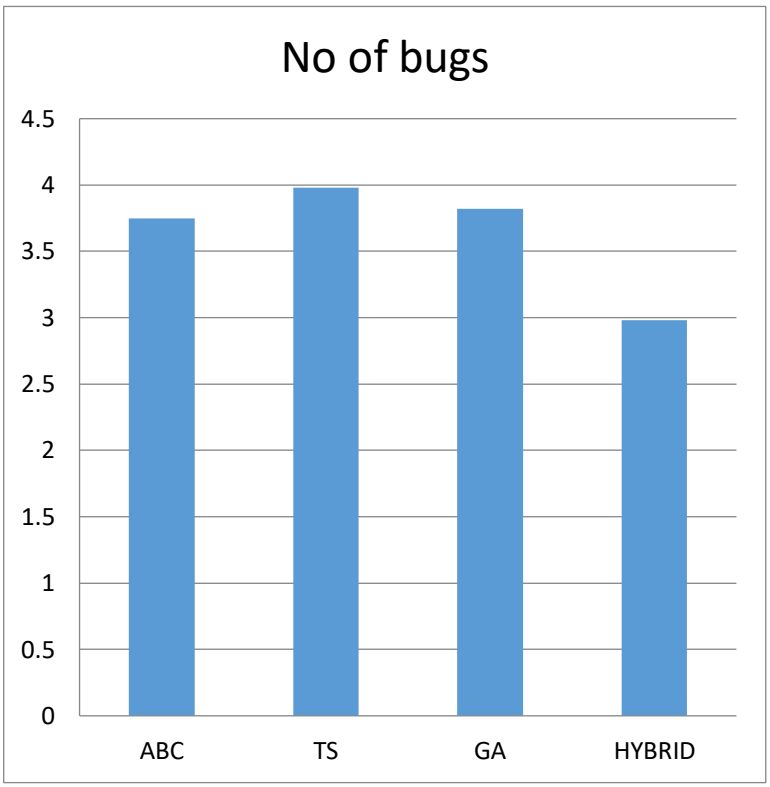

Figure 4.6: Bar chart showing number of bugs of the four Algorithms

Figure: 4.6 chart shows number of bugs in each of the program as 3.75, 3.98, 3.82 and 2.98 respectively. Tabu search algorithm has the highest no of bugs while the developed hybrid has the least.

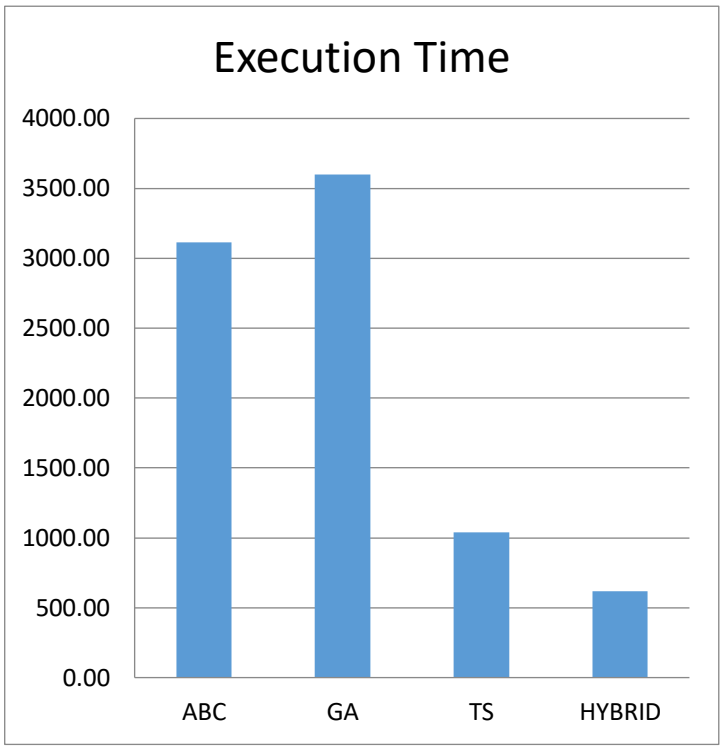

Figure 4.7: Bar chart showing execution time in second of the four Algorithms

The chart average execution time of theABC, TS, GA and hybrid algorithms are 3114.83s, 3601.00s, 1039.54s and 616.63s respectively.The developed hybrid returns a feasible office space allocation as an output in a lesser time compared with other algorithm.

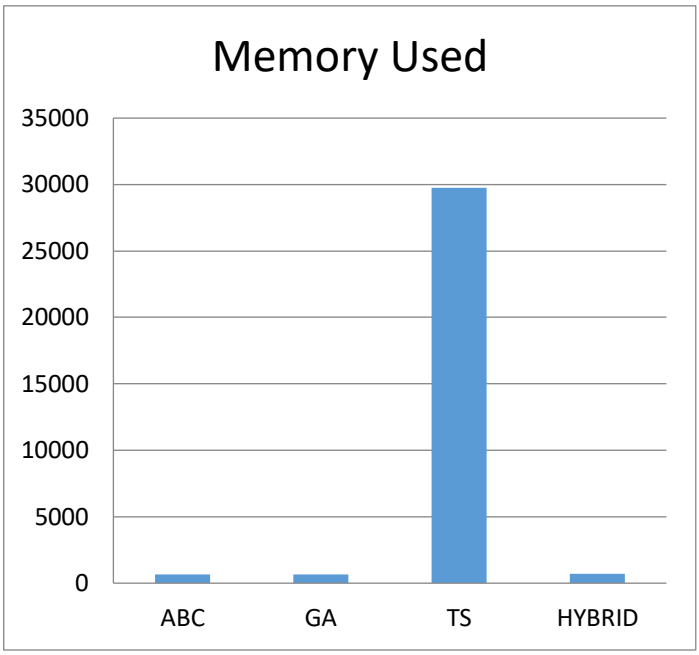

Figure 4.8: Bar chart showing memory used of the four Algorithms

Figure 4.8 depicts that the program size of ABC, TS, GA and hybrid algorithms are 667KB, 650KB, 29747KB and 692KB respectively, which shows that TS utilizes the least memory than ABC, GA and the developed hybrid.

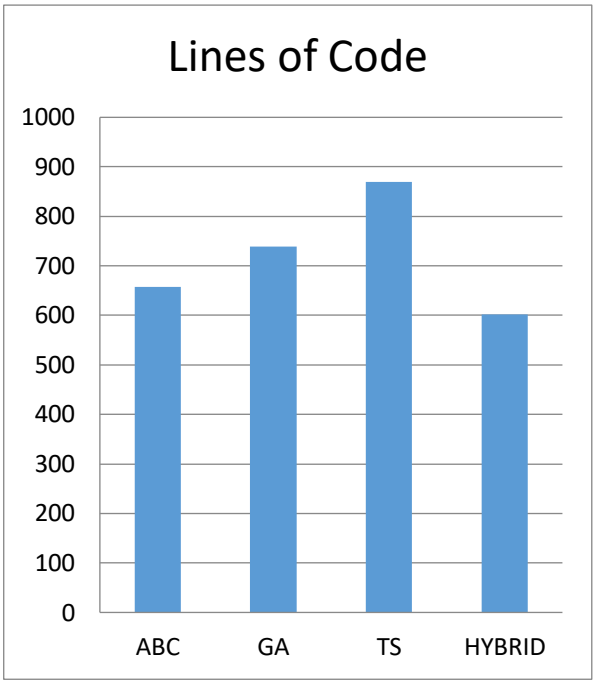

Figure 4.9: Bar chart showing lines of code of the four Algorithms

Figure 4.9 specify the lines of code which is the total number of lines of the executable codes in the program. It was 658, 739, 869 and 602 respectively for ABC, TS, GA and the developed hybrid.

From all the parameters used for computation and their values, the developed hybrid algorithm produced a feasible office space allocation using the least execution time, memory requirements, program effort, program length, lines of code and program intelligence.

4.5 System Specification

The test on office space allocation scheduling was run on a Laptop with the following configurations: Intel processor, Core i5, 4GRAM, 500GB HDD and Windows 8.1

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary

This study present method of office space allocation automation in tertiary institution, space is regarded as spatial resources within an organization/Institution. This research work aims at development of hybrid meta-heuristic algorithm in solving NP-hard combinatorial office space allocation problem (OSAP) by formulating a mathematical model for the problem, and hybridize populated based metaheuristic algorithm (ABC) with a local based metaheuristic algorithm (Tabu) to solve the stated problem. The OSAP has been attempted by several researchers using different methods ranging from the mathematical method, heuristic and meta-heuristic methods. The efficiency of these methods was based on the computational time complexity. Meta-heuristic is considered to be one of the best methods and this is the main focus of this research work. The work done in this research work confirmed the superiority of the hybrid algorithm over other algorithm used, the hybridised returned lesser average time and memory space used when compared with other algorithm. The dataset for the study was from a primary source (Faculty of Communication and Information Sciences, University of Ilorin). The implementation was carried out using C# programming language. The result of the OSAP shows that the hybridized algorithm(TABC) allocate at lesser time than the ABC,Tabu and the Gentic algorithm.

5.2Conclusion

The techniques and deployed were Tabu-Search algorithm (TB), Artificial Bee Colony algorithm (ABC), Genetic algorithm and hybridized algorithms (TABC). The purpose of the hybridization is to synergize the strengths of underlying algorithms for possible improved performance using Time Complexity. Consider the aim of this research work, the hybridized algorithm was able to allocate all the entities to their respective office in lesser time compare to the other algorithm when not hybridized, this serve as an improvement on the two other algorithms when compared independently.

5.3 Recomendation

The result of this study shows that the hybridized meta-heuristic algorithm outperformed other algorithm as used in the literature and the use of population-based algorithm also enhanced the performance. The researcher recommends that in future research work the use of more hybridized algorithm should be encouraged in other to have wider bases of comparism between several hybridized algorithms, consider more population-base algorithm in solving OSA problem and also consider hybridised heuristic and metaheuristic as well.

5.4 Contributions to knowledge

(i) The hybridized algorithm has given a very significant improvement in terms of the time used in the six runs, compare to other three algorithms which is the major consideration in this study. This shows that the hybridized algorithm of ABC and TABU Search give better result in solving Office Space Allocation Problem.

(ii) The study mathematically modelled the constraints used in solving office space allocation problem.

REFERENCES

Adewumi, A.O. & Ali, M.M (2010). A multi-level Genetic Algorithm for a multi stage space Allocation problem, Mathematical and Computer Modulling: *An international journal of computing, v.51.n.1-2, p.*109-126.

Ariyo, S.A (2013) *Studies of Heuristics for Hostel Space Allocation problem*. Unpublished MSc thesis Submitted to the School of Mathematics, Statistics and Computer Science, University of Kwazulu-Nata Durban, South Africa.

Al-Betar, M.A., Awedallah, M.A., Khader,A.T, Woon, P.C., & Doush, L.Y.(2013). A modified harmony search for office space allocation. *The 6th International Conference on Information Technolog. Malaysia: University of Sains.*

Awadallah, M. A., Khader, A. T, Al-Betar, M. A. & Woon, P.C. (2012). Office-space-allocation problem using harmony search algorithm. In T. Huang, Z. Zeng, C. Li, & C. Leung(Eds)*ICONIP (2), volume 7664 of Lecture Notes in Computer Science. (Pp.* 365-374). Springer.

Ayachi, R; kamati, M.k; Souri, A.& Borne, K. (2010). A Genetic Algorithm to solve the container space allocation problem. *International Conference on Computational Intelligence and Vehicular System* (CIVS) **LACS. Ecole Nationales des ingenieurs de tunis Bp 37 Le Belvelere 1002 Tunis, Tunisia.

Ashram, R. (2009). *Binary fuzzy goal programming for effective utilization of IT professional model for human resources allocation in health organization*. India: Institute of Engineering and Management, Ashram Campus.

Blum, C & Roli, A.(2003) Metaheuristics in combinatorial optimization: Overview and conceptual Comparison. *ACM Computing Surveys*. Vol. 35, No. 3, September 2003.

Burke, E. K., Cowling, P. & Landa, S. (2001a). Hybrid population-based metaheuristic approaches for the space allocation problem. *In Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001), pp* 232-239.

Burke, E.K. & Varley, D.B. (1998). Space allocation: An analysis of higher education requirements. In E.K. Burke & M.W. Carter. (Eds.) The practice and theory of automated timetabling II: Selected papers from the 2nd International Conference on the practice and theory of automated timetabling. Springer, pp.20-33.

Burke, E.K., Cowling, P., Landa, S. J., & Mecolumn, B.(2001). Three methods to automate the space allocation process in UK universities. Lecture notes in computer science. Springer, Heldelberg (2001). Vol 2019, pp. 254-273.

Burke, E.K., Cowling, P.,& Landa, S. J.(2007) Asynchronous cooperative local search for office space allocation .Informs journal on computing (2007*) pubsonline.informs.org*

Beyrouthy, C. (2008). *Models, solution methods and threshold behaviour for the teaching space allocation problem.* Unpublished Ph.D Thesis, School of Computer Science and Information Technology, University of Nottingham.

Beyrouthy, C., Burke, E.K., McCollum, B., McMullan, P., Landa, S. & Parkes, A.J. (2009). Towards improving the utilization of university teaching space. *The journal of operational Research Society,* 60;130-143.

Bremerman, H.(1962) *Optimization through evolution and recombination*. Barkely: University of California.

Bolaji, A.L, Micheal, L & Shola P.B (2017) Optimization of Office - Space Allocation Problem Using Artificial Bee Colony Algorithm. In: Tan Y., Takagi H., Shi Y.(Eds). Advances in Swarm Intelligence. ICSI 2017. LectureNote In Computer Science, Vol 10385. Springer, Cham.

Chieh, Y. & Ming,C. (2010). Applying two stages simulated Annealing algorithm for shelf space allocation problems. *Proceeding of the world Congress on engineering* 2010 vol III WCE 2010. June 30-July 2, 2010. London,UK.

Cook, A.(1971). *The complexity of theorem proving procedure* . Toronto: Department of Computer Science, University of Toronto.

Eberhart ,R. (1998). Proposed inertial weight value to the original PSO Algorithm. India: Department of Electrical Engineering, Pudue School of Engineering Technology.

Emmanuel,K. (2003). Bayesian subset simulation (BSS) on space allocation . Harvard University. *https//scholar.harvard.edu/../progressive.*.

Frank , A. (2015). Proposed linear programming model to solve allocation problem.Amsterdam: Department of Computer Science, Friedrich- Alexandra-University, Amsterdam,

Frimping, F.O. & Owusu, A. (2015). Allocation of classroom space using linear programming (A case study of Prember Nurses Training Collge Kumasi). *Journal of Economics and sustainable Development.* Vol 6.(2), 2-16.

Garey, M. R. & Johnson, D. S. (1979). Computers and Intractability – A guide to the Theory of NP – Completeness W. H. Freeman

Giannikos, J., El-Darzi, E. & Lees, P. (1998). *An integer goal programming model to allocate offices to staff in an academic institution.* London, UK.: London Press.

Glover, F. W. & Laguna, M. (1997). *Tabu search*. Norwell, U.S.A Kluwer Academic Publishers.

Goldberg, F. (1989). *Introduction to genetic algorithm* .Cambridge: Massachuselts Istitute 0f technology.UK.

Hansen, M.P. (1998). *Metaheuristics for multiple objective combinatorial optimization.* Unpublished PhD thesis. School of Mathematics, Statistics and Computer, University of Kwazulu-Nata Durban, South Africa

Islam, T., Shahriar, Z., Perves, M., Hasan, M. (2016). University Time tabu generator using

Tabu search. Journal of Computer and Communications. 4 pp.28-37.

Jain, K. S. & Singh, P. V. (2003). *Water Resource Systems Planning and Management*. UK, Elsevier Pub.

Jeel, L.O. (2013). Model and solution to campus parking space allocation problem. *International Journal of the Operational Research Society*, 46(6), 713-720.

Jyoti, S. R. & Shankar, S. (2014). Genetic algorithm and hybrid genetic algorithm for space allocation problem *International Journal of Computer Application* 95(4), 975-987 .

Karaboga, D. (2005). *An idea based on honey bee swarm for numeric optimization*. Netherland: Computer Engineering Department, Erciyes University.

Landa Silva, J.D (2003). *metaheuristics and multi objectives approaches for space llocation*. Unpublished PhD Thesis. University of Nottingham.UK.

Landa, S.J. & Burke, E.K.(2007). Asynchronous cooperative local search for the office space-allocation problem. *INFORMS Journal on Computing* 19(4), 575–587

Lee, J. (2004). *A first Course in Combinatorial Optimization*. Cambridge: Cambridge University Press, UK.

Lagunna, H. (2003). *Principle of scatter search algorithm* . Colorado: University of Colorado Boulder.

Mladenovic, N. & Hansen, P. (1997). Variable neighbourhood search. *Journal of Computers and Operations Research.* 24(11),1097-1100.

Man, E. & Michalewlez, Y. (1999). Introduction to genetic algorithm. Cambridge: Massachuselts Institute of Technology.

Nicholas, B. (2012). *The relevance of efficiency to different theories of society:* Economics of the welfare state (5th ed.). Oxford: Oxford University Press

Nyonyi, Y. (2010). *Modelling of hostel space allocation.* Nigeria: African Institute for Mathematical Sciences (AIMS).

Osman, O. & Kelly, L. (1996). Modern heuristic method.Beurit: American University of Beurit.(AUB).

Ozgur, U. & Landa, S.J. (2011). *A 0/1 integer programming model for the office space allocation problem. Automated Scheduling Optimization and Planning (ASAP)* Nottingham: university of Nottingham.

Ozgur, U. & Landa S. (2012). Designing difficult office space allocation problem instance with mathematical programming. *Proceedings of the 10th International Conference on Experimental Algorithm,* Crete Greece in SEA, pp 280-291.

Ozgur, U. & Landa, S.(2012). Evolutionary local search for solving the office space allocation problem. In *WCCI 2012 IEEE World Congress in Computational Intelligence*, June 10-15, 2012. Brisbane Australia pp. 3573-3580.

Ozgur, U. & Landa, S.J. (2011). Designing difficult Office space allocation problem instance with mathematical programming with Automated Scheduling Optimization and Planning (ASAP). Nottingham: University of Nottingham, Jubilee Campus.

Ozgur, U. (2013). *Office space Allocation by using mathematical programming and metaheuristic*. Unpulished PhD Thesis, University of Nottingham, UK.

Lopes,R. & Girimonte, D. (2010). The office-space-allocation problem in strongly hierarchized organizations. *Journal of Evolutionary Computation in Combinatorial Optimization,* 60(22), 143-153.

Rayward, S., Osman, I. Reeves, C & Smith, G. D. (Eds) (1996). *Modern Heuristic Search Method*. Wiley: westley Publishers.

Reeves, C. R. (Ed) (1995). Modern heuristic techniques for combinatorial problems. New York, McGraw Hill.

Remi, D. (2009). *Adaptive meta-models for crack characterization in eddy-current testing using particle swam*. France: Department of Electromagnetism Laboratories .University of Paris.

Resende, M. G..& Ribeiro, C. C. (2003). Greedy randomized adaptive Search Procedures. In F. Glover & G. Kochenberger (Eds) H*andbook of metaheuristics* (pp.219-249). Kluwer: Academic Publishers.

Ritzman, L., Bradford, J. & Jacobs, R. (1980). A multiple objective approach to space planning for academic facilities. *Journal of Management Science*. 25(9),895-906.

ROJAS, G.S. & TORRES, J.F. (2005). Genetic algorithm for designing bank office layouts. 19th*International Conference on Production Research (Industrial Engineering Department. Los Andes University. Carnera 1N ISA 10. Bogota Colombia)*.

Romuald, J & John S. G. (2006). *A genetic programming approach to the space layout planning problem. Key centre of Design computing*. Australia: Department of Architectural and Design Science, University of Sydney.

Sandeep,M. (2013). *A review of meta-heuristic approach to solve facility layout problem.* India:Mechanical and Engineering Department, National Institute of Technology (NIT) kurushetra, Haryana.

Syam, P. W. & Al-Harkan, M. I(2010). Comparison of three meta heurisics to optimizehybrid flow shop scheduling problem with parallel machines. World Academy of Science, Engineering and Technology 62 (20),10-20.

University of Michigan. Research space guidelines (2012). URL http://www.provost.umich.edu/space/other/ResearchSpaceGuildelines.pdf.

Voss, E. (1999). *Heuristic and multi-objectives Approach for space allocation*.UK: University of Nottingham.

Weise, T. (2009). Global optimization algorithms – theory and application. Retrieved from http://www.it-weise.de/projects/book.pdf

APPENDIX A

			GENETIC.CIS					
STAFF NO	**DEPT.**	**CADRE**	**RUN1**	**RUN 2**	**RUN 3**	**RUN 4**	**RUN 5**	**RUN 6**
1	TELECOM	AL	30	30	30	26	30	21
2	TELECOM	AL	11	26	1	8	12	17
3	ICS	AL	16	18	12	7	40	14
4	ICS	AL	33	39	33	38	38	2
5	ICS	AL	2	38	2	32	2	30
6	ICS	AL	38	38	3	2	36	33
7	COMSC	AL	35	24	11	35	6	35
8	COMSC	AL	6	6	39	11	6	6
9	COMSC	AL	5	6	6	17	6	11
10	COMSC	AL	6	11	11	6	6	6
11	MASSCOM	AL	6	3	3	6	12	2
12	MASSCOM	AL	3	34	36	2	35	5
13	ICS	AL	1	28	2	2	38	5
14	TELECOM	AL	40	38	37	34	37	36
15	MASSCOM	AL	3	15	5	7	34	17
16	ICS	AL	36	33	38	2	33	8
17	ICS	AL	39	25	33	33	37	38
18	ICS	GA	3	38	38	38	2	16
19	ICS	GA	2	38	37	2	4	33
20	LIB	L1	4	32	37	3	2	39
21	LIB	L1	28	37	29	3	23	40
22	LIB	L1	23	38	24	37	28	28
23	LIB	L1	28	40	28	28	23	4
24	LIB	L1	29	33	28	28	28	39
25	TELECOM	L1	34	37	29	34	23	34
26	MASSCOM	L1	7	7	3	3	5	3
27	COMSC	L1	3	37	35	6	35	2
28	COMSC	L1	39	3	3	8	39	12
29	COMSC	L1	18	2	1	2	2	6
30	LIB	L1	2	29	40	40	37	29
31	LIB	L1	23	3	26	32	34	37
32	MASSCOM	L1	38	34	32	37	26	3
33	LIB	L1	4	34	32	4	39	17
34	ICS	L2	29	36	27	36	66	5
35	COMSC	L2	3	35	35	35	6	37
36	COMSC	L2	6	2	2	16	12	4
37	ICS	L2	2	2	39	33	38	38
38	LIB	L2	37	37	37	37	37	32
39	TELECOM	L2	26	39	26	37	26	5

40	MASSCOM	L2	7	7	15	15	7	15
41	ICS	L2	33	35	33	5	33	18
42	COMSC	L2	35	3	36	3	6	35
43	MASSCOM	L2	37	6	8	5	3	3
44	MASSCOM	L2	37	43	31	14	31	5
45	LIB	L2	36	29	37	14	29	6
46	COMSC	PROF	5	2	2	2	55	2
47	LIB	PROF	5	6	3	6	5	6
48	LIB	PROF	3	1	3	3	64	4
49	COMSC	PROF	4	4	3	2	71	5
50	COMSC	READER	5	5	5	17	5	17
51	ICS	READER	14	5	3	21	12	4
52	TELECOM	READER	2	5	2	1	14	25
53	LIB	READER	4	5	2	4	17	12
54	LIB	SL	24	31	24	31	31	5
55	LIB	SL	23	7	25	4	24	16
56	COMSC	SL	27	5	27	27	27	2
57	COMSC	SL	2	17	5	17	5	5
58	COMSC	SL	17	3	6	11	12	6
59	COMSC	SL	12	15	14	4	2	8
60	COMSC	SL	3	27	2	14	12	17
61	COMSC	SL	6	14	5	5	5	17
62	COMSC	SL	5	5	5	6	5	5
63	COMSC	SL	5	11	17	17	11	5
64	COMSC	SL	8	5	15	18	1	5

			ABC .CIS					
STAFF NO	**DEPT.**	**CADRE**	**RUN1**	**RUN2**	**RUN3**	**RUN4**	**RUN5**	**RUN6**
1	TELECOM	AL	26	30	39	39	1	31
2	TELECOM	AL	1	39	6	30	39	22
3	ICS	AL	3	1	4	25	6	4
4	ICS	AL	2	37	38	6	4	46
5	ICS	AL	2	37	32	37	17	30
6	ICS	AL	38	2	35	4	4	17
7	COMSC	AL	32	39	29	8	7	33
8	COMSC	AL	3	16	8	16	8	21
9	COMSC	AL	12	16	17	16	14	16
10	COMSC	AL	17	16	16	16	16	3
11	MASSCOM	AL	38	11	18	18	5	21
12	MASSCOM	AL	5	15	32	28	28	12
13	ICS	AL	2	6	37	32	35	16
14	TELECOM	AL	39	39	38	1	14	28

15	MASSCOM	AL	11	39	5	29	6	35
16	ICS	AL	6	4	36	26	26	3
17	ICS	AL	35	32	33	37	5	3
18	ICS	GA	4	2	28	37	2	10
19	ICS	GA	2	33	37	38	2	27
20	LIB	L1	40	3	30	6	2	27
21	LIB	L1	5	2	3	3	37	3
22	LIB	L1	3	7	7	3	34	3
23	LIB	L1	40	34	7	3	6	38
24	LIB	L1	4	40	14	34	3	34
25	TELECOM	L1	39	33	17	4	3	3
26	MASSCOM	L1	28	5	15	25	11	40
27	COMSC	L1	2	3	2	29	29	39
28	COMSC	L1	2	4	12	14	5	17
29	COMSC	L1	37	2	2	14	4	7
30	LIB	L1	40	40	34	6	40	33
31	LIB	L1	40	40	40	3	15	27
32	MASSCOM	L1	7	38	5	38	38	34
33	LIB	L1	40	22	27	26	30	14
34	ICS	L2	3	2	38	14	3	3
35	COMSC	L2	8	35	29	8	2	27
36	COMSC	L2	37	17	6	17	6	16
37	ICS	L2	35	35	33	6	37	37
38	LIB	L2	32	4	30	6	7	37
39	TELECOM	L2	40	3	4	39	39	12
40	MASSCOM	L2	32	3	15	28	11	5
41	ICS	L2	3	6	31	27	26	16
42	COMSC	L2	6	39	29	37	8	37
43	MASSCOM	L2	8	11	5	17	16	24
44	MASSCOM	L2	11	17	28	25	28	17
45	LIB	L2	4	5	31	24	24	6
46	COMSC	PROF	5	3	5	5	4	2
47	LIB	PROF	2	6	6	5	6	5
48	LIB	PROF	5	3	5	2	2	4
49	COMSC	PROF	3	2	5	4	3	4
50	COMSC	READER	5	5	5	5	5	3
51	ICS	READER	5	16	12	21	18	18
52	TELECOM	READER	3	14	25	5	2	15
53	LIB	READER	3	12	17	31	12	13
54	LIB	SL	3	3	4	2	2	5
55	LIB	SL	7	12	8	3	5	17
56	COMSC	SL	5	5	5	5	3	4
57	COMSC	SL	18	17	17	17	17	23
58	COMSC	SL	36	14	17	14	14	14
59	COMSC	SL	14	8	36	36	36	12
60	COMSC	SL	2	5	26	25	26	3
61	COMSC	SL	5	5	17	14	5	6
62	COMSC	SL	5	17	17	17	17	24
63	COMSC	SL	5	17	6	17	17	8
64	COMSC	SL	2	18	17	14	5	7

			ABC .CIS					
STAFF NO	**DEPT.**	**CADRE**	**RUN1**	**RUN2**	**RUN3**	**RUN4**	**RUN5**	**RUN6**
1	TELECOM	AL	26	30	39	39	1	31
2	TELECOM	AL	1	39	6	30	39	22
3	ICS	AL	3	1	4	25	6	4
4	ICS	AL	2	37	38	6	4	46
5	ICS	AL	2	37	32	37	17	30
6	ICS	AL	38	2	35	4	4	17
7	COMSC	AL	32	39	29	8	7	33
8	COMSC	AL	3	16	8	16	8	21
9	COMSC	AL	12	16	17	16	14	16
10	COMSC	AL	17	16	16	16	16	3
11	MASSCOM	AL	38	11	18	18	5	21
12	MASSCOM	AL	5	15	32	28	28	12
13	ICS	AL	2	6	37	32	35	16
14	TELECOM	AL	39	39	38	1	14	28
15	MASSCOM	AL	11	39	5	29	6	35
16	ICS	AL	6	4	36	26	26	3
17	ICS	AL	35	32	33	37	5	3
18	ICS	GA	4	2	28	37	2	10
19	ICS	GA	2	33	37	38	2	27
20	LIB	L1	40	3	30	6	2	27
21	LIB	L1	5	2	3	3	37	3
22	LIB	L1	3	7	7	3	34	3
23	LIB	L1	40	34	7	3	6	38
24	LIB	L1	4	40	14	34	3	34
25	TELECOM	L1	39	33	17	4	3	3
26	MASSCOM	L1	28	5	15	25	11	40
27	COMSC	L1	2	3	2	29	29	39
28	COMSC	L1	2	4	12	14	5	17
29	COMSC	L1	37	2	2	14	4	7
30	LIB	L1	40	40	34	6	40	33
31	LIB	L1	40	40	40	3	15	27
32	MASSCOM	L1	7	38	5	38	38	34
33	LIB	L1	40	22	27	26	30	14
34	ICS	L2	3	2	38	14	3	3
35	COMSC	L2	8	35	29	8	2	27
36	COMSC	L2	37	17	6	17	6	16
37	ICS	L2	35	35	33	6	37	37
38	LIB	L2	32	4	30	6	7	37
39	TELECOM	L2	40	3	4	39	39	12
40	MASSCOM	L2	32	3	15	28	11	5
41	ICS	L2	3	6	31	27	26	16
42	COMSC	L2	6	39	29	37	8	37
43	MASSCOM	L2	8	11	5	17	16	24
44	MASSCOM	L2	11	17	28	25	28	17
45	LIB	L2	4	5	31	24	24	6
46	COMSC	PROF	5	3	5	5	4	2

47	LIB	PROF	2	6	6	5	6	5
48	LIB	PROF	5	3	5	2	2	4
49	COMSC	PROF	3	2	5	4	3	4
50	COMSC	READER	5	5	5	5	5	3
51	ICS	READER	5	16	12	21	18	18
52	TELECOM	READER	3	14	25	5	2	15
53	LIB	READER	3	12	17	31	12	13
54	LIB	SL	3	3	4	2	2	5
55	LIB	SL	7	12	8	3	5	17
56	COMSC	SL	5	5	5	5	3	4
57	COMSC	SL	18	17	17	17	17	23
58	COMSC	SL	36	14	17	14	14	14
59	COMSC	SL	14	8	36	36	36	12
60	COMSC	SL	2	5	26	25	26	3
61	COMSC	SL	5	5	17	14	5	6
62	COMSC	SL	5	17	17	17	17	24
63	COMSC	SL	5	17	6	17	17	8
64	COMSC	SL	2	18	17	14	5	7

			HYBRID.CIS					
STAFF NO	**DEPT.**	**CADRE**	**RUN1**	**RUN 2**	**RUN 3**	**RUN 4**	**RUN 5**	**RUN 6**
1	TELECOM	AL	8	33	8	33	29	57
2	TELECOM	AL	3	21	24	8	4	60
3	ICS	AL	6	28	43	38	24	24
4	ICS	AL	1	65	18	1	37	15
5	ICS	AL	11	6	21	16	8	29
6	ICS	AL	6	6	6	15	47	35
7	COMSC	AL	39	23	23	11	28	4
8	COMSC	AL	15	23	2	11	16	47
9	COMSC	AL	57	15	39	6	33	1
10	COMSC	AL	7	39	34	4	30	26
11	MASSCOM	AL	2	37	11	2	6	65
12	MASSCOM	AL	2	3	63	60	34	11
13	ICS	AL	35	24	7	19	3	28
14	TELECOM	AL	4	8	47	24	4	18
15	MASSCOM	AL	2	26	1	39	26	6
16	ICS	AL	34	35	40	28	7	15
17	ICS	AL	37	43	16	21	37	23
18	ICS	GA	6	43	16	16	7	39
19	ICS	GA	37	4	33	23	38	7
20	LIB	L1	40	2	60	7	1	23
21	LIB	L1	47	34	30	30	2	2
22	LIB	L1	33	38	35	7	1	30
23	LIB	L1	16	2	65	34	2	37

24	LIB	L1	40	29	38	37	40	16
25	TELECOM	L1	38	60	26	43	54	3
26	MASSCOM	L1	26	1	28	18	15	34
27	COMSC	L1	28	18	34	40	28	40
28	COMSC	L1	23	11	29	4	16	47
29	COMSC	L1	18	30	3	16	11	38
30	LIB	L1	65	40	4	47	2	33
31	LIB	L1	63	19	30	26	23	21
32	MASSCOM	L1	29	37	37	3	57	8
33	LIB	L1	43	57	15	35	21	19
34	ICS	L2	66	66	66	66	66	66
35	COMSC	L2	15	39	39	6	33	26
36	COMSC	L2	1	35	33	28	35	38
37	ICS	L2	6	7	7	54	38	28
38	LIB	L2	43	57	54	37	2	37
39	TELECOM	L2	8	33	26	33	39	57
40	MASSCOM	L2	26	1	37	39	15	8
41	ICS	L2	37	24	6	19	8	15
42	COMSC	L2	28	11	29	6	30	1
43	MASSCOM	L2	29	26	28	2	26	6
44	MASSCOM	L2	2	37	1	60	6	11
45	LIB	L2	19	29	30	35	2	30
46	COMSC	PROF	55	64	64	64	55	64
47	LIB	PROF	64	55	55	62	62	62
48	LIB	PROF	62	62	62	55	64	55
49	COMSC	PROF	71	71	71	71	71	71
50	COMSC	READER	27	27	36	45	45	14
51	ICS	READER	31	46	61	41	61	17
52	TELECOM	READER	46	17	5	22	46	36
53	LIB	READER	61	5	14	31	31	45
54	LIB	SL	17	5	42	36	5	52
55	LIB	SL	17	5	51	42	22	22
56	COMSC	SL	42	61	41	27	14	41
57	COMSC	SL	5	22	31	14	46	27
58	COMSC	SL	52	52	52	5	27	31
59	COMSC	SL	36	14	46	17	51	42
60	COMSC	SL	51	27	45	61	14	61
61	COMSC	SL	22	42	31	46	36	51
62	COMSC	SL	5	31	27	51	61	5
63	COMSC	SL	14	36	31	14	41	46
64	COMSC	SL	70	70	70	70	70	70

APPENDIX B

ABC Program: Distinct Operators and Operands used for the Complexity

S/N	OPERATOR	FREQUENCY	OPERANDS	FREQUENCY
1	{}	179	USING_SYSTEM	41
2	Int	85	YOUR_OFICEF	2
3	=	238	STAF E ID	12
4	+=	19	OFFICE _ID	38
5	\n	7	I	46
6	+	226	OFFICE GROUP	6
7	\r\n	4	LIST	47
8	&&	17	OFFICE TOILET	9
9	>	29	OFFICE_CAPACITY	22
10	<	29	OFFICE_PROPERTIES	10
11	Foreach	16	STAFF LIST	26
12	If	63	ID	166
13	Else	12	OFFICE_PROXIMITY	15
14	For	33	RESOURCES	6
15	I++	18	TYPE	56
16	==	33	THREADING	11
17	GOTO	1	TABLE	42
18	++k	3	DEPT	13
19	Catch	11	CADRE	12
20	Console.Write	59	ALLOCS.LENGHT	12
21	STRING	54	THIS.CADRE A	2
22	NAME SPACE.abc	7	THIS.CADRE B	2
23	INPUT SET.CS	1	THIS.CADRE C	4
24	BREAK	8	STAFF NAME	4
25	RETURN	41	OCCUPIED	2
26	SsTATIC	39	My OFFICE	2
27	PUBLIC	58	K	14
28			staffGroups	4
29			0	7
			1	5
	n1=27	**N1=1290**	n2= 30	**N2=638**

Tabu Search: Distinct Operators and Operands used for the Complexity

S/N	OPERATOR	FREQUENCY	OPERANDS	FREQUENCY
1	{}	163	USING SYSTEM	41
2	Int	107	Penalty	18
3	=	278	STAFF ID	11
4	+=	18	OFFICE ID	34
5	\n	6	I	48
6	+	185	OFFICE GROUP	6
7	\r\n	3	LIST	44
8	&&	18	OFFICE TOILET	9
9	>	28	OFFICE CAPACITY	26
10	<	27	OFFICE PROPERTIES	10
11	Foreach	14	STAFF LIST	41
12	If	40	ID	154
13	Else	9	OFFICE PROXIMITY	15
14	For	47	RESOURCES	6
15	I++	13	TYPE	54
16	==	33	THREADING	10
17	Console.Write	13	TABLE	42
18	++k	3	DEPT	15
19	Name space.abc	7	CADRE	12
20	Input set.cs	1	K	14
21	Break	8	0	15
22	Return	42		
23	Public	21		
24	String	54		
25	catch	5		
	n1=25	**N1=1143**	n2=21	**N2=625**

Genetic Algorithm: Distinct Operators and Operands used for the Complexity

S/N	OPERATOR	FREQUENCY	OPERANDS	FREQUENCY
1	{}	179	USING SYSTEM	41
2	Int	152	Penalty	21
3	=	346	STAF E ID	11
4	+=	19	OFFICE ID	37
5	\n	6	I	48
6	+	207	OFFICE GROUP	6
7	\r\n	4	LIST	38
8	&&	17	OFFICE TOILET	9
9	>	50	OFFICE CAPACITY	22
10	<	61	OFFICE PROPERTIES	9
11	Foreach	16	STAFF LIST	19
12	If	63	ID	172
13	Else	12	OFFICE PROXIMITY	15
14	For	52	RESOURCES	6
15	I++	18	TYPE	67
16	==	39	THREADING	11
17	Inti	29	TABLE	52
18	++k	3	DEPT	13
19	Name space.abc	7	CADRE	14
20	RETURN	41	DATATABLE	13
21	PUBLIC	58	ARRAY	21
22			INSERTDATA	57
23			0	17
	n1=21	**N1= 1379**	**n2=23**	**N2= 719**

Hybrid Program: Distinct Operators and Operands used for the Complexity

S/N	OPERATOR	FREQUENCY	OPERANDS	FREQUENCY
1	{}	181	USING SYSTEM	41
2	Int	165	penalty	14
3	=	188	STAF E ID	11
4	+=	19	OFFICE ID	28
5	\n	8	ID	34
6	+	146	OFFICE GROUP	14
7	\r\n	3	LIST	39
8	&&	19	OFFICE TOILET	9
9	>	25	OFFICE CAPACITY	18
10	<	27	OFFICE PROPERTIES	10
11	For each	16	STAFF LIST	34
12	If	45	ID	97
13	Else	16	OFFICE PROXIMITY	15
14	For	45	RESOURCES	6
15	I++	22	TYPE	62
16	==	45	THREADING	11
17	*	5	TABLE	40
18	++k	3	DEPT	13
19	CONSOLE.WRITE	17	CADRE	10
20	RETURN	26	INSERTDATA	12
21	PUBLIC	29	0	21
22	STRING	57		
	n1=22	1107	n2=21	539

APPENDIX C

ABC SOURCE CODE

Allocation.CIS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace abc_osap
{
   class Allocation
   {
      public int staffID { get; set; }
      public int officeID { get; set; }
            public Allocation (int staff, int office)
      {
         staffID = staff;
         officeID = office;
      }
      public string Print()
      {
         string to_ret = "";
         string staff_rec = InputSet.GetStaffById(staffID).Display();
         to_ret += staff_rec + "\n";
         Console.Write(" ---- Allocated to: ");
         to_ret += " ---- Allocated to: " + "\r\n";
         string office_rec = InputSet.GetOfficeById(officeID).Display();
         to_ret += office_rec + "\r\n";
         Console.Write("\n");
         return to_ret;
      }
   }
}
```

InputSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
namespace abc_osap
{
   class InputSet
   {
      static Office[] offices;
      public static List<Office> officesList = new List<Office>();
      public static Dictionary<string, List<Office>> officeGroups = new
Dictionary<string,List<Office>>();

      static Staff[] staff;
```

```
        static int[] officeGrp;
        public static List<Staff> staffList = new List<Staff>();
        public static Dictionary<string, List<Staff>> staffGroups = new Dictionary<string,
List<Staff>>();

        public static void DeclareArray(int numOff)
        {
```



```
    //numSol is the number of office in the inputset
            officeGrp = new int[numOff];
        }

        public static void Load()
        {
            int i;

            MySQLDatabase myDb = new MySQLDatabase();
            //selecting data from table
            DataTable records;
            try
            {
                //selecting office records
                //Console.WriteLine("1");
                records = myDb.Select(null, "offices");//null means to fetch all data from the table
                //Console.WriteLine("2");
                if (records != null && records.Rows.Count > 0)
                {
                    offices = new Office[records.Rows.Count];
                    i = 0;
                    foreach (DataRow row in records.Rows)
                    {

                        //Console.WriteLine(row["id"].GetType());
                        //string d = row["type"].ToString();
                        offices[i] = new Office();

                        offices[i].id = (int)row["id"];
                        offices[i].type = row["type"].ToString();
                        offices[i].properties = row["properties"].ToString();
                        offices[i].capacity = (int)row["capacity"];
                        offices[i].toilet = row["toilet"].ToString() == "" ? 0 : (int)row["toilet"];
                        offices[i].resources = (int)row["resources"];
                        offices[i].proximity = row["proximity"].ToString();

                        officesList.Add(offices[i]);
                        if (!officeGroups.ContainsKey(offices[i].type))
                            officeGroups[offices[i].type] = new List<Office>();
                        officeGroups[offices[i].type].Add(offices[i]);
                    }
                }
                else
                    Console.WriteLine("No office record found!");
```

```
        //System.Threading.Thread.Sleep(1000000);

        //selecting staff records
        //Console.WriteLine("3");
        records = myDb.Select(null, "staff");//null means to fetch all data from the table
        //Console.WriteLine("4");
        if (records != null && records.Rows.Count > 0)
        {
            staff = new Staff[records.Rows.Count];
            i = 0;
            foreach (DataRow row in records.Rows)
            {
                staff[i] = new Staff();
                staff[i].id = Convert.ToInt16(row["id"]);
                staff[i].typeId = Convert.ToString(row["type_id"]);
                staff[i].cadre = Convert.ToString(row["cadre"]);
                staff[i].staffName = Convert.ToString(row["staff_name"]);
                staff[i].dept = Convert.ToString(row["dept"]);

                staffList.Add(staff[i]);

                //this part might not be needed
                if (!staffGroups.ContainsKey(staff[i].typeId))
                    staffGroups[staff[i].typeId] = new List<Staff>();
                staffGroups[staff[i].typeId].Add(staff[i]);
                //this part might not be needed
            }
        }
        else
            Console.WriteLine("No staff record found!");

    }
    catch (Exception e)
    {
        Console.WriteLine("Error selecting: " + e.Message);
        System.Threading.Thread.Sleep(1000000);
    }
}

public static Office GetOfficeById(int id)
{
    foreach (Office office in officesList)
    {
        if (office.id == id) return office;
    }
    return null;
}

public static void loadOfficeCapacity()
{
    foreach (Office office in officesList)
    {
```

```
            Program.officeCapacity[office.id] = office.capacity;
        }
    }

    public static int GetOfficeIndexById(int id)
    {
        for (int i = 0; i < officesList.Count(); i++)
        {
            if (officesList[i].id == id) return i;
        }
        return -1;
    }

    public static Staff GetStaffById(int id)
    {
        foreach (Staff staff in staffList)
        {
            if (staff.id == id) return staff;
        }
        return null;
    }

    public static int getOfficeByTypeName(string typey, string dept)
    {
        Found:
        int k = 0;
        foreach (Office office in officesList)
        {
            if (office.type == typey)
            {
                if (office.resources == 1)
                {
                    if (crossCheck(office.id) && checkCapacity(office.id) > 0 &&
checkWhoIsInTheOffice(office.id, dept) && isItProfSuit(typey, office.toilet))
                    {
                        officeGrp[k] = office.id;
                        k += 1;
                    }
                }
            }
        }
        //Console.WriteLine(ox + " T:" + typey);
        if (k == 0)
        {
            //Console.WriteLine(k);
            ClearTypeRoom(typey);
            goto Found;
        }
        //System.Threading.Thread.Sleep(600);
        int rnd = Program.r.Next(0, k - 1);
        return officeGrp[rnd];
    }
```

```
        static void ClearTypeRoom(string typey)
        {
            for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
            {
                Office xx = GetOfficeById(Program.tmp[numTmp]);
                if (xx != null)
                {
                    if (xx.type == typey)
                    {
                        Program.tmp[numTmp] = 0;
                    }
                }
            }
        }

        static bool crossCheck(int officeid)
        {
            for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
            {
                if (officeid == Program.tmp[numTmp])
                {
                    return false;
                }
            }
            return true;
        }

        static int checkCapacity(int officeid)
        {
            return Program.officeCapacity[officeid];
        }

        static bool checkWhoIsInTheOffice(int officeid, string dept)
        {
            if (Program.officeWhoIsThere[officeid] == dept || Program.officeWhoIsThere[officeid]
== null)
            {
                return true;
            }
            return false;
        }

        static bool isItProfSuit(string typed, int toilet)
        {
            if (typed == "A" && toilet == 0)
            {
                return false;
            }
            return true;
        }

        static int[] ShuffleArray(int[] array)
        {
```

```
            Random r = new Random();
            for (int i = array.Length; i > 0; i--)
            {
               int j = r.Next(i);
               int k = array[j];
               array[j] = array[i - 1];
               array[i - 1] = k;
            }
            return array;
         }
      }
   }
```

```
/*


            //THIS PART IS TO TEST THE DATABASE CLASS
            string db = "vp_validation_system";
            string dbuid = "root";
            string dbpwd = "";

            //instantiating the db class
            MySQLDatabase myDb = new MySQLDatabase(dbuid, dbpwd, db);

            //selecting data from table - there shouldn't be any records for now
            System.Data.DataTable records;
            try
            {
               records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
               // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
               Console.WriteLine("All records inserted:");
               if (records != null && records.Rows.Count > 0)
               {
                  foreach (System.Data.DataRow row in records.Rows)
                  {
                     Console.Write(row["id"] + " || "); //use double quotes
                     Console.Write(row["code"] + " || ");
                     Console.WriteLine(row["used"].ToString());
                  }
               }
               else
                  Console.WriteLine("No record found!");
            }
            catch (Exception e)
            {
               Console.WriteLine("Error selecting: " + e.Message);
            }

            //insert record to table 'sample_codes'
```

```
        Dictionary<string, string>[] insertData = new Dictionary<string, string>[3]; //An array of dictionaries, each dictionary represents a row to be inserted
        insertData[0] = new Dictionary<string, string>();
        insertData[0].Add("code", "563443y3uh87grr"); //column name, value in table 'sample_codes'
        insertData[0].Add("used", "0"); //another column name, value in table 'sample_codes'

        insertData[1] = new Dictionary<string, string>();
        insertData[1].Add("code", "fhbeiurhg34u23434");
        insertData[1].Add("used", "0");

        insertData[2] = new Dictionary<string, string>();
        insertData[2].Add("code", "fdhrbru3u9rwei9jcks");
        insertData[2].Add("used", "0");

        try
        {
           myDb.Insert(insertData[0], "sample_codes");
           myDb.Insert(insertData[1], "sample_codes");
           myDb.Insert(insertData[2], "sample_codes");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error inserting: " + e.Message);
        }

        //selecting data from table
        Console.WriteLine();
        try
        {
           records = myDb.Select(null, "sample_codes");//null means to fetch all data from the table
           // You could just add a statement 'using System.Data;' at the beginning and then use 'System.Data.DataTable' just as 'DataTable'
           Console.WriteLine("All records inserted:");
           if (records != null && records.Rows.Count > 0)
           {
              foreach (System.Data.DataRow row in records.Rows)
              {
                 Console.Write(row["id"] + " || "); //use double quotes
                 Console.Write(row["code"] + " || ");
                 Console.WriteLine(row["used"].ToString());
              }
           }
           else
              Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error selecting: " + e.Message);
        }

        //another select
```

```
        Console.WriteLine();
        Dictionary<string, string> search = new Dictionary<string, string>();
        search.Add("id", "1");
        search.Add("used", "0"); //this is not necessary, but just to indicate that u could have multiple search conditions

        try
        {
           records = myDb.Select(search, "sample_codes");
           Console.WriteLine("Record with id=1 and used=0 (should be only one record):");
           if (records != null && records.Rows.Count > 0)
           {
              foreach (System.Data.DataRow row in records.Rows)
              {
                 Console.Write(row["id"] + " || "); //use double quotes
                 Console.Write(row["code"] + " || ");
                 Console.WriteLine(row["used"].ToString());
              }
           }
           else
              Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error selecting: " + e.Message);
        }

        //update
        Dictionary<string, string> updateData = new Dictionary<string, string>();
        updateData.Add("code", "5555555555555555");
        updateData.Add("id", "2"); //this will be only used in the WHERE clause and not to update bcos I will instruct the function so via its parameter
        try
        {
           myDb.Update(updateData, "id", "sample_codes");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error updating: " + e.Message);
        }

        //delete
        try
        {
           myDb.Delete("id", "3", "sample_codes"); //deleting record with id=3
        }
        catch (Exception e)
        {
           Console.WriteLine("Error deleting: " + e.Message);
        }
        //selecting everything again to see changes made with uodate and delete
        Console.WriteLine();
        try
```

```
        {
            records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
            Console.WriteLine("All records after modifications:");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
                    Console.Write(row["code"] + " || ");
                    Console.WriteLine(row["used"].ToString());
                }
            }
            else
                Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error selecting: " + e.Message);
        }

        //Using the function 'Count'
        Console.WriteLine();
        Console.WriteLine("Number of all records left in table 'sample_codes': " +
myDb.Count("sample_codes"));
        Console.Read();
    }*/
```

MySQLDatabase.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using MySql.Data.MySqlClient;
namespace abc_osap
{
  class MySQLDatabase
  {
    private MySqlConnection connection;
    private string server = "localhost";
    private string database = "abc_osap";
    private string uid = "root";
    private string password = "";
    //Constructor
    //public MySQLDatabase(string uname, string pwd, string db)
    public MySQLDatabase()
    {
      Initialize();
      //Initialize(uname, pwd, db);
      //In case there any other thing one would like to have done on instantiation aside from
just initializing
    }
```

```
//Initialize values
//private void Initialize(string uname, string pwd, string db)
private void Initialize()
{
   //server = "localhost";
   //database = db;
   //uid = uname;
   //password = pwd;
   string connectionString;
   connectionString = "SERVER=" + server + ";" + "DATABASE=" +
               database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";

   connection = new MySqlConnection(connectionString);
}

//open connection to database
private bool OpenConnection()
{
   try
   {
      connection.Open();
      return true;
   }
   catch (MySqlException ex)
   {
      //The two most common error numbers when connecting are as follows:
      //0: Cannot connect to server.
      //1045: Invalid user name and/or password.
      switch (ex.Number)
      {
         case 0:
            throw new System.Exception("0 - Cannot connect to server");

         case 1045:
            throw new System.Exception("1045 - Invalid username/password");
      }
      return false;
   }
}

//Close connection
private bool CloseConnection()
{
   try
   {
      connection.Close();
      return true;
   }
   catch (MySqlException ex)
   {
      throw new System.Exception(ex.Message);
   }
```

```
    }

    //Insert statement
    public void Insert(Dictionary<string, string> ins, string table) //a dictionary data structure
is used to easily accomodate any number of columns in insert
    {
        int k=0;
        string query = "INSERT INTO "+table+" SET ";
        foreach (KeyValuePair<string, string> column in ins) {
            if (++k > 1) query += ", ";
            query += column.Key+" = '"+column.Value+"'";
        }
        //open connection
        if (this.OpenConnection() == true)
        {
            //create command and assign the query and connection from the constructor
            MySqlCommand cmd = new MySqlCommand(query, connection);
            //Execute command
            cmd.ExecuteNonQuery();
            //close connection
            this.CloseConnection();
        }
    }

    //Update statement
    public void Update(Dictionary<string, string> ins, string updateKey, string table)
    {
        int k = 0;
        string query = "UPDATE " + table + " SET ";
        foreach (KeyValuePair<string, string> column in ins)
        {
            if (column.Key != updateKey) {
                if (++k > 1) query += ", ";
                query += column.Key + " = '" + column.Value + "'";
            }
        }
        query += "WHERE "+updateKey+" = '"+ins[updateKey]+"'";

        //Open connection
        if (this.OpenConnection() == true)
        {
            //create mysql command
            MySqlCommand cmd = new MySqlCommand();
            //Assign the query using CommandText
            cmd.CommandText = query;
            //Assign the connection using Connection
            cmd.Connection = connection;

            //Execute query
            cmd.ExecuteNonQuery();

            //close connection
            this.CloseConnection();
```

```
        }
    }

    //Delete statement
    public void Delete(string key, string value, string table)
    {
        string query = "DELETE FROM " + table + " WHERE "+key+" = '"+value+"'";
        if (this.OpenConnection() == true)
        {
            MySqlCommand cmd = new MySqlCommand(query, connection);
            cmd.ExecuteNonQuery();
            this.CloseConnection();
        }
    }

    //Select statement
    public DataTable Select(Dictionary<string, string> search, string table, string connector =
"AND")
    {
        int k = 0;
        DataTable dbTable = new DataTable();

        string query = "SELECT * FROM " + table;
        if (search != null)
        {
            query += " WHERE ";
            foreach (KeyValuePair<string, string> column in search)
            {
                if (++k > 1) query += " " + connector + " ";
                query += column.Key + " = '" + column.Value + "'";
            }
        }

        //Open connection
        if (this.OpenConnection() == true)
        {
            //Create Command
            MySqlCommand cmd = new MySqlCommand(query, connection);
            //Create a data reader and Execute the command
            MySqlDataReader dataReader = cmd.ExecuteReader();
            if (dataReader.HasRows)
                dbTable.Load(dataReader);

            //close Data Reader
            dataReader.Close();

            //close Connection
            this.CloseConnection();

        }
        return dbTable;
    }
```

```
        public DataTable Query(string query)
        {
            DataTable dbTable = new DataTable();

            //Open connection
            if (this.OpenConnection() == true)
            {
                //Create Command
                MySqlCommand cmd = new MySqlCommand(query, connection);
                //Create a data reader and Execute the command
                MySqlDataReader dataReader = cmd.ExecuteReader();
                if (dataReader.HasRows)
                    dbTable.Load(dataReader);

                //close Data Reader
                dataReader.Close();

                //close Connection
                this.CloseConnection();

            }
            return dbTable;
        }

        //Count statement
        public int Count(string table)
        {
            string query = "SELECT Count(*) FROM "+table;
            int count = -1;

            //Open Connection
            if (this.OpenConnection() == true)
            {
                //Create Mysql Command
                MySqlCommand cmd = new MySqlCommand(query, connection);

                //ExecuteScalar will return one value
                count = int.Parse(cmd.ExecuteScalar() + "");

                //close Connection
                this.CloseConnection();
            }
            return count;
        }
    }
}
```

Office.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace abc_osap
{
   class Office
   {
      public int id { get; set; }
      public int capacity { get; set; }
      public string proximity { get; set; }
      public string properties { get; set; }
      public string type { get; set; }
      public int toilet { get; set; }
      public int resources { get; set; }

      public string Display()
      {
         Console.Write("Room: " + id + "; Properties: " + properties + "; TypeID: " + type + ";
Capacity:" + capacity);
         return "Room: " + id + "; Properties: " + properties + "; TypeID: " + type + ";
Capacity:" + capacity + "\r\n";
      }
      public void Copy(Office other)
      {
         this.capacity = other.capacity;
         this.properties = other.properties;
         this.type = other.type;
         this.id = other.id;
         this.proximity = other.proximity;
         this.toilet = other.toilet;
         this.resources = other.resources;
      }
   }
}
```

Program.cs

```
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace abc_osap
{
   class Program
   {
      static int np = 20; /* The number of colony size (employed bees+onlooker bees)*/
      static int foodNumber = np/2; //ie. number of solutions, equal to num of employed bees
      static int trialsLimit = 100;  /*A food source which could not be improved through
"trialsLimit" trials is abandoned by its employed bee*/
      static Solution[] foods = new Solution[foodNumber]; //foods - initial solution
      static int beeSearchLimit = 5;
      public static int[] tmp = new  int[100];
      public static int[] officeCapacity = new int[100];
```

```
        public static string[] officeWhoIsThere = new string[100];

        //upper bounds
        static int staff_ub, office_ub;

        public static Random r = new Random();
        static void Main(string[] args)
        {
            Stopwatch stpWatch = new Stopwatch();
            stpWatch.Start();

            MySQLDatabase myDb = new MySQLDatabase();

            //int maxCycle = 2500; /*The number of cycles for foraging {a stopping criteria}*/
            int maxCycle = 2; /*The number of cycles for foraging {a stopping criteria}*/
            //int runtime = 30;  /*Algorithm can be run many times in order to see its robustness*/
            int runtime = 1;  /*Algorithm can be run many times in order to see its robustness*/

            //use inputset class to handle input
            InputSet.Load(); //loading inputs

            //setting upper bounds
            staff_ub = InputSet.staffList.Count - 1;
            office_ub = InputSet.officesList.Count - 1;

            Solution fittest = new Solution (InputSet.staffList.Count);

            //continue here after writing other functions
            //double mean=0;

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"C:\Users\Public\abc_results.txt"))
            {
                for (int run = 0; run < runtime; run++)
                {
                    stpWatch.Reset();
                    stpWatch.Start();
                    Console.WriteLine("Run: " + (run + 1));
                    file.WriteLine("Run: " + (run + 1));
                    InputSet.loadOfficeCapacity();
                    Init();
                    //fittest = GetFittest();
                    for (int iter = 0; iter < maxCycle; iter++)
                    {
                        Console.WriteLine("iteration: " + (iter + 1));
                        file.WriteLine("iteration: " + (iter + 1));
                        //fittest.Print();
                        SendEmployedBees();
                        System.Threading.Thread.Sleep(20000);

                        CalcProbabilities();
                        SendOnlookerBees();
```

```
                fittest = GetFittest();
                SendScoutBees();
            }
            Console.WriteLine((run + 1).ToString() + ":");
            file.WriteLine((run + 1).ToString() + ":");
            Console.WriteLine("Best Allocation:");
            file.WriteLine("Best Allocation:");
            Array.Clear(tmp, 0, tmp.Length);
            Dictionary<string, string> solsToInsert = new Dictionary<string, string>();

            string fit_sol = fittest.Print();
            file.WriteLine(fit_sol);
            /*//System.out.println("%d. run: %e \n",run+1,GlobalMin);
            System.out.println((run+1)+".run:"+bee.GlobalMin);
            bee.GlobalMins[run]=bee.GlobalMin;
            mean=mean+bee.GlobalMin;*/
            stpWatch.Stop();
            TimeSpan ts = stpWatch.Elapsed;
            String elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours,
ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
            Console.WriteLine("=========");
            file.WriteLine("=========");
            Console.WriteLine("RunTime: " + elapsedTime);
            file.WriteLine("RunTime: " + elapsedTime);

            long memory = GC.GetTotalMemory(true);
            double mem = memory / 1024;
            mem = Math.Round(mem, 2);
            Console.WriteLine("Memory Used: {0}KB", mem.ToString());
            file.WriteLine("Memory Used: {0}KB", mem.ToString());
            Console.WriteLine();
            file.WriteLine(" ");

            foreach (Allocation solAlloc in fittest.allocs)
            {
                solsToInsert.Add("run", run.ToString());
                solsToInsert.Add("staff_id", solAlloc.staffID.ToString());
                solsToInsert.Add("office_id", solAlloc.officeID.ToString());
                solsToInsert.Add("penalty", fittest.penalty.ToString());
                solsToInsert.Add("alorithm_time", elapsedTime);
                solsToInsert.Add("memory_used", mem.ToString() + "KB");
                myDb.Insert(solsToInsert, "solutions");
                solsToInsert.Clear();
            }
        }
    }

    Console.Read();
}

static void Init()
{
    for (int i = 0; i < foodNumber; i++)
```

```
        {
            foods[i] = new Solution(InputSet.staffList.Count);
            //Console.WriteLine("Staff Count: "+InputSet.staffList.Count);
            for (int k = 0; k < InputSet.staffList.Count; k++)
            {
                int randOffice = 0;//Convert.ToInt16(r.NextDouble() *
(InputSet.officesList.Count() - 1));

InputSet.DeclareArray(InputSet.officeGroups[InputSet.staffList[k].typeId].Count);
                randOffice = InputSet.getOfficeByTypeName(InputSet.staffList[k].typeId,
InputSet.staffList[k].dept);
                tmp[k] = randOffice;
                officeWhoIsThere[randOffice] = InputSet.staffList[k].dept;
                officeCapacity[randOffice] -= 1;
                //Console.WriteLine(InputSet.staffList[k].id + "OF" + InputSet.staffList.Count +
"**" + InputSet.staffList[k].cadre + "(" + InputSet.staffList[k].id + ")==" + randOffice + "(" +
InputSet.GetOfficeById(randOffice).properties + ")");
                foods[i].allocs[k] = new Allocation(InputSet.staffList[k].id, randOffice);
            }
            foods[i].calcPenalty();
            Array.Clear(tmp, 0, tmp.Length);
            InputSet.loadOfficeCapacity();
        }
        //Console.Read();
    }

    static void ReInit(int i)
    {
        //foods[i] = new Solution(InputSet.staffList.Count);
        for (int k = 0; k < InputSet.staffList.Count; k++)
        {
            int randOffice = Convert.ToInt16(0 + r.NextDouble() *
(InputSet.officeGroups[InputSet.staffList[k].typeId].Count()-1));
            foods[i].allocs[k] = new Allocation(InputSet.staffList[k].id,
InputSet.officesList[randOffice].id);
        }
        foods[i].calcPenalty();
    }

    static Solution GetFittest() {
        Solution fittest = foods[0];
        for (int i = 1; i < foods.Length; i++) {
            if (fittest.penalty > foods[i].penalty) fittest = foods[i];
        }
        return fittest;
    }

    static double SumFitness()
    {
        double fitSum = 0;
        for (int i = 1; i < foods.Length; i++)
        {
            fitSum += foods[i].getFitness();
```

```
        }
        return fitSum;
    }

        static void SendEmployedBees()
        {
        for (int i=0; i<foodNumber; i++)
        {
            for (int q = 0; q < beeSearchLimit; q++) {
                int d = InputSet.staffList.Count - 1;
                Solution newSol = new Solution(InputSet.staffList.Count);
                newSol.Copy(foods[i]);

                /*The parameter to be changed is determined randomly*/
                int param2change = Convert.ToInt16(r.NextDouble() * d);

                /*A randomly chosen neighbour solution*/
                int neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber - 1));

                /*Randomly selected solution must be different from the solution i*/
                while (neighbour == i)
                    neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber - 1));

                newSol.allocs[param2change].officeID =
foods[neighbour].allocs[param2change].officeID;
                newSol.calcPenalty();

                if (newSol.penalty < foods[i].penalty)
                    foods[i].Copy(newSol);
                else
                    foods[i].trials++; /*if the solution i can not be improved, increase its trial
counter*/
            }

        }
    }

    static void CalcProbabilities()
    {
        for (int i = 1; i < foods.Length; i++)
        {
            foods[i].probability = foods[i].getFitness() / SumFitness();
        }
    }

    static void SendOnlookerBees()
    {
        int i = 0, t = 0;

        while (t < foodNumber)
        {
            if (r.NextDouble() < foods[i].probability)
            {
```

```
                t++;
                int d = InputSet.staffList.Count - 1;
                Solution newSol = new Solution(InputSet.staffList.Count);
                newSol.Copy(foods[i]);

                /*The parameter to be changed is determined randomly*/
                int param2change = Convert.ToInt16(r.NextDouble() * d);

                /*A randomly chosen neighbour solution*/
                int neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber-1));

                /*Randomly selected solution must be different from the solution i*/
                while (neighbour == i)
                   neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber-1));

                newSol.allocs[param2change].officeID =
foods[neighbour].allocs[param2change].officeID;
                newSol.calcPenalty();

                if (newSol.penalty < foods[i].penalty)
                   foods[i].Copy(newSol);
                else
                   foods[i].trials++; /*if the solution i can not be improved, increase its trial
counter*/
             }
             i++;
             if (i == foodNumber)
                i = 0;
          }
       }

       /*determine the food sources whose trial counter exceeds the "trialsLimit" value. In Basic
ABC, only one scout is allowed to occur in each cycle*/
       static void SendScoutBees()
       {
          int maxTrialIndex, i;
          maxTrialIndex = 0;
          for (i = 1; i < foodNumber; i++)
          {
             if (foods[i].trials > foods[maxTrialIndex].trials)
                maxTrialIndex = i;
          }
          if (foods[maxTrialIndex].trials >= trialsLimit)
          {
             ReInit(maxTrialIndex);
          }
       }
    }
}
```

Solution.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace abc_osap
{
   class Solution // AKA Food Source
   {
      public double penalty { get; set; } //inverse is the fitness; the nectar amount
      public double probability;
      public int trials;
      public Allocation[] allocs;

      public Solution(int numSol)
      {
         //numSol is the number of staff in the inputset
         allocs = new Allocation[numSol];
      }

      public Solution(Solution clone)
      {
         allocs = new Allocation[clone.allocs.Length];
         for (int i = 0; i < clone.allocs.Length; i++)
         {
            allocs[i] = new Allocation(clone.allocs[i].staffID, clone.allocs[i].officeID);
         }
         penalty = clone.penalty;
         probability = clone.probability;
      }

      public string Print()
      {
         string to_ret = "";
         for (int i = 0; i < allocs.Length; i++)
         {
            to_ret += allocs[i].Print();
         }
         Console.WriteLine("Penalty: " + penalty);
         to_ret += "Penalty: " + penalty;
         return to_ret;
      }

      public void Copy(Solution clone)
      {
         allocs = new Allocation[clone.allocs.Length];
         for (int i = 0; i < clone.allocs.Length; i++)
         {
            //Console.WriteLine(InputSet.staffList[clone.allocs[i].staffID].cadre + "==" +
InputSet.officesList[clone.allocs[i].officeID].type);
            allocs[i] = new Allocation(clone.allocs[i].staffID, clone.allocs[i].officeID);
         }
         penalty = clone.penalty;
```

```
            probability = clone.probability;
        }

        public double getFitness()
        {
            return 1 / penalty;
        }
        public void calcPenalty()
        {
            MySQLDatabase myDb = new MySQLDatabase();
            //selecting data from table
            DataTable records;
            penalty = 0;

            for (int i = 0; i < allocs.Length; i++)
            {
                try
                {
                    //selecting office records
                    records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'staff' AND entity_id =
"'+Convert.ToString(allocs[i].staffID)+"'");
                    if (records != null && records.Rows.Count > 0)
                    {
                        foreach (DataRow row in records.Rows)
                            penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
                            //penalty += Convert.ToDouble(row["weight"]);
                    }

                    records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'office' AND entity_id = "' +
Convert.ToString(allocs[i].officeID) + "'");
                    if (records != null && records.Rows.Count > 0)
                    {
                        foreach (DataRow row in records.Rows)
                            //penalty += Convert.ToDouble(row["weight"]);
                            penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("Error selecting: " + e.Message);
                }
            }
            //Console.WriteLine("Penalty: " + penalty);
        }

        int Violates(string constraintType, Allocation presentAlloc, DataRow constraintRow)
        {
            switch (constraintType)
            {
```

```
            case "allocation":
                if (presentAlloc.officeID !=
Convert.ToInt16(constraintRow["concerned_entity_id"]))
                    return Convert.ToInt16(constraintRow["weight"]);
                break;

            case "non-allocation":
                if (presentAlloc.officeID ==
Convert.ToInt16(constraintRow["concerned_entity_id"]))
                    return Convert.ToInt16(constraintRow["weight"]);
                break;

            case "same-room":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID != presentAlloc.officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "not-same-room":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID == presentAlloc.officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "not-sharing":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (presentAlloc.officeID == allocs[i].officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "capacity": //remember to put a counter such that for each solution, if the
constraint is capacity, it can only be called once
                int occupied = 0;
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (presentAlloc.officeID == allocs[i].officeID)
                        occupied++;
                }
                if (occupied > InputSet.GetOfficeById(presentAlloc.officeID).capacity)
                    return Convert.ToInt16(constraintRow["weight"]);
                break;

            case "nearby": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be defined only in one way
                for (int i = 0; i < allocs.Length; i++)
```

```
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID != allocs[i].officeID &&
myOffice.proximity.IndexOf("," + yourOffice.id + ",") < 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") < 0)
                        { //this means the constraint has been violated bcos the offices are not nearby
as expected
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
                break;

            case "away-from": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be define only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID == allocs[i].officeID ||
myOffice.proximity.IndexOf("," + yourOffice.id + ",") >= 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") >= 0)
                        { //this means the constraint has been violated bcos the offices are nearby
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are not in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
                break;

            default:
                return 0;
        }
        return 0;
    }
  }
}
```

Staff.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace abc_osap
{
   class Staff
   {
      public string[] cadarA = new string[3];
      public string[] cadarB = new string[3];
      public string[] cadarC = new string[5];
      public int id { get; set; }
      public string staffName { get; set; }
      public string cadre { get; set; }
      public string typeId { get; set; }
      public string dept { get; set; }

      public string Display()
      {
         Console.Write("Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " +
typeId);
         return "Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " + typeId +
"\r\n";
      }
      public void Copy(Staff other)
      {
         this.staffName = other.staffName;
         this.cadre = other.cadre;
         this.typeId = other.typeId;
      }
      public void initCadre()
      {
         //Type A
         this.cadarA[0] = "HOD";
         this.cadarA[1] = "PROF";

         //Type B
         this.cadarB[0] = "SL";
         this.cadarB[1] = "READER";

         //Type C
         this.cadarC[0] = "AL";
         this.cadarC[1] = "L1";
         this.cadarC[2] = "L2";
         this.cadarC[3] = "GA";
      }
   }
}
```

TABU SOURCE CODE ICS

Allocation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_osap
{
   class Allocation
   {
      public int staffID { get; set; }
      public int officeID { get; set; }

      public Allocation (int staff, int office)
      {
         staffID = staff;
         officeID = office;
      }

      public string Print()
      {
         string to_ret = "";
         string staff_rec = InputSet.GetStaffById(staffID).Display();
         to_ret += staff_rec + "\n";
         Console.Write(" ---- Allocated to: ");
         to_ret += " ---- Allocated to: " + "\n";
         string office_rec = InputSet.GetOfficeById(officeID).Display();
         to_ret += office_rec + "\r\n";
         Console.Write("\n");
         return to_ret;
      }
   }
}
```

InputSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace tabu_osap
{
   class InputSet
   {
      static Office[] offices;
      public static List<Office> officesList = new List<Office>();
      public static Dictionary<string, List<Office>> officeGroups = new Dictionary<string,
List<Office>>();
```

```
        static int[] officeGrp;
        static Staff[] staff;
        public static List<Staff> staffList = new List<Staff>();
        public static Dictionary<string, List<Staff>> staffGroups = new Dictionary<string,
List<Staff>>();

        public static void DeclareArray(int numOff)
        {
            //numSol is the number of office in the inputset
            officeGrp = new int[numOff];
        }

        public static void Load()
        {
            int i;

            MySQLDatabase myDb = new MySQLDatabase();
            //selecting data from table
            DataTable records;
            try
            {
                //selecting office records
                //Console.WriteLine("1");
                records = myDb.Select(null, "offices");//null means to fetch all data from the table
                //Console.WriteLine("2");
                if (records != null && records.Rows.Count > 0)
                {
                    offices = new Office[records.Rows.Count];
                    i = 0;
                    foreach (DataRow row in records.Rows)
                    {

                        //Console.WriteLine(row["id"].GetType());
                        //string d = row["type"].ToString();
                        offices[i] = new Office();

                        offices[i].id = (int)row["id"];
                        offices[i].type = row["type"].ToString();
                        offices[i].properties = row["properties"].ToString();
                        offices[i].capacity = (int)row["capacity"];
                        offices[i].toilet = row["toilet"].ToString() == "" ? 0 : (int)row["toilet"];
                        offices[i].resources = (int)row["resources"];
                        offices[i].proximity = row["proximity"].ToString();

                        officesList.Add(offices[i]);
                        if (!officeGroups.ContainsKey(offices[i].type))
                            officeGroups[offices[i].type] = new List<Office>();
                        officeGroups[offices[i].type].Add(offices[i]);
                    }
                }
                else
                    Console.WriteLine("No office record found!");
```

```
        //System.Threading.Thread.Sleep(1000000);

        //selecting staff records
        //Console.WriteLine("3");
        records = myDb.Select(null, "staff");//null means to fetch all data from the table
        //Console.WriteLine("4");
        if (records != null && records.Rows.Count > 0)
        {
            staff = new Staff[records.Rows.Count];
            i = 0;
            foreach (DataRow row in records.Rows)
            {
                staff[i] = new Staff();
                staff[i].id = Convert.ToInt16(row["id"]);
                staff[i].typeId = Convert.ToString(row["type_id"]);
                staff[i].cadre = Convert.ToString(row["cadre"]);
                staff[i].staffName = Convert.ToString(row["staff_name"]);
                staff[i].dept = Convert.ToString(row["dept"]);

                staffList.Add(staff[i]);

                //this part might not be needed
                if (!staffGroups.ContainsKey(staff[i].typeId))
                    staffGroups[staff[i].typeId] = new List<Staff>();
                staffGroups[staff[i].typeId].Add(staff[i]);
                //this part might not be needed
            }
        }
        else
            Console.WriteLine("No staff record found!");

    }
    catch (Exception e)
    {
        Console.WriteLine("Error selecting: " + e.Message);
        System.Threading.Thread.Sleep(1000000);
    }
}

public static Office GetOfficeById(int id)
{
    foreach (Office office in officesList)
    {
        if (office.id == id) return office;
    }
    return null;
}

public static void loadOfficeCapacity()
{
    foreach (Office office in officesList)
    {
```

```
            Program.officeCapacity[office.id] = office.capacity;
        }
    }

    public static int GetOfficeIndexById(int id)
    {
        for (int i = 0; i < officesList.Count(); i++)
        {
            if (officesList[i].id == id) return i;
        }
        return -1;
    }

    public static Staff GetStaffById(int id)
    {
        foreach (Staff staff in staffList)
        {
            if (staff.id == id) return staff;
        }
        return null;
    }

    public static int getOfficeByTypeName(string typey, string dept)
    {
    Found:
        int k = 0;
        foreach (Office office in officesList)
        {
            if (office.type == typey)
            {
                if (office.resources == 1)
                {
                    if (crossCheck(office.id) && checkCapacity(office.id) > 0 &&
checkWhoIsInTheOffice(office.id, dept) && isItProfSuit(typey, office.toilet))
                    {
                        officeGrp[k] = office.id;
                        k += 1;
                    }
                }
            }
        }
        //Console.WriteLine(ox + " T:" + typey);
        if (k == 0)
        {
            //Console.WriteLine(k);
            ClearTypeRoom(typey);
            goto Found;
        }
        //System.Threading.Thread.Sleep(600);
        int rnd = Program.r.Next(0, k - 1);
        return officeGrp[rnd];
    }
```

```
        static void ClearTypeRoom(string typey)
        {
            for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
            {
                Office xx = GetOfficeById(Program.tmp[numTmp]);
                if (xx != null)
                {
                    if (xx.type == typey)
                    {
                        Program.tmp[numTmp] = 0;
                    }
                }
            }
        }

        static bool crossCheck(int officeid)
        {
            for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
            {
                if (officeid == Program.tmp[numTmp])
                {
                    return false;
                }
            }
            return true;
        }

        static int checkCapacity(int officeid)
        {
            return Program.officeCapacity[officeid];
        }

        static bool checkWhoIsInTheOffice(int officeid, string dept)
        {
            if (Program.officeWhoIsThere[officeid] == dept || Program.officeWhoIsThere[officeid]
== null)
            {
                return true;
            }

            return false;
        }

        static bool isItProfSuit(string typed, int toilet)
        {
            if (typed == "A" && toilet == 0)
            {
                return false;
            }
            return true;
        }

        static int[] ShuffleArray(int[] array)
```

```
        {
            Random r = new Random();
            for (int i = array.Length; i > 0; i--)
            {
                int j = r.Next(i);
                int k = array[j];
                array[j] = array[i - 1];
                array[i - 1] = k;
            }
            return array;
        }

    }
}



/*


            //THIS PART IS TO TEST THE DATABASE CLASS
            string db = "vp_validation_system";
            string dbuid = "root";
            string dbpwd = "";

            //instantiating the db class
            MySQLDatabase myDb = new MySQLDatabase(dbuid, dbpwd, db);

            //selecting data from table - there shouldn't be any records for now
            System.Data.DataTable records;
            try
            {
                records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
                // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
                Console.WriteLine("All records inserted:");
                if (records != null && records.Rows.Count > 0)
                {
                    foreach (System.Data.DataRow row in records.Rows)
                    {
                        Console.Write(row["id"] + " || "); //use double quotes
                        Console.Write(row["code"] + " || ");
                        Console.WriteLine(row["used"].ToString());
                    }
                }
                else
                    Console.WriteLine("No record found!");
            }
            catch (Exception e)
            {
                Console.WriteLine("Error selecting: " + e.Message);
            }
```

```
        //insert record to table 'sample_codes'
        Dictionary<string, string>[] insertData = new Dictionary<string, string>[3]; //An array
of dictionaries, each dictionary represents a row to be inserted
        insertData[0] = new Dictionary<string, string>();
        insertData[0].Add("code", "563443y3uh87grr"); //column name, value in table
'sample_codes'
        insertData[0].Add("used", "0"); //another column name, value in table 'sample_codes'

        insertData[1] = new Dictionary<string, string>();
        insertData[1].Add("code", "fhbeiurhg34u23434");
        insertData[1].Add("used", "0");

        insertData[2] = new Dictionary<string, string>();
        insertData[2].Add("code", "fdhrbru3u9rwei9jcks");
        insertData[2].Add("used", "0");

        try
        {
           myDb.Insert(insertData[0], "sample_codes");
           myDb.Insert(insertData[1], "sample_codes");
           myDb.Insert(insertData[2], "sample_codes");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error inserting: " + e.Message);
        }

        //selecting data from table
        Console.WriteLine();
        try
        {
           records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
           // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
           Console.WriteLine("All records inserted:");
           if (records != null && records.Rows.Count > 0)
           {
              foreach (System.Data.DataRow row in records.Rows)
              {
                 Console.Write(row["id"] + " || "); //use double quotes
                 Console.Write(row["code"] + " || ");
                 Console.WriteLine(row["used"].ToString());
              }
           }
           else
              Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error selecting: " + e.Message);
        }
```

```
        //another select
        Console.WriteLine();
        Dictionary<string, string> search = new Dictionary<string, string>();
        search.Add("id", "1");
        search.Add("used", "0"); //this is not necessary, but just to indicate that u could have
multiple search conditions

        try
        {
           records = myDb.Select(search, "sample_codes");
           Console.WriteLine("Record with id=1 and used=0 (should be only one record):");
           if (records != null && records.Rows.Count > 0)
           {
              foreach (System.Data.DataRow row in records.Rows)
              {
                 Console.Write(row["id"] + " || "); //use double quotes
                 Console.Write(row["code"] + " || ");
                 Console.WriteLine(row["used"].ToString());
              }
           }
           else
              Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error selecting: " + e.Message);
        }

        //update
        Dictionary<string, string> updateData = new Dictionary<string, string>();
        updateData.Add("code", "5555555555555555");
        updateData.Add("id", "2"); //this will be only used in the WHERE clause and not to
update bcos I will instruct the function so via its parameter
        try
        {
           myDb.Update(updateData, "id", "sample_codes");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error updating: " + e.Message);
        }

        //delete
        try
        {
           myDb.Delete("id", "3", "sample_codes"); //deleting record with id=3
        }
        catch (Exception e)
        {
           Console.WriteLine("Error deleting: " + e.Message);
        }
```

```
        //selecting everything again to see changes made with uodate and delete
        Console.WriteLine();
        try
        {
            records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
            Console.WriteLine("All records after modifications:");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
                    Console.Write(row["code"] + " || ");
                    Console.WriteLine(row["used"].ToString());
                }
            }
            else
                Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error selecting: " + e.Message);
        }

        //Using the function 'Count'
        Console.WriteLine();
        Console.WriteLine("Number of all records left in table 'sample_codes': " +
myDb.Count("sample_codes"));
        Console.Read();
    }*/
```

MySQLDatabase.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Data;
using MySql.Data.MySqlClient;

namespace tabu_osap
{
    class MySQLDatabase
    {
        private MySqlConnection connection;
        private string server = "localhost";
        private string database = "abc_osap";
        private string uid = "root";
        private string password = "";

        //Constructor
```

```
        //public MySQLDatabase(string uname, string pwd, string db)
        public MySQLDatabase()
        {
            Initialize();
            //Initialize(uname, pwd, db);
            //In case there any other thing one would like to have done on instantiation aside from
    just initializing
        }

        //Initialize values
        //private void Initialize(string uname, string pwd, string db)
        private void Initialize()
        {
            //server = "localhost";
            //database = db;
            //uid = uname;
            //password = pwd;
            string connectionString;
            connectionString = "SERVER=" + server + ";" + "DATABASE=" +
            database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";

            connection = new MySqlConnection(connectionString);
        }

        //open connection to database
        private bool OpenConnection()
        {
            try
            {
                connection.Open();
                return true;
            }
            catch (MySqlException ex)
            {
                //The two most common error numbers when connecting are as follows:
                //0: Cannot connect to server.
                //1045: Invalid user name and/or password.
                switch (ex.Number)
                {
                    case 0:
                        throw new System.Exception("0 - Cannot connect to server");

                    case 1045:
                        throw new System.Exception("1045 - Invalid username/password");
                }
                return false;
            }
        }

        //Close connection
        private bool CloseConnection()
        {
            try
```

```
            {
                connection.Close();
                return true;
            }
            catch (MySqlException ex)
            {
                throw new System.Exception(ex.Message);
            }
        }

        //Insert statement
        public void Insert(Dictionary<string, string> ins, string table) //a dictionary data structure
is used to easily accomodate any number of columns in insert
        {
            int k = 0;
            string query = "INSERT INTO " + table + " SET ";
            foreach (KeyValuePair<string, string> column in ins)
            {
                if (++k > 1) query += ", ";
                query += column.Key + " = '" + column.Value + "'";
            }

            //open connection
            if (this.OpenConnection() == true)
            {
                //create command and assign the query and connection from the constructor
                MySqlCommand cmd = new MySqlCommand(query, connection);

                //Execute command
                cmd.ExecuteNonQuery();

                //close connection
                this.CloseConnection();
            }
        }

        //Update statement
        public void Update(Dictionary<string, string> ins, string updateKey, string table)
        {
            int k = 0;
            string query = "UPDATE " + table + " SET ";
            foreach (KeyValuePair<string, string> column in ins)
            {
                if (column.Key != updateKey)
                {
                    if (++k > 1) query += ", ";
                    query += column.Key + " = '" + column.Value + "'";
                }
            }
            query += "WHERE " + updateKey + " = '" + ins[updateKey] + "'";

            //Open connection
            if (this.OpenConnection() == true)
```

```
        {
            //create mysql command
            MySqlCommand cmd = new MySqlCommand();
            //Assign the query using CommandText
            cmd.CommandText = query;
            //Assign the connection using Connection
            cmd.Connection = connection;

            //Execute query
            cmd.ExecuteNonQuery();

            //close connection
            this.CloseConnection();
        }
    }

    //Delete statement
    public void Delete(string key, string value, string table)
    {
        string query = "DELETE FROM " + table + " WHERE " + key + " = '" + value + "'";
        if (this.OpenConnection() == true)
        {
            MySqlCommand cmd = new MySqlCommand(query, connection);
            cmd.ExecuteNonQuery();
            this.CloseConnection();
        }
    }

    //Select statement
    public DataTable Select(Dictionary<string, string> search, string table, string connector =
"AND")
    {
        int k = 0;
        DataTable dbTable = new DataTable();

        string query = "SELECT * FROM " + table;
        if (search != null)
        {
            query += " WHERE ";
            foreach (KeyValuePair<string, string> column in search)
            {
                if (++k > 1) query += " " + connector + " ";
                query += column.Key + " = '" + column.Value + "'";
            }
        }

        //Open connection
        if (this.OpenConnection() == true)
        {
            //Create Command
            MySqlCommand cmd = new MySqlCommand(query, connection);
            //Create a data reader and Execute the command
            MySqlDataReader dataReader = cmd.ExecuteReader();
```

```
        if (dataReader.HasRows)
            dbTable.Load(dataReader);

        //close Data Reader
        dataReader.Close();

        //close Connection
        this.CloseConnection();

    }
    return dbTable;
}

public DataTable Query(string query)
{
    DataTable dbTable = new DataTable();

    //Open connection
    if (this.OpenConnection() == true)
    {
        //Create Command
        MySqlCommand cmd = new MySqlCommand(query, connection);
        //Create a data reader and Execute the command
        MySqlDataReader dataReader = cmd.ExecuteReader();
        if (dataReader.HasRows)
            dbTable.Load(dataReader);

        //close Data Reader
        dataReader.Close();

        //close Connection
        this.CloseConnection();

    }
    return dbTable;
}

//Count statement
public int Count(string table)
{
    string query = "SELECT Count(*) FROM " + table;
    int count = -1;

    //Open Connection
    if (this.OpenConnection() == true)
    {
        //Create Mysql Command
        MySqlCommand cmd = new MySqlCommand(query, connection);

        //ExecuteScalar will return one value
        count = int.Parse(cmd.ExecuteScalar() + "");

        //close Connection
```

```
            this.CloseConnection();
        }
        return count;
    }
  }
}
```

Office.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_osap
{
  class Office
  {
    public int id { get; set; }
    public int capacity { get; set; }
    public string proximity;
    public string properties { get; set; }
    public string type { get; set; }
    public int toilet { get; set; }
    public int resources { get; set; }

    public string Display()
    {
      Console.Write("Capacity: " + capacity + "Room: " + id + "; Properties: " + properties +
"; TypeID: " + type + "; Capacity:" + capacity);
      return "Room: " + id + "; Properties: " + properties + "; TypeID: " + type + ";
Capacity:" + capacity + "\r\n";
    }
    public void Copy(Office other)
    {
      this.capacity = other.capacity;
      this.properties = other.properties;
      this.type = other.type;
      this.id = other.id;
      this.proximity = other.proximity;
      this.toilet = other.toilet;
      this.resources = other.resources;
    }
  }
}
```

Program.cs

```
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace tabu_osap
{
   class Program
   {
      //upper bounds
      static int staff_ub, office_ub;
      public static int[] tmp = new int[100];
      public static int[] officeCapacity = new int[100];
      public static string[] officeWhoIsThere = new string[100];

      public static Random r = new Random();
      static void Main(string[] args)
      {
         Stopwatch stpWatch = new Stopwatch();
         stpWatch.Start();

         MySQLDatabase myDb = new MySQLDatabase();

         int maxCycle = 2;
         int no_of_runs = 1; /*The number of cycles for foraging {a stopping criteria}*/

         //use inputset class to handle input
         InputSet.Load(); //loading inputs

         //setting upper bounds
         staff_ub = InputSet.staffList.Count - 1;
         office_ub = InputSet.officesList.Count - 1;

         List<Solution> candidateSols = new List<Solution>();
         List<Solution> tabuSolutions = new List<Solution>();

         using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"C:\Users\Public\tabu_results.txt"))
         {
            for (int iter = 0; iter < no_of_runs; iter++)
            {
               stpWatch.Reset();
               stpWatch.Start();
               Console.WriteLine("Run: " + (iter + 1));
               file.WriteLine("Run: " + (iter + 1));

               InputSet.loadOfficeCapacity();
               Solution initialSol = GetInitialSolution();
               Solution fittest = new Solution(initialSol);

               for (int i = 0; i < maxCycle; i++)
               {
                  Console.WriteLine("iteration: " + (i + 1));
                  file.WriteLine("iteration: " + (i + 1));
                  for (int k = 0; k < initialSol.allocs.Count(); k++)
                  {
                     Solution initialSolCopy = new Solution(initialSol);
```

```
                    //for each then pick an office to randomly change

                    int randOffice = 0;//Convert.ToInt16(r.NextDouble() * office_ub);

InputSet.DeclareArray(InputSet.officeGroups[InputSet.staffList[k].typeId].Count);
                    randOffice = InputSet.getOfficeByTypeName(InputSet.staffList[k].typeId,
InputSet.staffList[k].dept);
                    tmp[k] = randOffice;
                    officeWhoIsThere[randOffice] = InputSet.staffList[k].dept;
                    officeCapacity[randOffice] -= 1;

                    initialSolCopy.allocs[k].officeID = randOffice;
                    initialSolCopy.calcPenalty();
                    if (!tabuSolutions.Contains(initialSolCopy) && fittest.penalty >=
initialSolCopy.penalty)
                    {
                       fittest.Copy(initialSolCopy);
                       tabuSolutions.Add(initialSolCopy);
                    }
                    else if (!tabuSolutions.Contains(initialSolCopy))
                    {
                       tabuSolutions.Add(initialSolCopy);
                    }
                 }
                 Array.Clear(tmp, 0, tmp.Length);
                 InputSet.loadOfficeCapacity();
                 candidateSols.Add(fittest);
              }

              Console.WriteLine("Best Allocation:");
              file.WriteLine("Best Allocation:");

              Dictionary<string, string> solsToInsert = new Dictionary<string, string>();

              string fit_sol = fittest.Print();
              file.WriteLine(fit_sol);

              stpWatch.Stop();
              TimeSpan ts = stpWatch.Elapsed;
              String elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours,
ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
              Console.WriteLine("=========");
              file.WriteLine("=========");
              Console.WriteLine("RunTime: " + elapsedTime);
              file.WriteLine("RunTime: " + elapsedTime);

              long memory = GC.GetTotalMemory(true);
              double mem = memory / 1024;
              mem = Math.Round(mem, 2);
              Console.WriteLine("Memory Used: {0}KB", mem.ToString());
              file.WriteLine("Memory Used: {0}KB", mem.ToString());
              Console.WriteLine();
              file.WriteLine(" ");
```

```
                foreach (Allocation solAlloc in fittest.allocs)
                {
                    solsToInsert.Add("run", iter.ToString());
                    solsToInsert.Add("staff_id", solAlloc.staffID.ToString());
                    solsToInsert.Add("office_id", solAlloc.officeID.ToString());
                    solsToInsert.Add("penalty", fittest.penalty.ToString());
                    solsToInsert.Add("alorithm_time", elapsedTime);
                    solsToInsert.Add("memory_used", mem.ToString() + "KB");
                    myDb.Insert(solsToInsert, "solutions");
                    solsToInsert.Clear();
                }
            }
        }

        Console.Read();
    }

    static Solution GetInitialSolution()
    {
        Solution sol = new Solution(InputSet.staffList.Count);
        for (int k = 0; k < InputSet.staffList.Count; k++)
        {
            int randOffice = 0;//Convert.ToInt16(r.NextDouble() * (InputSet.officesList.Count()
- 1));
            InputSet.DeclareArray(InputSet.officeGroups[InputSet.staffList[k].typeId].Count);
            randOffice = InputSet.getOfficeByTypeName(InputSet.staffList[k].typeId,
InputSet.staffList[k].dept);
            tmp[k] = randOffice;
            officeWhoIsThere[randOffice] = InputSet.staffList[k].dept;
            officeCapacity[randOffice] -= 1;
            //Console.WriteLine(InputSet.staffList[k].cadre + "(" + InputSet.staffList[k].id +
")==" + randOffice + "(" + InputSet.GetOfficeById(randOffice).properties + ")");
            sol.allocs[k] = new Allocation(InputSet.staffList[k].id, randOffice);

        }
        sol.calcPenalty();
        Array.Clear(tmp, 0, tmp.Length);
        InputSet.loadOfficeCapacity();
        return sol;
    }

    static Solution GetFittest(List<Solution> solutions) {
        Solution fittest = new Solution(solutions.ElementAt(0));
        for (int i = 1; i < solutions.Count; i++)
        {
            if (fittest.penalty > solutions.ElementAt(i).penalty)
fittest.Copy(solutions.ElementAt(i));
        }
        return fittest;
    }

}
```

```
}
```

Solution.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace tabu_osap
{
   class Solution // AKA Food Source
   {
      public double penalty { get; set; } //inverse is the fitness; the nectar amount
      public double probability;
      public int trials;
      public Allocation[] allocs;

      public Solution(int numSol)
      {
         //numSol is the number of staff in the inputset
         allocs = new Allocation[numSol];
      }

      public Solution(Solution clone)
      {
         allocs = new Allocation[clone.allocs.Length];
         for (int i = 0; i < clone.allocs.Length; i++)
         {
            allocs[i] = new Allocation(clone.allocs[i].staffID, clone.allocs[i].officeID);
         }
         penalty = clone.penalty;
         probability = clone.probability;
      }

      public string Print()
      {
         string to_ret = "";
         for (int i = 0; i < allocs.Length; i++)
         {
            to_ret += allocs[i].Print();
         }
         Console.WriteLine("Penalty: " + penalty);
         to_ret += "Penalty: " + penalty;
         return to_ret;
         //return penalty.ToString();
         //file.WriteLine(penalty);
      }

      public void Copy(Solution clone)
      {
         allocs = new Allocation[clone.allocs.Length];
```

```
            for (int i = 0; i < clone.allocs.Length; i++)
            {
                allocs[i] = new Allocation(clone.allocs[i].staffID, clone.allocs[i].officeID);
            }
            penalty = clone.penalty;
            probability = clone.probability;
        }

        public double getFitness()
        {
            return 1 / penalty;
        }
        public void calcPenalty()
        {
            MySQLDatabase myDb = new MySQLDatabase();
            //selecting data from table
            DataTable records;
            penalty = 0;

            for (int i = 0; i < allocs.Length; i++)
            {
                try
                {
                    //selecting office records
                    records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'staff' AND entity_id =
"'+Convert.ToString(allocs[i].staffID)+"'");
                    if (records != null && records.Rows.Count > 0)
                    {
                        foreach (DataRow row in records.Rows)
                            penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
                            //penalty += Convert.ToDouble(row["weight"]);
                    }

                    records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'office' AND entity_id = "' +
Convert.ToString(allocs[i].officeID) + "'");
                    if (records != null && records.Rows.Count > 0)
                    {
                        foreach (DataRow row in records.Rows)
                            //penalty += Convert.ToDouble(row["weight"]);
                            penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("Error selecting: " + e.Message);
                }
            }
        }
```

```
int Violates(string constraintType, Allocation presentAlloc, DataRow constraintRow)
{
   switch (constraintType)
   {
      case "allocation":
         if (presentAlloc.officeID !=
Convert.ToInt16(constraintRow["concerned_entity_id"]))
            return Convert.ToInt16(constraintRow["weight"]);
         break;

      case "non-allocation":
         if (presentAlloc.officeID ==
Convert.ToInt16(constraintRow["concerned_entity_id"]))
            return Convert.ToInt16(constraintRow["weight"]);
         break;

      case "same-room":
         for (int i = 0; i < allocs.Length; i++)
         {
            if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID != presentAlloc.officeID)
               return Convert.ToInt16(constraintRow["weight"]);
         }
         break;

      case "not-same-room":
         for (int i = 0; i < allocs.Length; i++)
         {
            if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID == presentAlloc.officeID)
               return Convert.ToInt16(constraintRow["weight"]);
         }
         break;

      case "not-sharing":
         for (int i = 0; i < allocs.Length; i++)
         {
            if (presentAlloc.officeID == allocs[i].officeID)
               return Convert.ToInt16(constraintRow["weight"]);
         }
         break;

      case "capacity": //remember to put a counter such that for each solution, if the
constraint is capacity, it can only be called once
         int occupied = 0;
         for (int i = 0; i < allocs.Length; i++)
         {
            if (presentAlloc.officeID == allocs[i].officeID)
               occupied++;
         }
         if (occupied > InputSet.GetOfficeById(presentAlloc.officeID).capacity)
            return Convert.ToInt16(constraintRow["weight"]);
         break;
```

```
            case "nearby": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be define only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID != allocs[i].officeID &&
myOffice.proximity.IndexOf("," + yourOffice.id + ",") < 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") < 0)
                        { //this means the constraint has been violated bcos the offices are not nearby
as expected
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
                break;

            case "away-from": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be define only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID == allocs[i].officeID ||
myOffice.proximity.IndexOf("," + yourOffice.id + ",") >= 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") >= 0)
                        { //this means the constraint has been violated bcos the offices are nearby
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are not in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
                break;

            default:
                return 0;
        }
        return 0;
    }
```

```
    }
}
```

Staff.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_osap
{
    class Staff
    {
        public int id { get; set; }
        public string staffName { get; set; }
        public string cadre { get; set; }
        public string typeId { get; set; }
        public string dept { get; set; }

        public string Display()
        {
            Console.Write("Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " +
typeId);
            return "Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " + typeId +
"\r\n";
        }
        public void Copy(Staff other)
        {
            this.staffName = other.staffName;
            this.cadre = other.cadre;
            this.typeId = other.typeId;
        }
    }
}
```

GENETIC SOURCE CODE ICS

Allocation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Genetic
{
    class Allocation
    {
        public int staffID { get; set; }
```

```
        public int officeID { get; set; }

        public Allocation (int staff, int office)
        {
            staffID = staff;
            officeID = office;
        }
        public string Print()
        {
            string to_ret = "";
            string staff_rec = InputSet.GetStaffById(staffID).Display();
            to_ret += staff_rec + "\n";
            Console.Write(" ---- Allocated to: ");
            to_ret += " ---- Allocated to: " + "\r\n";
            string office_rec = InputSet.GetOfficeById(officeID).Display();
            to_ret += office_rec + "\r\n";
            Console.Write("\n");
            return to_ret;
        }
    }
}
```

InputSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace Genetic
{
    class InputSet
    {
        static Office[] offices;
        public static List<Office> officesList = new List<Office>();
        public static Dictionary<string, List<Office>> officeGroups = new
Dictionary<string,List<Office>>();

        static Staff[] staff;
        static int[] officeGrp;
        public static List<Staff> staffList = new List<Staff>();
        public static Dictionary<string, List<Staff>> staffGroups = new Dictionary<string,
List<Staff>>();

        public static void DeclareArray(int numOff)
        {
            //numSol is the number of office in the inputset
            officeGrp = new int[numOff];
        }

        public static void Load()
        {
```

```
int i;

MySQLDatabase myDb = new MySQLDatabase();
//selecting data from table
DataTable records;
try
{
   //selecting office records
   //Console.WriteLine("1");
   records = myDb.Select(null, "offices");//null means to fetch all data from the table
   //Console.WriteLine("2");
   if (records != null && records.Rows.Count > 0)
   {
      offices = new Office[records.Rows.Count];
      i = 0;
      foreach (DataRow row in records.Rows)
      {

         //Console.WriteLine(row["id"].GetType());
         //string d = row["type"].ToString();
         offices[i] = new Office();

         offices[i].id = (int)row["id"];
         offices[i].type = row["type"].ToString();
         offices[i].properties = row["properties"].ToString();
         offices[i].capacity = (int)row["capacity"];
         offices[i].toilet = row["toilet"].ToString() == "" ? 0 : (int)row["toilet"];
         offices[i].resources = (int)row["resources"];
         offices[i].proximity = row["proximity"].ToString();

         officesList.Add(offices[i]);
         if (!officeGroups.ContainsKey(offices[i].type))
            officeGroups[offices[i].type] = new List<Office>();
         officeGroups[offices[i].type].Add(offices[i]);
      }
   }
   else
      Console.WriteLine("No office record found!");

   //System.Threading.Thread.Sleep(1000000);

   //selecting staff records
   //Console.WriteLine("3");
   records = myDb.Select(null, "staff");//null means to fetch all data from the table
   //Console.WriteLine("4");
   if (records != null && records.Rows.Count > 0)
   {
      staff = new Staff[records.Rows.Count];
      i = 0;
      foreach (DataRow row in records.Rows)
      {
         staff[i] = new Staff();
         staff[i].id = Convert.ToInt16(row["id"]);
```

```
                staff[i].typeId = Convert.ToString(row["type_id"]);
                staff[i].cadre = Convert.ToString(row["cadre"]);
                staff[i].staffName = Convert.ToString(row["staff_name"]);
                staff[i].dept = Convert.ToString(row["dept"]);

                staffList.Add(staff[i]);

                //this part might not be needed
                if (!staffGroups.ContainsKey(staff[i].typeId))
                    staffGroups[staff[i].typeId] = new List<Staff>();
                staffGroups[staff[i].typeId].Add(staff[i]);
                //this part might not be needed
            }
        }
        else
            Console.WriteLine("No staff record found!");

    }
    catch (Exception e)
    {
        Console.WriteLine("Error selecting: " + e.Message);
        System.Threading.Thread.Sleep(1000000);
    }
}

public static Office GetOfficeById(int id)
{
    foreach (Office office in officesList)
    {
        if (office.id == id) return office;
    }
    return null;
}

public static void loadOfficeCapacity()
{
    foreach (Office office in officesList)
    {
        Program.officeCapacity[office.id] = office.capacity;
    }
}

public static int GetOfficeIndexById(int id)
{
    for (int i = 0; i < officesList.Count(); i++)
    {
        if (officesList[i].id == id) return i;
    }
    return -1;
}

public static Staff GetStaffById(int id)
{
```

```
        foreach (Staff staff in staffList)
        {
            if (staff.id == id) return staff;
        }
        return null;
    }

    public static int getOfficeByTypeName(string typey, string dept)
    {
        Found:
        int k = 0;
        foreach (Office office in officesList)
        {
            if (office.type == typey)
            {
                if (office.resources == 1)
                {
                    if (crossCheck(office.id) && checkCapacity(office.id) > 0 &&
checkWhoIsInTheOffice(office.id, dept) && isItProfSuit(typey, office.toilet))
                    {
                        officeGrp[k] = office.id;
                        k += 1;
                    }
                }
            }
        }
        //Console.WriteLine(ox + " T:" + typey);
        if (k == 0)
        {
            //Console.WriteLine(k);
            ClearTypeRoom(typey);
            goto Found;
        }
        //System.Threading.Thread.Sleep(600);
        int rnd = Program.r.Next(0, k - 1);
        return officeGrp[rnd];
    }

    static void ClearTypeRoom(string typey)
    {
        for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
        {
            Office xx = GetOfficeById(Program.tmp[numTmp]);
            if (xx != null)
            {
                if (xx.type == typey)
                {
                    Program.tmp[numTmp] = 0;
                }
            }
        }
    }
```

```
        static bool crossCheck(int officeid)
        {
            for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
            {
                if (officeid == Program.tmp[numTmp])
                {
                    return false;
                }
            }
            return true;
        }

        static int checkCapacity(int officeid)
        {
            return Program.officeCapacity[officeid];
        }

        static bool checkWhoIsInTheOffice(int officeid, string dept)
        {
            if (Program.officeWhoIsThere[officeid] == dept || Program.officeWhoIsThere[officeid]
== null)
            {
                return true;
            }
            return false;
        }

        static bool isItProfSuit(string typed, int toilet)
        {
            if (typed == "A" && toilet == 0)
            {
                return false;
            }
            return true;
        }

        static int[] ShuffleArray(int[] array)
        {
            Random r = new Random();
            for (int i = array.Length; i > 0; i--)
            {
                int j = r.Next(i);
                int k = array[j];
                array[j] = array[i - 1];
                array[i - 1] = k;
            }
            return array;
        }
    }
}
```

```
/*

        //THIS PART IS TO TEST THE DATABASE CLASS
        string db = "vp_validation_system";
        string dbuid = "root";
        string dbpwd = "";

        //instantiating the db class
        MySQLDatabase myDb = new MySQLDatabase(dbuid, dbpwd, db);

        //selecting data from table - there shouldn't be any records for now
        System.Data.DataTable records;
        try
        {
           records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
           // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
           Console.WriteLine("All records inserted:");
           if (records != null && records.Rows.Count > 0)
           {
              foreach (System.Data.DataRow row in records.Rows)
              {
                 Console.Write(row["id"] + " || "); //use double quotes
                 Console.Write(row["code"] + " || ");
                 Console.WriteLine(row["used"].ToString());
              }
           }
           else
              Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error selecting: " + e.Message);
        }

        //insert record to table 'sample_codes'
        Dictionary<string, string>[] insertData = new Dictionary<string, string>[3]; //An array
of dictionaries, each dictionary represents a row to be inserted
        insertData[0] = new Dictionary<string, string>();
        insertData[0].Add("code", "563443y3uh87grr"); //column name, value in table
'sample_codes'
        insertData[0].Add("used", "0"); //another column name, value in table 'sample_codes'

        insertData[1] = new Dictionary<string, string>();
        insertData[1].Add("code", "fhbeiurhg34u23434");
        insertData[1].Add("used", "0");

        insertData[2] = new Dictionary<string, string>();
        insertData[2].Add("code", "fdhrbru3u9rwei9jcks");
        insertData[2].Add("used", "0");
```

```
        try
        {
            myDb.Insert(insertData[0], "sample_codes");
            myDb.Insert(insertData[1], "sample_codes");
            myDb.Insert(insertData[2], "sample_codes");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error inserting: " + e.Message);
        }

        //selecting data from table
        Console.WriteLine();
        try
        {
            records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
            // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
            Console.WriteLine("All records inserted:");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
                    Console.Write(row["code"] + " || ");
                    Console.WriteLine(row["used"].ToString());
                }
            }
            else
                Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error selecting: " + e.Message);
        }

        //another select
        Console.WriteLine();
        Dictionary<string, string> search = new Dictionary<string, string>();
        search.Add("id", "1");
        search.Add("used", "0"); //this is not necessary, but just to indicate that u could have
multiple search conditions

        try
        {
            records = myDb.Select(search, "sample_codes");
            Console.WriteLine("Record with id=1 and used=0 (should be only one record):");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
```

```
                Console.Write(row["code"] + " || ");
                Console.WriteLine(row["used"].ToString());
            }
        }
        else
            Console.WriteLine("No record found!");
    }
    catch (Exception e)
    {
        Console.WriteLine("Error selecting: " + e.Message);
    }

    //update
    Dictionary<string, string> updateData = new Dictionary<string, string>();
    updateData.Add("code", "5555555555555555");
    updateData.Add("id", "2"); //this will be only used in the WHERE clause and not to
update bcos I will instruct the function so via its parameter
    try
    {
        myDb.Update(updateData, "id", "sample_codes");
    }
    catch (Exception e)
    {
        Console.WriteLine("Error updating: " + e.Message);
    }

    //delete
    try
    {
        myDb.Delete("id", "3", "sample_codes"); //deleting record with id=3
    }
    catch (Exception e)
    {
        Console.WriteLine("Error deleting: " + e.Message);
    }


    //selecting everything again to see changes made with uodate and delete
    Console.WriteLine();
    try
    {
        records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
        Console.WriteLine("All records after modifications:");
        if (records != null && records.Rows.Count > 0)
        {
            foreach (System.Data.DataRow row in records.Rows)
            {
                Console.Write(row["id"] + " || "); //use double quotes
                Console.Write(row["code"] + " || ");
                Console.WriteLine(row["used"].ToString());
            }
        }
```

```
            else
                Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error selecting: " + e.Message);
        }

        //Using the function 'Count'
        Console.WriteLine();
        Console.WriteLine("Number of all records left in table 'sample_codes': " +
myDb.Count("sample_codes"));
        Console.Read();
    }*/
```

MySQLDatabase.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Data;
using MySql.Data.MySqlClient;

namespace Genetic
{
    class MySQLDatabase
    {
        private MySqlConnection connection;
        private string server = "localhost";
        private string database = "abc_osap";
        private string uid = "root";
        private string password = "";

        //Constructor
        //public MySQLDatabase(string uname, string pwd, string db)
        public MySQLDatabase()
        {
            Initialize();
            //Initialize(uname, pwd, db);
            //In case there any other thing one would like to have done on instantiation aside from
just initializing
        }

        //Initialize values
        //private void Initialize(string uname, string pwd, string db)
        private void Initialize()
        {
            //server = "localhost";
            //database = db;
            //uid = uname;
            //password = pwd;
```

```
            string connectionString;
            connectionString = "SERVER=" + server + ";" + "DATABASE=" +
                        database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";

            connection = new MySqlConnection(connectionString);
        }

        //open connection to database
        private bool OpenConnection()
        {
            try
            {
                connection.Open();
                return true;
            }
            catch (MySqlException ex)
            {
                //The two most common error numbers when connecting are as follows:
                //0: Cannot connect to server.
                //1045: Invalid user name and/or password.
                switch (ex.Number)
                {
                    case 0:
                        throw new System.Exception("0 - Cannot connect to server");

                    case 1045:
                        throw new System.Exception("1045 - Invalid username/password");
                }
                return false;
            }
        }

        //Close connection
        private bool CloseConnection()
        {
            try
            {
                connection.Close();
                return true;
            }
            catch (MySqlException ex)
            {
                throw new System.Exception(ex.Message);
            }
        }

        //Insert statement
        public void Insert(Dictionary<string, string> ins, string table) //a dictionary data structure
is used to easily accomodate any number of columns in insert
        {
            int k=0;
            string query = "INSERT INTO "+table+" SET ";
            foreach (KeyValuePair<string, string> column in ins) {
```

```
        if (++k > 1) query += ", ";
        query += column.Key+" = '"+column.Value+"'";
    }

    //open connection
    if (this.OpenConnection() == true)
    {
        //create command and assign the query and connection from the constructor
        MySqlCommand cmd = new MySqlCommand(query, connection);

        //Execute command
        cmd.ExecuteNonQuery();

        //close connection
        this.CloseConnection();
    }
}

//Update statement
public void Update(Dictionary<string, string> ins, string updateKey, string table)
{
    int k = 0;
    string query = "UPDATE " + table + " SET ";
    foreach (KeyValuePair<string, string> column in ins)
    {
        if (column.Key != updateKey) {
            if (++k > 1) query += ", ";
            query += column.Key + " = '" + column.Value + "'";
        }
    }
    query += "WHERE "+updateKey+" = '"+ins[updateKey]+"'";

    //Open connection
    if (this.OpenConnection() == true)
    {
        //create mysql command
        MySqlCommand cmd = new MySqlCommand();
        //Assign the query using CommandText
        cmd.CommandText = query;
        //Assign the connection using Connection
        cmd.Connection = connection;

        //Execute query
        cmd.ExecuteNonQuery();

        //close connection
        this.CloseConnection();
    }
}

//Delete statement
public void Delete(string key, string value, string table)
{
```

```
            string query = "DELETE FROM " + table + " WHERE "+key+" = '"+value+"'";
            if (this.OpenConnection() == true)
            {
                MySqlCommand cmd = new MySqlCommand(query, connection);
                cmd.ExecuteNonQuery();
                this.CloseConnection();
            }
        }

        //Select statement
        public DataTable Select(Dictionary<string, string> search, string table, string connector =
"AND")
        {
            int k = 0;
            DataTable dbTable = new DataTable();

            string query = "SELECT * FROM " + table;
            if (search != null)
            {
                query += " WHERE ";
                foreach (KeyValuePair<string, string> column in search)
                {
                    if (++k > 1) query += " " + connector + " ";
                    query += column.Key + " = '" + column.Value + "'";
                }
            }

            //Open connection
            if (this.OpenConnection() == true)
            {
                //Create Command
                MySqlCommand cmd = new MySqlCommand(query, connection);
                //Create a data reader and Execute the command
                MySqlDataReader dataReader = cmd.ExecuteReader();
                if (dataReader.HasRows)
                    dbTable.Load(dataReader);

                //close Data Reader
                dataReader.Close();

                //close Connection
                this.CloseConnection();

            }
            return dbTable;
        }

        public DataTable Query(string query)
        {
            DataTable dbTable = new DataTable();

            //Open connection
            if (this.OpenConnection() == true)
```

```
        {
            //Create Command
            MySqlCommand cmd = new MySqlCommand(query, connection);
            //Create a data reader and Execute the command
            MySqlDataReader dataReader = cmd.ExecuteReader();
            if (dataReader.HasRows)
                dbTable.Load(dataReader);

            //close Data Reader
            dataReader.Close();

            //close Connection
            this.CloseConnection();

        }
        return dbTable;
    }

    //Count statement
    public int Count(string table)
    {
        string query = "SELECT Count(*) FROM "+table;
        int count = -1;

        //Open Connection
        if (this.OpenConnection() == true)
        {
            //Create Mysql Command
            MySqlCommand cmd = new MySqlCommand(query, connection);

            //ExecuteScalar will return one value
            count = int.Parse(cmd.ExecuteScalar() + "");

            //close Connection
            this.CloseConnection();
        }
        return count;
    }
  }
}
```

Office.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Genetic
{
  class Office
  {
    public int id { get; set; }
```

```
        public int capacity { get; set; }
        public string proximity { get; set; }
        public string properties { get; set; }
        public string type { get; set; }
        public int toilet { get; set; }
        public int resources { get; set; }

        public string Display()
        {
            Console.Write("Room: " + id + "; Properties: " + properties + "; TypeID: " + type + ";
Capacity:" + capacity);
            return "Room: " + id + "; Properties: " + properties + "; TypeID: " + type + ";
Capacity:" + capacity + "\r\n";
        }
        public void Copy(Office other)
        {
            this.capacity = other.capacity;
            this.properties = other.properties;
            this.type = other.type;
            this.id = other.id;
            this.proximity = other.proximity;
            this.toilet = other.toilet;
            this.resources = other.resources;
        }
    }
}
```

Program.cs

```
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Genetic
{
    class Program
    {
        static int np = 20; /* The size of the problem (new breed+new generation)*/
        static int Population = np / 2; //ie. number of solutions, equal to num of new breed
        static int crossOver = 100;
        static Solution[] problems = new Solution[Population]; //problems - initial solution
        static int chromosomeSearchLimit = 5;
        public static int[] tmp = new  int[100];
        public static int[] officeCapacity = new int[100];
        public static string[] officeWhoIsThere = new string[100];

        //upper bounds
        static int staff_ub, office_ub;

        public static Random r = new Random();
        static void Main(string[] args)
```

```
    {
        Stopwatch stpWatch = new Stopwatch();
        stpWatch.Start();

        MySQLDatabase myDb = new MySQLDatabase();

        //int maxGene = 2000; /*The number of genes for generation {a stopping criteria}*/
        int maxGene = 2; /*The number of genes for generation {a stopping criteria}*/
        //int runtime = 30;  /*Algorithm can be run many times in order to see its robustness*/
        int runtime = 1;  /*Algorithm can be run many times in order to see its robustness*/

        //use inputset class to handle input
        InputSet.Load(); //loading inputs

        //setting upper bounds
        staff_ub = InputSet.staffList.Count - 1;
        office_ub = InputSet.officesList.Count - 1;

        Solution fittest = new Solution(InputSet.staffList.Count);

        //continue here after writing other functions
        //double mean=0;

        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"C:\Users\Public\gen_results.txt"))
        {
            for (int run = 0; run < runtime; run++)
            {
                stpWatch.Reset();
                stpWatch.Start();
                Console.WriteLine("Run: " + (run + 1));
                file.WriteLine("Run: " + (run + 1));
                InputSet.loadOfficeCapacity();
                InitPopulation();
                //fittest = GetFittest();
                for (int iter = 0; iter < maxGene; iter++)
                {
                    Console.WriteLine("iteration: " + (iter + 1));
                    file.WriteLine("iteration: " + (iter + 1));
                    SelectionOperation();
                    System.Threading.Thread.Sleep(31000);
                    CalcSelections();
                    CrossOverOperator();

                    fittest = GetFittest();
                    MutationOperator();
                }
                Console.WriteLine((run + 1).ToString() + ":");
                file.WriteLine((run + 1).ToString() + ":");
                Console.WriteLine("Best Allocation:");
                file.WriteLine("Best Allocation:");
                Array.Clear(tmp, 0, tmp.Length);
                Dictionary<string, string> solsToInsert = new Dictionary<string, string>();
```

```
                string fit_sol = fittest.Print();
                file.WriteLine(fit_sol);
                stpWatch.Stop();
                TimeSpan ts = stpWatch.Elapsed;
                String elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours,
ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
                Console.WriteLine("=========");
                file.WriteLine("=========");
                Console.WriteLine("RunTime: " + elapsedTime);
                file.WriteLine("RunTime: " + elapsedTime);

                long memory = GC.GetTotalMemory(true);
                double mem = memory / 1024;
                mem = Math.Round(mem, 2);
                Console.WriteLine("Memory Used: {0}KB", mem.ToString());
                file.WriteLine("Memory Used: {0}KB", mem.ToString());
                Console.WriteLine();
                file.WriteLine(" ");

                foreach (Allocation solAlloc in fittest.allocs)
                {
                    solsToInsert.Add("run", run.ToString());
                    solsToInsert.Add("staff_id", solAlloc.staffID.ToString());
                    solsToInsert.Add("office_id", solAlloc.officeID.ToString());
                    solsToInsert.Add("penalty", fittest.penalty.ToString());
                    solsToInsert.Add("alorithm_time", elapsedTime);
                    solsToInsert.Add("memory_used", mem.ToString() + "KB");
                    myDb.Insert(solsToInsert, "solutions");
                    solsToInsert.Clear();
                }
            }
        }

        Console.Read();
    }

    static void InitPopulation()
    {
        for (int i = 0; i < Population; i++)
        {
            problems[i] = new Solution(InputSet.staffList.Count);
            for (int k = 0; k < InputSet.staffList.Count; k++)
            {
                int randOffice = 0;

InputSet.DeclareArray(InputSet.officeGroups[InputSet.staffList[k].typeId].Count);
                randOffice = InputSet.getOfficeByTypeName(InputSet.staffList[k].typeId,
InputSet.staffList[k].dept);
                tmp[k] = randOffice;
                officeWhoIsThere[randOffice] = InputSet.staffList[k].dept;
                officeCapacity[randOffice] -= 1;
                problems[i].allocs[k] = new Allocation(InputSet.staffList[k].id, randOffice);
```

```
            }
            problems[i].calcPenalty();
            Array.Clear(tmp, 0, tmp.Length);
            InputSet.loadOfficeCapacity();
        }
    }

    static void ReInitPopulation(int i)
    {
        for (int k = 0; k < InputSet.staffList.Count; k++)
        {
            int randOffice = Convert.ToInt16(0 + r.NextDouble() *
(InputSet.officeGroups[InputSet.staffList[k].typeId].Count()-1));
            problems[i].allocs[k] = new Allocation(InputSet.staffList[k].id,
InputSet.officesList[randOffice].id);
        }
        problems[i].calcPenalty();
    }

    static Solution GetFittest() {
        Solution fittest = problems[0];
        for (int i = 1; i < problems.Length; i++)
        {
            if (fittest.penalty > problems[i].penalty) fittest = problems[i];
        }
        return fittest;
    }

    static double SumFitness()
    {
        double fitSum = 0;
        for (int i = 1; i < problems.Length; i++)
        {
            fitSum += problems[i].getFitness();
        }
        return fitSum;
    }

    static void SelectionOperation()
            {
        for (int i = 0; i < Population; i++)
        {
            for (int q = 0; q < chromosomeSearchLimit; q++)
            {
                int d = InputSet.staffList.Count - 1;
                Solution newSol = new Solution(InputSet.staffList.Count);
                newSol.Copy(problems[i]);

                /*The parameter to be changed is determined randomly*/
                int param2change = Convert.ToInt16(r.NextDouble() * d);

                /*A randomly chosen neighbour solution*/
                int neighbour = Convert.ToInt16(r.NextDouble() * (Population - 1));
```

```
                /*Randomly selected solution must be different from the solution i*/
                while (neighbour == i)
                    neighbour = Convert.ToInt16(r.NextDouble() * (Population - 1));

                newSol.allocs[param2change].officeID =
problems[neighbour].allocs[param2change].officeID;
                newSol.calcPenalty();

                if (newSol.penalty < problems[i].penalty)
                    problems[i].Copy(newSol);
                else
                    problems[i].offsprings++; /*if the solution i can not be improved, increase its
trial counter*/
            }

        }
    }

    static void CalcSelections()
    {
        for (int i = 1; i < problems.Length; i++)
        {
            problems[i].selection = problems[i].getFitness() / SumFitness();
        }
    }

    static void CrossOverOperator()
    {
        int i = 0, t = 0;

        while (t < Population)
        {
            if (r.NextDouble() < problems[i].selection)
            {
                t++;
                int d = InputSet.staffList.Count - 1;
                Solution newSol = new Solution(InputSet.staffList.Count);
                newSol.Copy(problems[i]);

                /*The parameter to be changed is determined randomly*/
                int param2change = Convert.ToInt16(r.NextDouble() * d);

                /*A randomly chosen neighbour solution*/
                int neighbour = Convert.ToInt16(r.NextDouble() * (Population - 1));

                /*Randomly selected solution must be different from the solution i*/
                while (neighbour == i)
                    neighbour = Convert.ToInt16(r.NextDouble() * (Population - 1));

                newSol.allocs[param2change].officeID =
problems[neighbour].allocs[param2change].officeID;
                newSol.calcPenalty();
```

```
                if (newSol.penalty < problems[i].penalty)
                    problems[i].Copy(newSol);
                else
                    problems[i].offsprings++; /*if the solution i can not be improved, increase its
trial counter*/
            }
            i++;
            if (i == Population)
                i = 0;
        }
    }

    /*determine the food sources whose trial counter exceeds the "trialsLimit" value. In Basic
ABC, only one scout is allowed to occur in each cycle*/
    static void MutationOperator()
    {
        int maxChromeIndex, i;
        maxChromeIndex = 0;
        for (i = 1; i < Population; i++)
        {
            if (problems[i].offsprings > problems[maxChromeIndex].offsprings)
                maxChromeIndex = i;
        }
        if (problems[maxChromeIndex].offsprings >= crossOver)
        {
            ReInitPopulation(maxChromeIndex);
        }
    }
  }
}
```

Solution.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace Genetic
{
  class Solution // AKA Population Source
  {
    public double penalty { get; set; } //inverse is the fitness; the breed value
    public double selection;
    public int offsprings;
    public Allocation[] allocs;

    public Solution(int numSol)
    {
      //numSol is the number of staff in the inputset
      allocs = new Allocation[numSol];
```

```
}

public Solution(Solution mutation)
{
   allocs = new Allocation[mutation.allocs.Length];
   for (int i = 0; i < mutation.allocs.Length; i++)
   {
      allocs[i] = new Allocation(mutation.allocs[i].staffID, mutation.allocs[i].officeID);
   }
   penalty = mutation.penalty;
   selection = mutation.selection;
}

public string Print()
{
   string to_ret = "";
   for (int i = 0; i < allocs.Length; i++)
   {
      to_ret += allocs[i].Print();
   }
   Console.WriteLine("Penalty: " + penalty);
   to_ret += "Penalty: " + penalty;
   return to_ret;
}

public void Copy(Solution mutation)
{
   allocs = new Allocation[mutation.allocs.Length];
   for (int i = 0; i < mutation.allocs.Length; i++)
   {
      allocs[i] = new Allocation(mutation.allocs[i].staffID, mutation.allocs[i].officeID);
   }
   penalty = mutation.penalty;
   selection = mutation.selection;
}

public double getFitness()
{
   return 1 / penalty;
}
public void calcPenalty()
{
   MySQLDatabase myDb = new MySQLDatabase();
   //selecting data from table
   DataTable records;
   penalty = 0;

   for (int i = 0; i < allocs.Length; i++)
   {
      try
      {
         //selecting office records
```

```
            records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'staff' AND entity_id =
"'+Convert.ToString(allocs[i].staffID)+"'");
            if (records != null && records.Rows.Count > 0)
            {
               foreach (DataRow row in records.Rows)
                  penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
                  //penalty += Convert.ToDouble(row["weight"]);
            }

            records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'office' AND entity_id = '" +
Convert.ToString(allocs[i].officeID) + "'");
            if (records != null && records.Rows.Count > 0)
            {
               foreach (DataRow row in records.Rows)
                  //penalty += Convert.ToDouble(row["weight"]);
                  penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
            }
         }
         catch (Exception e)
         {
            Console.WriteLine("Error selecting: " + e.Message);
         }
      }
      //Console.WriteLine("Penalty: " + penalty);
   }

   int Violates(string constraintType, Allocation presentAlloc, DataRow constraintRow)
   {
      switch (constraintType)
      {
         case "allocation":
            if (presentAlloc.officeID !=
Convert.ToInt16(constraintRow["concerned_entity_id"]))
               return Convert.ToInt16(constraintRow["weight"]);
            break;

         case "non-allocation":
            if (presentAlloc.officeID ==
Convert.ToInt16(constraintRow["concerned_entity_id"]))
               return Convert.ToInt16(constraintRow["weight"]);
            break;

         case "same-room":
            for (int i = 0; i < allocs.Length; i++)
            {
               if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID != presentAlloc.officeID)
                  return Convert.ToInt16(constraintRow["weight"]);
            }
```

```
                break;

            case "not-same-room":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID == presentAlloc.officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "not-sharing":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (presentAlloc.officeID == allocs[i].officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "capacity": //remember to put a counter such that for each solution, if the
constraint is capacity, it can only be called once
                int occupied = 0;
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (presentAlloc.officeID == allocs[i].officeID)
                        occupied++;
                }
                if (occupied > InputSet.GetOfficeById(presentAlloc.officeID).capacity)
                    return Convert.ToInt16(constraintRow["weight"]);
                break;

            case "nearby": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be defined only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID != allocs[i].officeID &&
myOffice.proximity.IndexOf("," + yourOffice.id + ",") < 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") < 0)
                        { //this means the constraint has been violated bcos the offices are not nearby
as expected
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
```

```
                break;

            case "away-from": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be define only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID == allocs[i].officeID ||
myOffice.proximity.IndexOf("," + yourOffice.id + ",") >= 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") >= 0)
                        { //this means the constraint has been violated bcos the offices are nearby
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are not in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
                break;
            default:
                return 0;
        }
        return 0;
    }
  }
}
```

Staff.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Genetic
{
   class Staff
   {
      public string[] cadarA = new string[3];
      public string[] cadarB = new string[3];
      public string[] cadarC = new string[5];
      public int id { get; set; }
      public string staffName { get; set; }
      public string cadre { get; set; }
      public string typeId { get; set; }
      public string dept { get; set; }

      public string Display()
      {
```

```
        Console.Write("Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " +
typeId);
        return "Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " + typeId +
"\r\n";
      }
      public void Copy(Staff other)
      {
        this.staffName = other.staffName;
        this.cadre = other.cadre;
        this.typeId = other.typeId;
      }
      public void initCadre()
      {
        //Type A
        this.cadarA[0] = "HOD";
        this.cadarA[1] = "PROF";

        //Type B
        this.cadarB[0] = "SL";
        this.cadarB[1] = "READER";

        //Type C
        this.cadarC[0] = "AL";
        this.cadarC[1] = "L1";
        this.cadarC[2] = "L2";
        this.cadarC[3] = "GA";

      }
   }
}
```

HYBRID SOURCE CODE ICS

Allocation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_abc_osap
{
   class Allocation
   {
      public int staffID { get; set; }
      public int officeID { get; set; }

      public Allocation (int staff, int office)
```

```
        {
            staffID = staff;
            officeID = office;
        }
        public string Print()
        {
            string to_ret = "";
            string staff_rec = InputSet.GetStaffById(staffID).Display();
            to_ret += staff_rec+"\n";
            Console.Write(" ---- Allocated to: ");
            to_ret += " ---- Allocated to: " + "\n";
            string office_rec = InputSet.GetOfficeById(officeID).Display();
            to_ret += office_rec + "\r\n";
            Console.Write("\n");
            return to_ret;
        }
    }
}
```

InputSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace tabu_abc_osap
{
    class InputSet
    {
        static Office[] offices;
        public static List<Office> officesList = new List<Office>();
        public static Dictionary<string, List<Office>> officeGroups = new
Dictionary<string,List<Office>>();

        static Staff[] staff;
        public static List<Staff> staffList = new List<Staff>();
        public static Dictionary<string, List<Staff>> staffGroups = new Dictionary<string,
List<Staff>>();
        static int[] officeGrp;

        public static void DeclareArray(int numOff)
        {
            //numSol is the number of office in the inputset
            officeGrp = new int[numOff];
        }

        public static void Load()
        {
            int i;

            MySQLDatabase myDb = new MySQLDatabase();
```

```
//selecting data from table
DataTable records;
try
{
   //selecting office records
   //Console.WriteLine("1");
   records = myDb.Select(null, "offices");//null means to fetch all data from the table
   //Console.WriteLine("2");
   if (records != null && records.Rows.Count > 0)
   {
      offices = new Office[records.Rows.Count];
      i = 0;
      foreach (DataRow row in records.Rows)
      {

         //Console.WriteLine(row["id"].GetType());
         //string d = row["type"].ToString();
         offices[i] = new Office();

         offices[i].id = (int)row["id"];
         offices[i].type = row["type"].ToString();
         offices[i].properties = row["properties"].ToString();
         offices[i].capacity = (int)row["capacity"];
         offices[i].toilet = row["toilet"].ToString() == "" ? 0 : (int)row["toilet"];
         offices[i].resources = (int)row["resources"];
         offices[i].proximity = row["proximity"].ToString();

         officesList.Add(offices[i]);
         if (!officeGroups.ContainsKey(offices[i].type))
            officeGroups[offices[i].type] = new List<Office>();
         officeGroups[offices[i].type].Add(offices[i]);
      }
   }
   else
      Console.WriteLine("No office record found!");

   //System.Threading.Thread.Sleep(1000000);

   //selecting staff records
   //Console.WriteLine("3");
   records = myDb.Select(null, "staff");//null means to fetch all data from the table
   //Console.WriteLine("4");
   if (records != null && records.Rows.Count > 0)
   {
      staff = new Staff[records.Rows.Count];
      i = 0;
      foreach (DataRow row in records.Rows)
      {
         staff[i] = new Staff();
         staff[i].id = Convert.ToInt16(row["id"]);
         staff[i].typeId = Convert.ToString(row["type_id"]);
         staff[i].cadre = Convert.ToString(row["cadre"]);
         staff[i].staffName = Convert.ToString(row["staff_name"]);
```

```
            staff[i].dept = Convert.ToString(row["dept"]);

            staffList.Add(staff[i]);

            //this part might not be needed
            if (!staffGroups.ContainsKey(staff[i].typeId))
               staffGroups[staff[i].typeId] = new List<Staff>();
            staffGroups[staff[i].typeId].Add(staff[i]);
            //this part might not be needed
         }
      }
      else
         Console.WriteLine("No staff record found!");

   }
   catch (Exception e)
   {
      Console.WriteLine("Error selecting: " + e.Message);
      System.Threading.Thread.Sleep(1000000);
   }
}

public static void loadOfficeCapacity()
{
   foreach (Office office in officesList)
   {
      Program.officeCapacity[office.id] = office.capacity;
   }
}

public static Office GetOfficeById(int id)
{
   foreach (Office office in officesList)
   {
      if (office.id == id) return office;
   }
   return null;
}

public static int GetOfficeIndexById(int id)
{
   for (int i = 0; i < officesList.Count(); i++)
   {
      if (officesList[i].id == id) return i;
   }
   return -1;
}

public static Staff GetStaffById(int id)
{
   foreach (Staff staff in staffList)
   {
      if (staff.id == id) return staff;
```

```
        }
        return null;
    }

    public static int getOfficeByTypeName(string typey, string dept)
    {
    Found:
        int k = 0;
        foreach (Office office in officesList)
        {
            if (office.type == typey)
            {
                if (office.resources == 1)
                {
                    if (crossCheck(office.id) && checkCapacity(office.id) > 0 &&
checkWhoIsInTheOffice(office.id, dept) && isItProfSuit(typey, office.toilet))
                    {
                        officeGrp[k] = office.id;
                        k += 1;
                    }
                }
            }
        }
        //Console.WriteLine(ox + " T:" + typey);
        if (k == 0)
        {
            //Console.WriteLine(k);
            ClearTypeRoom(typey);
            goto Found;
        }
        //System.Threading.Thread.Sleep(600);
        int rnd = Program.r.Next(0, k - 1);
        return officeGrp[rnd];
    }

    static void ClearTypeRoom(string typey)
    {
        for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
        {
            Office xx = GetOfficeById(Program.tmp[numTmp]);
            if (xx != null)
            {
                if (xx.type == typey)
                {
                    Program.tmp[numTmp] = 0;
                }
            }
        }
    }

    static bool crossCheck(int officeid)
    {
        for (int numTmp = 0; numTmp < Program.tmp.Length; numTmp++)
```

```
            {
                if (officeid == Program.tmp[numTmp])
                {
                    return false;
                }
            }
            return true;
        }

        static int checkCapacity(int officeid)
        {
            return Program.officeCapacity[officeid];
        }

        static bool checkWhoIsInTheOffice(int officeid, string dept)
        {
            if (Program.officeWhoIsThere[officeid] == dept || Program.officeWhoIsThere[officeid]
== null)
            {
                return true;
            }

            return false;
        }

        static bool isItProfSuit(string typed, int toilet)
        {
            if (typed == "A" && toilet == 0)
            {
                return false;
            }
            return true;
        }

        static int[] ShuffleArray(int[] array)
        {
            Random r = new Random();
            for (int i = array.Length; i > 0; i--)
            {
                int j = r.Next(i);
                int k = array[j];
                array[j] = array[i - 1];
                array[i - 1] = k;
            }
            return array;
        }

    }
}



/*
```

```
        //THIS PART IS TO TEST THE DATABASE CLASS
        string db = "vp_validation_system";
        string dbuid = "root";
        string dbpwd = "";

        //instantiating the db class
        MySQLDatabase myDb = new MySQLDatabase(dbuid, dbpwd, db);

        //selecting data from table - there shouldn't be any records for now
        System.Data.DataTable records;
        try
        {
            records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
            // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
            Console.WriteLine("All records inserted:");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
                    Console.Write(row["code"] + " || ");
                    Console.WriteLine(row["used"].ToString());
                }
            }
            else
                Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error selecting: " + e.Message);
        }

        //insert record to table 'sample_codes'
        Dictionary<string, string>[] insertData = new Dictionary<string, string>[3]; //An array
of dictionaries, each dictionary represents a row to be inserted
        insertData[0] = new Dictionary<string, string>();
        insertData[0].Add("code", "563443y3uh87grr"); //column name, value in table
'sample_codes'
        insertData[0].Add("used", "0"); //another column name, value in table 'sample_codes'

        insertData[1] = new Dictionary<string, string>();
        insertData[1].Add("code", "fhbeiurhg34u23434");
        insertData[1].Add("used", "0");

        insertData[2] = new Dictionary<string, string>();
        insertData[2].Add("code", "fdhrbru3u9rwei9jcks");
        insertData[2].Add("used", "0");

        try
```

```
        {
            myDb.Insert(insertData[0], "sample_codes");
            myDb.Insert(insertData[1], "sample_codes");
            myDb.Insert(insertData[2], "sample_codes");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error inserting: " + e.Message);
        }

        //selecting data from table
        Console.WriteLine();
        try
        {
            records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
            // You could just add a statement 'using System.Data;' at the beginning and then use
'System.Data.DataTable' just as 'DataTable'
            Console.WriteLine("All records inserted:");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
                    Console.Write(row["code"] + " || ");
                    Console.WriteLine(row["used"].ToString());
                }
            }
            else
                Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
            Console.WriteLine("Error selecting: " + e.Message);
        }

        //another select
        Console.WriteLine();
        Dictionary<string, string> search = new Dictionary<string, string>();
        search.Add("id", "1");
        search.Add("used", "0"); //this is not necessary, but just to indicate that u could have
multiple search conditions

        try
        {
            records = myDb.Select(search, "sample_codes");
            Console.WriteLine("Record with id=1 and used=0 (should be only one record):");
            if (records != null && records.Rows.Count > 0)
            {
                foreach (System.Data.DataRow row in records.Rows)
                {
                    Console.Write(row["id"] + " || "); //use double quotes
                    Console.Write(row["code"] + " || ");
```

```
                Console.WriteLine(row["used"].ToString());
            }
        }
        else
            Console.WriteLine("No record found!");
    }
    catch (Exception e)
    {
        Console.WriteLine("Error selecting: " + e.Message);
    }

    //update
    Dictionary<string, string> updateData = new Dictionary<string, string>();
    updateData.Add("code", "5555555555555555");
    updateData.Add("id", "2"); //this will be only used in the WHERE clause and not to
update bcos I will instruct the function so via its parameter
    try
    {
        myDb.Update(updateData, "id", "sample_codes");
    }
    catch (Exception e)
    {
        Console.WriteLine("Error updating: " + e.Message);
    }

    //delete
    try
    {
        myDb.Delete("id", "3", "sample_codes"); //deleting record with id=3
    }
    catch (Exception e)
    {
        Console.WriteLine("Error deleting: " + e.Message);
    }


    //selecting everything again to see changes made with uodate and delete
    Console.WriteLine();
    try
    {
        records = myDb.Select(null, "sample_codes");//null means to fetch all data from the
table
        Console.WriteLine("All records after modifications:");
        if (records != null && records.Rows.Count > 0)
        {
            foreach (System.Data.DataRow row in records.Rows)
            {
                Console.Write(row["id"] + " || "); //use double quotes
                Console.Write(row["code"] + " || ");
                Console.WriteLine(row["used"].ToString());
            }
        }
        else
```

```
            Console.WriteLine("No record found!");
        }
        catch (Exception e)
        {
           Console.WriteLine("Error selecting: " + e.Message);
        }

        //Using the function 'Count'
        Console.WriteLine();
        Console.WriteLine("Number of all records left in table 'sample_codes': " +
myDb.Count("sample_codes"));
        Console.Read();
    }*/
```

MySQLDatabase.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Data;
using MySql.Data.MySqlClient;

namespace tabu_abc_osap
{
   class MySQLDatabase
   {
      private MySqlConnection connection;
      private string server = "localhost";
      private string database = "abc_osap";
      private string uid = "root";
      private string password = "";

      //Constructor
      //public MySQLDatabase(string uname, string pwd, string db)
      public MySQLDatabase()
      {
         Initialize();
         //Initialize(uname, pwd, db);
         //In case there any other thing one would like to have done on instantiation aside from
just initializing
      }

      //Initialize values
      //private void Initialize(string uname, string pwd, string db)
      private void Initialize()
      {
         //server = "localhost";
         //database = db;
         //uid = uname;
         //password = pwd;
         string connectionString;
```

```
            connectionString = "SERVER=" + server + ";" + "DATABASE=" +
                    database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";

            connection = new MySqlConnection(connectionString);
        }

        //open connection to database
        private bool OpenConnection()
        {
            try
            {
                connection.Open();
                return true;
            }
            catch (MySqlException ex)
            {
                //The two most common error numbers when connecting are as follows:
                //0: Cannot connect to server.
                //1045: Invalid user name and/or password.
                switch (ex.Number)
                {
                    case 0:
                        throw new System.Exception("0 - Cannot connect to server");

                    case 1045:
                        throw new System.Exception("1045 - Invalid username/password");
                }
                return false;
            }
        }

        //Close connection
        private bool CloseConnection()
        {
            try
            {
                connection.Close();
                return true;
            }
            catch (MySqlException ex)
            {
                throw new System.Exception(ex.Message);
            }
        }

        //Insert statement
        public void Insert(Dictionary<string, string> ins, string table) //a dictionary data structure
is used to easily accomodate any number of columns in insert
        {
            int k=0;
            string query = "INSERT INTO "+table+" SET ";
            foreach (KeyValuePair<string, string> column in ins) {
                if (++k > 1) query += ", ";
```

```
            query += column.Key+" = '"+column.Value+"'";
        }

        //open connection
        if (this.OpenConnection() == true)
        {
            //create command and assign the query and connection from the constructor
            MySqlCommand cmd = new MySqlCommand(query, connection);

            //Execute command
            cmd.ExecuteNonQuery();

            //close connection
            this.CloseConnection();
        }
    }

    //Update statement
    public void Update(Dictionary<string, string> ins, string updateKey, string table)
    {
        int k = 0;
        string query = "UPDATE " + table + " SET ";
        foreach (KeyValuePair<string, string> column in ins)
        {
            if (column.Key != updateKey) {
                if (++k > 1) query += ", ";
                query += column.Key + " = '" + column.Value + "'";
            }
        }
        query += "WHERE "+updateKey+" = '"+ins[updateKey]+"'";

        //Open connection
        if (this.OpenConnection() == true)
        {
            //create mysql command
            MySqlCommand cmd = new MySqlCommand();
            //Assign the query using CommandText
            cmd.CommandText = query;
            //Assign the connection using Connection
            cmd.Connection = connection;

            //Execute query
            cmd.ExecuteNonQuery();

            //close connection
            this.CloseConnection();
        }
    }

    //Delete statement
    public void Delete(string key, string value, string table)
    {
        string query = "DELETE FROM " + table + " WHERE "+key+" = '"+value+"'";
```

```
            if (this.OpenConnection() == true)
            {
                MySqlCommand cmd = new MySqlCommand(query, connection);
                cmd.ExecuteNonQuery();
                this.CloseConnection();
            }
        }

        //Select statement
        public DataTable Select(Dictionary<string, string> search, string table, string connector =
"AND")
        {
            int k = 0;
            DataTable dbTable = new DataTable();

            string query = "SELECT * FROM " + table;
            if (search != null)
            {
                query += " WHERE ";
                foreach (KeyValuePair<string, string> column in search)
                {
                    if (++k > 1) query += " " + connector + " ";
                    query += column.Key + " = '" + column.Value + "'";
                }
            }

            //Open connection
            if (this.OpenConnection() == true)
            {
                //Create Command
                MySqlCommand cmd = new MySqlCommand(query, connection);
                //Create a data reader and Execute the command
                MySqlDataReader dataReader = cmd.ExecuteReader();
                if (dataReader.HasRows)
                    dbTable.Load(dataReader);

                //close Data Reader
                dataReader.Close();

                //close Connection
                this.CloseConnection();

            }
            return dbTable;
        }

        public DataTable Query(string query)
        {
            DataTable dbTable = new DataTable();

            //Open connection
            if (this.OpenConnection() == true)
            {
```

```
            //Create Command
            MySqlCommand cmd = new MySqlCommand(query, connection);
            //Create a data reader and Execute the command
            MySqlDataReader dataReader = cmd.ExecuteReader();
            if (dataReader.HasRows)
               dbTable.Load(dataReader);

            //close Data Reader
            dataReader.Close();

            //close Connection
            this.CloseConnection();

         }
         return dbTable;
      }

      //Count statement
      public int Count(string table)
      {
         string query = "SELECT Count(*) FROM "+table;
         int count = -1;

         //Open Connection
         if (this.OpenConnection() == true)
         {
            //Create Mysql Command
            MySqlCommand cmd = new MySqlCommand(query, connection);

            //ExecuteScalar will return one value
            count = int.Parse(cmd.ExecuteScalar() + "");

            //close Connection
            this.CloseConnection();
         }
         return count;
      }
   }
}
```

Office.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_abc_osap
{
   class Office
   {
      public int id { get; set; }
      public int capacity { get; set; }
```

```
        public string proximity { get; set; }
        public string properties { get; set; }
        public string type { get; set; }
        public int toilet { get; set; }
        public int resources { get; set; }

        public string Display()
        {
            Console.Write("Room: " + id + "; Properties: " + properties + "; TypeID: " + type + "; 
Capacity:" + capacity);
            return "Room: " + id + "; Properties: " + properties + "; TypeID: " + type + "; 
Capacity:" + capacity + "\r\n";
        }
        public void Copy(Office other)
        {
            this.capacity = other.capacity;
            this.properties = other.properties;
            this.type = other.type;
            this.id = other.id;
            this.proximity = other.proximity;
            this.toilet = other.toilet;
            this.resources = other.resources;
        }
    }
}
```

Program.cs

```
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_abc_osap
{
    class Program
    {
        static int np = 20; /* The number of colony size (employed bees+onlooker bees)*/
        static int foodNumber = np/2; //ie. number of solutions, equal to num of employed bees
        static int trials = 5;
        static int trialsLimit = 100;  /*A food source which could not be improved through 
"trialsLimit" trials is abandoned by its employed bee*/
        static Solution[] foods = new Solution[foodNumber]; //foods - initial solution
        public static int[] tmp = new int[100];
        public static int[] officeCapacity = new int[100];
        public static string[] officeWhoIsThere = new string[100];

        //Tabu Memory structures
        static List<int> tabuOffice = new List<int>();
        static List<int> tabuOfficeIds = new List<int>();

        //upper bounds
```

```
        static int staff_ub, office_ub;

        public static Random r = new Random();
        static void Main(string[] args)
        {
            Stopwatch stpWatch = new Stopwatch();
            stpWatch.Start();

            MySQLDatabase myDb = new MySQLDatabase();

            //int maxCycle = 2500; /*The number of cycles for foraging {a stopping criteria}*/
            int maxCycle = 2; /*The number of cycles for foraging {a stopping criteria}*/
            //int runtime = 30;  /*Algorithm can be run many times in order to see its robustness*/
            int runtime = 1;  /*Algorithm can be run many times in order to see its robustness*/

            //use inputset class to handle input
            InputSet.Load(); //loading inputs

            //setting upper bounds
            staff_ub = InputSet.staffList.Count - 1;
            office_ub = InputSet.officesList.Count - 1;

            Solution fittest = new Solution(InputSet.staffList.Count);

            //continue here after writing other functions
            //double mean=0;
            //string[] lines = new string[3];
            using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"C:\Users\Public\tabu_abc_results.txt"))
            {
                for (int run = 0; run < runtime; run++)
                {
                    stpWatch.Reset();
                    stpWatch.Start();
                    Console.WriteLine("Run: " + (run + 1));
                    file.WriteLine("Run: " + (run + 1));
                    InputSet.loadOfficeCapacity();
                    Init();
                    //fittest = GetFittest();
                    for (int iter = 0; iter < maxCycle; iter++)
                    {
                        //lines[0]
                        Console.WriteLine("iteration: " + (iter + 1));
                        file.WriteLine("iteration: " + (iter + 1));
                        //fittest.Print();
                        SendEmployedBees();
                        System.Threading.Thread.Sleep(2000);
                        CalcProbabilities();
                        SendOnlookerBees();
                        fittest = GetFittest();
                        SendScoutBees();
                    }
                    Console.WriteLine((run + 1).ToString() + ":");
```

```
            file.WriteLine((run + 1).ToString() + ":");
            Console.WriteLine("Best Allocation:");
            file.WriteLine("Best Allocation:");
            Dictionary<string, string> solsToInsert = new Dictionary<string, string>();

            string fit_sol = fittest.Print();
            file.WriteLine(fit_sol);
            /*//System.out.println("%d. run: %e \n",run+1,GlobalMin);
                        System.out.println((run+1)+".run:"+bee.GlobalMin);
                        bee.GlobalMins[run]=bee.GlobalMin;
                        mean=mean+bee.GlobalMin;*/
            stpWatch.Stop();
            TimeSpan ts = stpWatch.Elapsed;
            String elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours,
ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
            Console.WriteLine("=========");
            file.WriteLine("=========");
            Console.WriteLine("RunTime: " + elapsedTime);
            file.WriteLine("RunTime: " + elapsedTime);

            long memory = GC.GetTotalMemory(true);
            double mem = memory / 1024;
            mem = Math.Round(mem, 2);
            Console.WriteLine("Memory Used: {0}KB", mem.ToString());
            file.WriteLine("Memory Used: {0}KB", mem.ToString());
            Console.WriteLine();
            file.WriteLine(" ");

            foreach (Allocation solAlloc in fittest.allocs)
            {
               solsToInsert.Add("run", run.ToString());
               solsToInsert.Add("staff_id", solAlloc.staffID.ToString());
               solsToInsert.Add("office_id", solAlloc.officeID.ToString());
               solsToInsert.Add("penalty", fittest.penalty.ToString());
               solsToInsert.Add("alorithm_time", elapsedTime);
               solsToInsert.Add("memory_used", mem.ToString() + "KB");
               myDb.Insert(solsToInsert, "solutions");
               solsToInsert.Clear();
            }
         }
         /*mean=mean/bee.runtime;
         //System.out.println("Means of %d runs: %e\n",runtime,mean);
         System.out.println("Means  of "+bee.runtime+"runs: "+mean);*/
      }

      Console.Read();
   }


   static void Init()
   {
      for (int i = 0; i < foodNumber; i++)
      {
```

```
            foods[i] = new Solution(InputSet.staffList.Count);
            //Console.WriteLine("Staff Count: "+InputSet.staffList.Count);
            for (int k = 0; k < InputSet.staffList.Count; k++)
            {
               int randOffice = 0;//Convert.ToInt16(r.NextDouble() *
(InputSet.officesList.Count() - 1));

InputSet.DeclareArray(InputSet.officeGroups[InputSet.staffList[k].typeId].Count);
               randOffice = InputSet.getOfficeByTypeName(InputSet.staffList[k].typeId,
InputSet.staffList[k].dept);
               tmp[k] = randOffice;
               officeWhoIsThere[randOffice] = InputSet.staffList[k].dept;
               officeCapacity[randOffice] -= 1;
               //Console.WriteLine(InputSet.staffList[k].cadre + "(" + InputSet.staffList[k].id +
")==" + randOffice + "(" + InputSet.GetOfficeById(randOffice).properties + ")");
               foods[i].allocs[k] = new Allocation(InputSet.staffList[k].id, randOffice);
            }
            foods[i].calcPenalty();
            Array.Clear(tmp, 0, tmp.Length);
            InputSet.loadOfficeCapacity();
         }
         //Console.Read();
      }

      static void ReInit(int i)
      {
         //foods[i] = new Solution(InputSet.staffList.Count);
         for (int k = 0; k < InputSet.staffList.Count; k++)
         {
            int randOffice = Convert.ToInt16(0 + r.NextDouble() *
(InputSet.officeGroups[InputSet.staffList[k].typeId].Count()-1));
            int x = 0;
            while
(tabuOfficeIds.Contains(InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(randO
ffice).id) && InputSet.staffList[k].typeId !=
InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(randOffice).type && ++x <
trials)
               randOffice = Convert.ToInt16(0 + r.NextDouble() *
InputSet.officeGroups[InputSet.staffList[k].typeId].Count());

            if
(tabuOfficeIds.Contains(InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(randO
ffice).id) || InputSet.staffList[k].typeId !=
InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(randOffice).type)
            {
               randOffice = Convert.ToInt16(0 + r.NextDouble() * office_ub);
               foods[i].allocs[k] = new Allocation(InputSet.staffList[k].id,
InputSet.officesList[randOffice].id);
               if (--InputSet.officesList[randOffice].capacity < 0)
               {
                  tabuOffice.Add(randOffice);
                  tabuOfficeIds.Add(InputSet.officesList[randOffice].id);
               }
```

```
            }
            else
            {
               foods[i].allocs[k] = new Allocation(InputSet.staffList[k].id,
InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(randOffice).id);

               //find a way to get the index of
InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(randOffice) in officesList, the
replace randOffice with it below
               int tabuOfficeIndex =
InputSet.GetOfficeIndexById(InputSet.officeGroups[InputSet.staffList[k].typeId].ElementAt(r
andOffice).id);

               if (--InputSet.officesList[randOffice].capacity < 0)
               {
                  tabuOffice.Add(tabuOfficeIndex);
                  tabuOfficeIds.Add(InputSet.officesList[tabuOfficeIndex].id);
               }
            }
         }
         foods[i].calcPenalty();
      }

      static Solution GetFittest() {
         Solution fittest = foods[0];
         for (int i = 1; i < foods.Length; i++) {
            if (fittest.penalty > foods[i].penalty) fittest = foods[i];
         }
         return fittest;
      }

      static double SumFitness()
      {
         double fitSum = 0;
         for (int i = 1; i < foods.Length; i++)
         {
            fitSum += foods[i].getFitness();
         }
         return fitSum;
      }

            static void SendEmployedBees()
            {
         for (int i=0; i<foodNumber; i++)
         {
            int d = InputSet.staffList.Count-1;
            Solution newSol = new Solution(InputSet.staffList.Count);
            newSol.Copy(foods[i]);

            /*The parameter to be changed is determined randomly*/
            int param2change = Convert.ToInt16(r.NextDouble() * d);

            /*A randomly chosen neighbour solution*/
```

```
            int neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber-1));

            /*Randomly selected solution must be different from the solution i*/
            while (neighbour == i)
               neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber-1));

            newSol.allocs[param2change].officeID =
foods[neighbour].allocs[param2change].officeID;
            newSol.calcPenalty();

            if (newSol.penalty < foods[i].penalty)
               foods[i].Copy(newSol);
            else
               foods[i].trials++; /*if the solution i can not be improved, increase its trial
counter*/

         }
      }

      static void CalcProbabilities()
      {
         for (int i = 1; i < foods.Length; i++)
         {
            foods[i].probability = foods[i].getFitness() / SumFitness();
         }
      }

      static void SendOnlookerBees()
      {
         int i = 0, t = 0;

         while (t < foodNumber)
         {
            if (r.NextDouble() < foods[i].probability)
            {
               t++;
               int d = InputSet.staffList.Count - 1;
               Solution newSol = new Solution(InputSet.staffList.Count);
               newSol.Copy(foods[i]);

               /*The parameter to be changed is determined randomly*/
               int param2change = Convert.ToInt16(r.NextDouble() * d);

               /*A randomly chosen neighbour solution*/
               int neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber-1));

               /*Randomly selected solution must be different from the solution i*/
               while (neighbour == i)
                  neighbour = Convert.ToInt16(r.NextDouble() * (foodNumber-1));

               newSol.allocs[param2change].officeID =
foods[neighbour].allocs[param2change].officeID;
               newSol.calcPenalty();
```

```
                if (newSol.penalty < foods[i].penalty)
                    foods[i].Copy(newSol);
                else
                    foods[i].trials++; /*if the solution i can not be improved, increase its trial
counter*/
            }
            i++;
            if (i == foodNumber)
                i = 0;
        }
    }

    /*determine the food sources whose trial counter exceeds the "trialsLimit" value. In Basic
ABC, only one scout is allowed to occur in each cycle*/
    static void SendScoutBees()
    {
        int maxTrialIndex, i;
        maxTrialIndex = 0;
        for (i = 1; i < foodNumber; i++)
        {
            if (foods[i].trials > foods[maxTrialIndex].trials)
                maxTrialIndex = i;
        }
        if (foods[maxTrialIndex].trials >= trialsLimit)
        {
            ReInit(maxTrialIndex);
        }
    }
  }
}
```

Solution.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;

namespace tabu_abc_osap
{
  class Solution // AKA Food Source
  {
    public double penalty { get; set; } //inverse is the fitness; the nectar amount
    public double probability;
    public int trials;
    public Allocation[] allocs;

    public Solution(int numSol)
    {
      //numSol is the number of staff in the inputset
      allocs = new Allocation[numSol];
```

```
}

public Solution(Solution clone)
{
   allocs = new Allocation[clone.allocs.Length];
   for (int i = 0; i < clone.allocs.Length; i++)
   {
      allocs[i] = new Allocation(clone.allocs[i].staffID, clone.allocs[i].officeID);
   }
   penalty = clone.penalty;
   probability = clone.probability;
}

public string Print()
{
   string to_ret = "";
   for (int i = 0; i < allocs.Length; i++)
   {
      to_ret += allocs[i].Print();
   }
   Console.WriteLine("Penalty: "+penalty);
   to_ret += "Penalty: " + penalty;
   return to_ret;
   //return penalty.ToString();
   //file.WriteLine(penalty);
}

public void Copy(Solution clone)
{
   allocs = new Allocation[clone.allocs.Length];
   for (int i = 0; i < clone.allocs.Length; i++)
   {
      allocs[i] = new Allocation(clone.allocs[i].staffID, clone.allocs[i].officeID);
   }
   penalty = clone.penalty;
   probability = clone.probability;
}

public double getFitness()
{
   return 1 / penalty;
}
public void calcPenalty()
{
   MySQLDatabase myDb = new MySQLDatabase();
   //selecting data from table
   DataTable records;

   penalty = 0;
   for (int i = 0; i < allocs.Length; i++)
   {
      try
      {
```

```
            //selecting office records
            records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'staff' AND entity_id =
"'+Convert.ToString(allocs[i].staffID)+"'");
            if (records != null && records.Rows.Count > 0)
            {
               foreach (DataRow row in records.Rows)
                  penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
                  //penalty += Convert.ToDouble(row["weight"]);
            }

            records = myDb.Query("SELECT * FROM entity_constraints JOIN constraints
ON constraint_type LIKE type WHERE entity_kind LIKE 'office' AND entity_id = "' +
Convert.ToString(allocs[i].officeID) + "'");
            if (records != null && records.Rows.Count > 0)
            {
               foreach (DataRow row in records.Rows)
                  //penalty += Convert.ToDouble(row["weight"]);
                  penalty += Violates(Convert.ToString(row["constraint_type"]), allocs[i],
row);
            }
         }
         catch (Exception e)
         {
            Console.WriteLine("Error selecting: " + e.Message);
         }
      }
   }

   int Violates(string constraintType, Allocation presentAlloc, DataRow constraintRow)
   {
      switch (constraintType)
      {
         case "allocation":
            if (presentAlloc.officeID !=
Convert.ToInt16(constraintRow["concerned_entity_id"]))
               return Convert.ToInt16(constraintRow["weight"]);
            break;

         case "non-allocation":
            if (presentAlloc.officeID ==
Convert.ToInt16(constraintRow["concerned_entity_id"]))
               return Convert.ToInt16(constraintRow["weight"]);
            break;

         case "same-room":
            for (int i = 0; i < allocs.Length; i++)
            {
               if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID != presentAlloc.officeID)
                  return Convert.ToInt16(constraintRow["weight"]);
            }
```

```
                break;

            case "not-same-room":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) == allocs[i].staffID
&& allocs[i].officeID == presentAlloc.officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "not-sharing":
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (presentAlloc.officeID == allocs[i].officeID)
                        return Convert.ToInt16(constraintRow["weight"]);
                }
                break;

            case "capacity": //remember to put a counter such that for each solution, if the
constraint is capacity, it can only be called once
                int occupied = 0;
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (presentAlloc.officeID == allocs[i].officeID)
                        occupied++;
                }
                if (occupied > InputSet.GetOfficeById(presentAlloc.officeID).capacity)
                    return Convert.ToInt16(constraintRow["weight"]);
                break;

            case "nearby": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be defined only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID != allocs[i].officeID &&
myOffice.proximity.IndexOf("," + yourOffice.id + ",") < 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") < 0)
                        { //this means the constraint has been violated bcos the offices are not nearby
as expected
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
```

```
                break;

            case "away-from": //proximity for offices should be defined in both ways, but nearby
constraint for staff should be define only in one way
                for (int i = 0; i < allocs.Length; i++)
                {
                    if (Convert.ToInt16(constraintRow["concerned_entity_id"]) ==
allocs[i].staffID) //this is the staff whom I should be near
                    {//now check if both our offices are in close proximity. But what if we have the
same office?
                        Office myOffice = InputSet.GetOfficeById(presentAlloc.officeID);
                        Office yourOffice = InputSet.GetOfficeById(allocs[i].officeID);
                        if (presentAlloc.officeID == allocs[i].officeID ||
myOffice.proximity.IndexOf("," + yourOffice.id + ",") >= 0 &&
yourOffice.proximity.IndexOf("," + myOffice.id + ",") >= 0)
                        { //this means the constraint has been violated bcos the offices are nearby
                            return Convert.ToInt16(constraintRow["weight"]);
                        } //else, they are not in close proximity and the constraint is therefore not
violated
                        else
                            return 0;
                    }
                }
                break;

            default:
                return 0;
        }
        return 0;
    }
  }
}
```

Staff.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tabu_abc_osap
{
  class Staff
  {
    public int id { get; set; }
    public string staffName { get; set; }
    public string cadre { get; set; }
    public string typeId { get; set; }
    public string dept { get; set; }

    public string Display()
    {
```

```
            Console.Write("Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " +
typeId);
            return "Staff: " + id + "; Dept: " + dept + "; Cadre: " + cadre + "; TypeID: " + typeId +
"\r\n";
        }
        public void Copy(Staff other)
        {
            this.staffName = other.staffName;
            this.cadre = other.cadre;
            this.typeId = other.typeId;
        }
    }
}
```

APPENDIX D

TABU SEARCH

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 63; Properties: space: 13.38, tables: 1, chairs: 4, fans: 2, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 57; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to:Room: 65; Properties: space: 7.7, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:1

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to:Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 47; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 60; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 54; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to:Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 43; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 66; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2,

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 19; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to:Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 62; Properties: space: 26.76, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1,

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 64; Properties: space: 26.76, tables: 2, chairs: 5, fans: 1, AC: 1, cabinet: 2, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 55; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to:Room: 71; Properties: space: 26.76, tables: 2, chairs: 3, fans: 5, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 50; Dept: COM SC; Cadre:READER; TypeID: B, Allocated to: Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -,TV: -; TypeID: B; Capacity:1

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to: Room: 41; Properties: space: 27.69, tables: 2, chairs: 5, fans:1, AC: ,cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 52;Dept: TELCOM; Cadre: READER; TypeID:B,Allocated to:Room:36;Properties: space: 17.69, tables:1, chairs:3, fans:1, AC: 1, cabinet: 2, fridge: -,toilet: -,TV:-;TypeID: B; Capacity:1

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 42; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to:Room: 51; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 46; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 52; Properties: space: 21.08, tables: 1, chairs: 3, fans: , AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B , Allocated to: Room: 45; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to:Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 31; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 61; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 22; Properties: space: 17.69, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 27; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 63; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 61; Properties: space: 26.76, tables: 2, chairs: 4, fans: 2, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 70; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 4, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Penalty: 4030

RunTime: 00:18:26.40

Memory Used: 8951KB

Run: 6

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C , Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1,

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 60; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1,chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2,

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 57; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1,

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 47; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C , Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2
Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C , Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1
Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to: Room: 63; Properties: space: 13.38, tables: 1, chairs: 4, fans: 2, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2
Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1
Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2
Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 65; Properties: space: 7.7, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:1
Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 19; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2
Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 43; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:2
Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2
,Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 54; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 66; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to:Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C , Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to:Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C , Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 47; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: C; Capacity:2

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to:Room: 62; Properties: space: 26.76, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 55; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 64; Properties: space: 26.76, tables: 2, chairs: 5, fans: 1, AC: 1, cabinet: 2, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 71; Properties: space: 26.76, tables: 2, chairs: 3, fans: 5, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:1

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room: 27; Properties: space: 13.38,tables:1,chairs:3,fans:1,AC:1,cabinet:2, fridge:-, toilet:-, TV:-; TypeID:B; Capacity:1

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to: Room: 41; Properties: space: 27.69, tables: 2, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B;

Capacity:1

Staff: 52; Dept: TELCOM; Cadre: READER; TypeID: B,Allocated to:Room:36;Properties: space: 17.69,tables: 1, chairs:3, fans:1, AC:1, cabinet:2, fridge:-, toilet:-, TV:-; TypeID:B; Capacity:1

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 42; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 46; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 51; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 45; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 61; Properties: space: 26.76, tables: 2, chairs: 4, fans: 2, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 31; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 22; Properties: space: 17.69, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 61; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 52; Properties: space: 21.08, tables: 1, chairs: 3, fans: , AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 63; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 70; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 4, fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Penalty: 4120

=========

RunTime: 00:25:32.64

Memory Used: 50612KB

ABC RESULT

ABC Run: 1

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

1:

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 32; Properties: space: 13.72, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C ; Capacity:2

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, --- Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: - , cabinet:- , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to:Room: 32; Properties: space: 13.72, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C ; Capacity:2

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 32; Properties: space: 13.72, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C ; Capacity:2

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to:Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 52; Dept: TELCOM; Cadre: READER; TypeID: B, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: ,fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 36; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 61; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 63; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Penalty: 1640

=========

RunTime: 00:50:39.25

Memory Used: 664KB

Run: 6

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

6:

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 31; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 22; Properties: space: 17.69, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 46; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:1

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to:Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to:Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, - Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to:Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to:Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to:Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 10; Properties: space: 7.7, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 27; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 27; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: , Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C,Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to:Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 27; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to:Room: 27; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C ,Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A , Allocated to:Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to:Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B,, Allocated to:Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 52; Dept: TELCOM; Cadre: READER; TypeID: B, Allocated to: Room: 15; Properties: space:13.38,tables:1,chairs:3, fans:1, AC:, cabinet:1, fridge:-, toilet:-, TV:-; TypeID:C; Capacity:2

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 13; Properties: space: 13.38, tables: -, chairs: -, fans: -, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 61; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 63; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Penalty: 2490

RunTime: 00:52:47.15

Memory Used: 666KB

GENETIC RESULT

Run: 1

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

1:

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C ,Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C , Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C,Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C , Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 36; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C , Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C , Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C , Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

,Staff: 33; Dept: LIB; Cadre: L1; TypeID: C,Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C , Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 36; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to:Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans:1, AC:, cabinet: 1, fridge:-, toilet: -, TV:-; TypeID: B; Capacity:1

Staff: 52; Dept: TELCOM; Cadre:READER; TypeID: B, Allocated to: Room: 2; Properties: space: 13.38, tables:1, chairs:3, fans: 1,AC: -, cabinet: 1, fridge:-, toilet:-,TV:-; TypeID: C; Capacity:2

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 27; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 61; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 63; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Penalty: 1760

RunTime: 01:12:45.06

Memory Used: 648KB

Run: 6

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

6:

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to:Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C , Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 36; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C ,Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to:Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C , Allocated to:Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C ,Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C , Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to:Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 32; Properties: space: 13.72, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C ; Capacity:2

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to:Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C , Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to:Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C ,Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A , Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to:Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room: 17; Properties: space:17.69, tables:1, chairs:3, fans:1, AC:,cabinet:1, fridge:-,toilet:-, TV:-;TypeID: B; Capacity:1

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 52; Dept: TELCOM; Cadre: READER; TypeID: B , Allocated to: Room: 25; Properties: space: 17.69, tables: , chairs: , fans: , AC:, cabinet: , fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 12; Properties: space: 13.38, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:1

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:2

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 61; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 63; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:1

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to:Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet:3, fridge:-, toilet:-, TV:-;TypeID: B; Capacity:1

Penalty: 1730

RunTime: 01:09:50.71

Memory Used: 650KB

HYBRID RESULT

Run: 1

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

1:

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-31

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to:Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-40

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-35

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C , Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-20

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-47

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-35

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-36

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-46

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 57; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:-26

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to:Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-43

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet:1, fridge:-, toilet:-,TV:-;TypeID: C; Capacity:-54

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans:1, AC:-, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-54

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-34

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C , Allocated to: Room: 4; Properties: space: 10.35, tables:1, chairs:4, fans:2, AC:-, cabinet:2, fridge:-, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs:3, fans:1, AC:-, cabinet:1, fridge:-, toilet:-, TV -; TypeID:C; Capacity:-54

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C , Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-35

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-35

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to:Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-37

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 47; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: C; Capacity:-43

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-36

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-44

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C, Allocated to:, Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-37

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs:3, fans:1, AC:, cabinet:2, fridge:-, toilet: -, TV: -; TypeID: C; Capacity:-46

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC:, cabinet: , fridge:-, toilet:-, TV:-; TypeID:C; Capacity:-42,

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC:, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-38

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC:, cabinet: 2, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-52

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-36

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to:Room: 65; Properties: space: 7.7, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:-38

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 63; Properties: space: 13.38, tables: 1, chairs: 4, fans: 2, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:-36

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs:3, fans:1, AC:-, cabinet:1, fridge:-, toilet:-, TV: -; TypeID: C; Capacity:-43

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 43; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:-51

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 66; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:-40

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-46

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-20

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-35

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 43; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:-51

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to:Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-31

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs:2, fans:1, AC:, cabinet:, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet:2, fridge:-, toilet:-, TV:-; TypeID:C; Capacity:-38

Staff: 43; Dept: MASS COM; Cadre:L2; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC:-, cabinet:1, fridge:-,toilet:-, TV:-; TypeID: C; Capacity:-43

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to:Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans:1, AC:-, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-54

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 19; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-33

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 55; Properties: space: 13.38, tables: 1, chairs: 5, fans:1, AC:1, cabinet:1, fridge:, toilet: 1,TV: ; TypeID: A; Capacity:-37

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 64; Properties: space: 26.76, tables: 2, chairs: 5, fans: 1, AC: 1, cabinet: 2, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:-34

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 62; Properties: space: 26.76, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge:, toilet: 1, TV: ; TypeID: A; Capacity:-37

Staff: 49; Dept: COM SC; Cadre: PROF;TypeID: A, Allocated to:Room:71;Properties: space: 26.76, tables: 2, chairs: 3, fans:5, AC:1,cabinet: 1, fridge: , toilet: 1, TV:;TypeID: A; Capacity:-37

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room:27; Properties: space: 13.38, tables:1, chairs:3, fans:1, AC:1, cabinet:2, fridge: -, toilet:-, TV:-; TypeID: B; Capacity:-40

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to: Room: 31; Properties: space: 13.38, tables:1, chairs:3, fans:1, AC:1, cabinet:1, fridge:-, toilet:-, TV: -; TypeID: B; Capacity:-40

Staff:52;Dept:TELCOM;Cadre:READER;TypeID: B, Allocated to: Room:46; Properties: space:26.76, tables:1,chairs:3,fans:1, AC:1, cabinet:,fridge:, toilet:, TV:; TypeID: B; Capacity:-47

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to: Room: 61; Properties: space: 26.76, tables: 2, chairs: 4, fans: 2, AC: , cabinet: 1, fridge: , toilet: ,TV:; TypeID: B; Capacity:-39

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:-36

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:-36

Staff: 56; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 42; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:-44

Staff: 57; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:-49

Staff: 58; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 52; Properties: space: 21.08, tables: 1, chairs: 3, fans: , AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:-28,

Staff:59; Dept: MASS COM; Cadre:SL; TypeID:B, Allocated to:Room: 36;Properties:space: 17.69, tables:1, chairs:3, fans:1, AC:1, cabinet: 2, fridge:-, toilet: -,TV:-; TypeID: B; Capacity:-43

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 51; Properties: space: 26.76, tables:1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:-37

Staff: 61; Dept: COM SC; Cadre: SL;TypeID: B, Allocated to:Room: 22; Properties: space: 17.69, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet:-, TV: -; TypeID: B; Capacity:-41,

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet:3, fridge: -, toilet:-, TV: -; TypeID: B;Capacity:-49

Staff: 63; Dept: COM SC; Cadre: SL;TypeID:B, Allocated to: Room:14; Properties: space: 17.69, tables:1, chairs:3, fans: 1, AC:, cabinet: 1, fridge:-, toilet: -, TV:-; TypeID: B; Capacity:-32

Staff:64; Dept:COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 70; Properties: space: 13.38, tables:1, chairs:3, fans:1, AC:1, cabinet: 4, fridge: , toilet: , TV: ; TypeID: B; Capacity:-40

Penalty: 3960

RunTime: 00:09:15.04

Memory Used: 688KB

Run: 6

iteration: 1

iteration: 2

iteration: 3

iteration: 4

iteration: 5

iteration: 6

iteration: 7

iteration: 8

iteration: 9

iteration: 10

iteration: 11

iteration: 12

iteration: 13

iteration: 14

iteration: 15

iteration: 16

iteration: 17

iteration: 18

iteration: 19

iteration: 20

iteration: 21

iteration: 22

iteration: 23

iteration: 24

iteration: 25

iteration: 26

iteration: 27

iteration: 28

iteration: 29

iteration: 30

iteration: 31

iteration: 32

iteration: 33

iteration: 34

iteration: 35

iteration: 36

iteration: 37

iteration: 38

iteration: 39

iteration: 40

iteration: 41

iteration: 42

iteration: 43

iteration: 44

iteration: 45

iteration: 46

iteration: 47

iteration: 48

iteration: 49

iteration: 50

iteration: 51

iteration: 52

iteration: 53

iteration: 54

iteration: 55

iteration: 56

iteration: 57

iteration: 58

iteration: 59

iteration: 60

iteration: 61

iteration: 62

iteration: 63

iteration: 64

iteration: 65

iteration: 66

iteration: 67

iteration: 68

iteration: 69

iteration: 70

iteration: 71

iteration: 72

iteration: 73

iteration: 74

iteration: 75

iteration: 76

iteration: 77

iteration: 78

iteration: 79

iteration: 80

iteration: 81

iteration: 82

iteration: 83

iteration: 84

iteration: 85

iteration: 86

iteration: 87

iteration: 88

iteration: 89

iteration: 90

iteration: 91

iteration: 92

iteration: 93

iteration: 94

iteration: 95

iteration: 96

iteration: 97

iteration: 98

iteration: 99

iteration: 100

6:

Best Allocation:

Staff: 1; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 57; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2,fridge: , toilet: , TV: ; TypeID: C; Capacity:-35

Staff: 2; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to:Room: 60; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 2,fridge: , toilet: , TV: ; TypeID: C; Capacity:-51

Staff: 3; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 24; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 4; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-36,

Staff: 5; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 29; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 6; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-32

Staff: 7; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 4; Properties: space: 10.35, tables: 1, chairs: 4, fans: 2, AC: -, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-40

Staff: 8; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 47; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: C; Capacity:-35

Staff: 9; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-18

Staff: 10; Dept: COM SC; Cadre: AL; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs:2, fans: 1, AC:, cabinet:, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-49

Staff: 11; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 65; Properties: space: 7.7, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:-31

Staff: 12; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables: 2, chairs: 4, fans:1, AC:, cabinet:1, fridge:-, toilet:-, TV: -; TypeID: C; Capacity:-44

Staff: 13; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 14; Dept: TELCOM; Cadre: AL; TypeID: C, Allocated to: Room: 18; Properties: space: 7.7, tables: 1, chairs: 4, fans: -, AC: -, cabinet: 1, fridge -, toilet:-, TV:-; TypeID: C; Capacity:-39

Staff: 15; Dept: MASS COM; Cadre: AL; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables:1, chairs:1, fans: 1, AC:-, cabinet:-, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-46

Staff: 16; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 35; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-32

Staff: 17; Dept: ICS; Cadre: AL; TypeID: C, Allocated to: Room: 23; Properties: space: 8.56, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-43

Staff: 18; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 39; Properties: space: 13.38, tables: 2, chairs: 4, fans: 1, AC: , cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-51

Staff: 19; Dept: ICS; Cadre: GA; TypeID: C, Allocated to: Room: 7; Properties: space: 13.38, tables: 1, chairs: 1, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-44

Staff: 20; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-44

Staff: 21; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 2; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 22; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 23; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 24; Dept: LIB; Cadre: L1; TypeID: C, Allocated to:Room: 16; Properties: space: 9.34, tables: 1, chairs: 3, fans: 1, AC: , cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 25; Dept: TELCOM; Cadre: L1; TypeID: C, Allocated to: Room: 3; Properties: space: 10.35, tables: , chairs: , fans: , AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-38

Staff: 26; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to: Room: 34; Properties: space: 8.56, tables: 1, chairs: 2, fans:1, AC:1, cabinet:1, fridge:-, toilet:-, TV: -; TypeID: C; Capacity:-38

Staff: 27; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 40; Properties: space: 7.7, tables: 2, chairs: 4, fans:1, AC:, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-44

Staff: 28; Dept: COM SC; Cadre: L1; TypeID: C, Allocated to: Room: 47; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: C; Capacity:-35

Staff: 29; Dept: ICS; Cadre: L1; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-45

Staff: 30; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 33; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-44

Staff: 31; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 21; Properties: space: 13.38, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-37

Staff: 32; Dept: MASS COM; Cadre: L1; TypeID: C, Allocated to:Room: 8; Properties: space: 13.38, tables: 2, chairs: 1, fans:1, AC:-, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-48

Staff: 33; Dept: LIB; Cadre: L1; TypeID: C, Allocated to: Room: 19; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: -, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-30

Staff: 34; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 66; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: , TV: ; TypeID: C; Capacity:-33

Staff: 35; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 26; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-49

Staff: 36; Dept: ICS; Cadre: L2; TypeID: C, Allocated to: Room: 38; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-45

Staff: 37; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 28; Properties: space: 13.38, tables: 1, chairs: 2, fans: 1, AC: , cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 38; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 37; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-39

Staff: 39; Dept: TELCOM; Cadre: L2; TypeID: C, Allocated to: Room: 57; Properties: space: 13.38, tables: 1, chairs: 3, fans:1, AC:, cabinet: 2, fridge: , toilet: , TV: ; TypeID: C; Capacity:-35

Staff: 40; Dept: MASS COM; Cadre: L2; TypeID: C , Allocated to: Room: 8; Properties: space: 13.38, tables:2, chairs:1, fans:1, AC:-, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-48

Staff: 41; Dept: ICS; Cadre: L2; TypeID: C, Allocated to:Room: 15; Properties: space: 13.38, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-36

Staff: 42; Dept: COM SC; Cadre: L2; TypeID: C, Allocated to: Room: 1; Properties: space: 17.69, tables: 2, chairs: 7, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-18

Staff: 43; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 6; Properties: space: 13.38, tables: 1, chairs:1, fans:1, AC:-, cabinet:-, fridge:-, toilet:-, TV:-; TypeID: C; Capacity:-46

Staff: 44; Dept: MASS COM; Cadre: L2; TypeID: C, Allocated to: Room: 11; Properties: space: 13.38, tables:2, chairs:4, fans:1, AC:, cabinet:, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-44

Staff: 45; Dept: LIB; Cadre: L2; TypeID: C, Allocated to: Room: 30; Properties: space: 13.38, tables: 1, chairs: 2, fans: 2, AC: -, cabinet: -, fridge: -, toilet: -, TV: -; TypeID: C; Capacity:-42

Staff: 46; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to: Room: 64; Properties: space: 26.76, tables:2, chairs:5, fans:1, AC:1, cabinet: 2, fridge:, toilet: 1, TV: ; TypeID: A; Capacity:-48

Staff: 47; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 62; Properties: space: 26.76, tables: 2, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:-39

Staff: 48; Dept: LIB; Cadre: PROF; TypeID: A, Allocated to: Room: 55; Properties: space: 13.38, tables: 1, chairs: 5, fans: 1, AC: 1, cabinet: 1, fridge: , toilet: 1, TV: ; TypeID: A; Capacity:-41

Staff: 49; Dept: COM SC; Cadre: PROF; TypeID: A, Allocated to:Room:71; Properties: space: 26.76, tables: 2, chairs: 3, fans: 5, AC: 1, cabinet:1, fridge:, toilet:1, TV:; TypeID:A; Capacity:-44

Staff: 50; Dept: COM SC; Cadre: READER; TypeID: B, Allocated to: Room:14; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC:, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: B; Capacity:-43

Staff: 51; Dept: ICS; Cadre: READER; TypeID: B, Allocated to: Room: 17; Properties: space: 17.69, tables: 1, chairs: 3, fans: 1, AC:, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: B; Capacity:-36

Staff: 52; Dept: TELCOM; Cadre: READER; TypeID: B, Allocated to: Room: 36; Properties: space:17.69,tables:1,chairs:3,fans:1, AC:1,cabinet:2,fridge:-,toilet:-,TV:-;TypeID:B;Capacity:-39

Staff: 53; Dept: LIB; Cadre: READER; TypeID: B, Allocated to:Room: 45; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 2, fridge: , toilet:, TV:; TypeID:B;Capacity:-35

Staff: 54; Dept: LIB; Cadre: SL; TypeID: B, Allocated to: Room: 52; Properties: space: 21.08, tables: 1, chairs: 3, fans: , AC: , cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:-30

Staff: 55; Dept: LIB; Cadre: SL; TypeID: B , Allocated to: Room: 22; Properties: space: 17.69, tables: 1, chairs: 4, fans: 1, AC: 1, cabinet: 1, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:-49

Staff:56; Dept:COM SC;Cadre:SL; TypeID: B, Allocated to: Room: 41; Properties: space:27.69, tables: 2, chairs: 5, fans: 1, AC: , cabinet: 1, fridge: , toilet: , TV: ; TypeID: B; Capacity:-40

Staff: 57; Dept: COM SC; Cadre: SL;TypeID: B, Allocated to: Room: 27; Properties: space: 13.38, tables:1, chairs:3, fans:1, AC:1, cabinet: 2, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:-43

Staff:58; Dept:MASS COM; Cadre:SL; TypeID: B, Allocated to: Room: 31; Properties: space: 13.38, tables:1, chairs:3, fans:1, AC:1, cabinet:1, fridge:-, toilet:-, TV:-; TypeID: B; Capacity:-36

Staff: 59; Dept: MASS COM; Cadre: SL; TypeID: B, Allocated to: Room: 42; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: , fridge: , toilet:, TV: ; TypeID: B; Capacity:-38

Staff: 60; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room:61; Properties: space: 26.76, tables: 2, chairs: 4, fans: 2, AC:, cabinet:1, fridge:, toilet:, TV:; TypeID: B; Capacity:-29

Staff: 61;Dept:COM SC;Cadre:SL; TypeID: B, Allocated to:Room: 51; Properties: space: 26.76, tables: 1, chairs: 3, fans: 1, AC: , cabinet: 1, fridge: , toilet: ,TV: ; TypeID: B; Capacity:-40

Staff: 62; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 5; Properties: space: 17.85, tables: 2, chairs:4, fans:1, AC: 1, cabinet: 3, fridge: -, toilet: -, TV: -; TypeID: B; Capacity:-45

Staff: 63;Dept:COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 46; Properties: space: 26.76, tables:1, chairs: 3, fans:1, AC: 1, cabinet: , fridge: , toilet: , TV: ; TypeID: B; Capacity:-43

Staff: 64; Dept: COM SC; Cadre: SL; TypeID: B, Allocated to: Room: 70; Properties: space: 13.38, tables: 1, chairs: 3, fans:1, AC:1, cabinet:4, fridge:, toilet: , TV: ; TypeID: B; Capacity:-34

Penalty: 3980

RunTime: 00:11:01.32

Memory Used: 697KB

APPENDIX E

ROOM NO	TYPE	NO. OF TYPE STAFF		CLOSE BY (1)	cadre of staff	space capacity	dept	table(s)	chair(s)	fan(s)	AC	cabinet	fridge	toilet	TV
1	C	2 (c)	OR/W 1 (1)	(1) 2,4,5,6,11,12,32,33,34,35,36,37,38,39,40	L1	17.69	LIB	2	7	2	-	-	-	-	-
2	C	2 (c)	1 (B) W	(2) 1,3,4,5,6,7,8,11,12,14,15,32,33,34,35,36,37,38,39,40	L1	13.38	LIB	1	3	1	-	1	-	-	-
3	C	1 (c)		(3) 4,5,6,7,8,11,12,14,15,2,1,32,33,34,35,36,37,38,39,40	L1	10.35	LIB						-	-	-
4	C	1 (c)		(4) 5,6,7,8,11,12,14,15,16,17,1,2,3,40,39,38,37,36,35,	SL	10.35	LIB	1	4	2	-	2	-	-	-
5	B	1 (B)		(5) 6,7,8,11,12,14,15,16,17,18,21,4,3,2,1,40,39,38,37,36,35	SL	17.85	LIB	2	4	1	1	3	-	-	-
6	C	2 (c)	OR 1 (B)	(6) 7,8,11,12,14,15,16,17,18,21,5,4,3,2,1,40,39,38,37,37,36,35	L2	13.38	LIB	1	1	1	-	-	-	-	-
7	C	2 (c)			L1	13.38	LIB	1	1	1	-	1	-	-	-
8	C	2 ©			L1	13.38	LIB	2	1	1	-	1	-	-	-
9	C	2 ©			-	13.38	-	-	-	-	-	-	-	-	-
10	C	1 ©			-	7.7							-	-	-
11	C	2 ©	OR 1 (B) W		2(AL)	13.38	TELCOM	2	4	1		1	-	-	-
12	C	2 ©			-	13.38							-	-	-
13	C	2 ©			-	13.38	-	-	-	-	-	-	-	-	-
14	B	1 (B) W		(14) 15,16,17,18,21,22,23,24,25,12,11,8,7,6,5,4,3,2,1	SL	17.69	COM SC	1	3	1		1	-	-	-
15	C	2 ©			L2	13.38	TELCOM	1	3	1		1	-	-	-
16	C	1 ©		(16) 17,18,21,22,23,24,25,12,11,8,7,6,5,4,3,2,1	L1	9.34	TELCOM	1	3	1		-	-	-	-
17	B			(17) 18,21,22,23,24,25,16,15,14,12,11,8,7,6,5,4,3,2,1	SL	17.69	COM SC	1	3	1		1	-	-	-
18	C	1 ©		(18) 21,22,23,24,25,17,16,15,14,12,11,8,7,6,	AL	7.7	MASS COM	1	4	-	-	1	-	-	-
19	C	2©			AL	13.38	MASS COM	1	2	1	-	1	-	-	-
20	C	2 ©			-	13.38	-	-	-	-	-	-	-	-	-
21	C	2 ©			SL	13.38	MASS	1	4	1	1	1	-	-	-

							COM								
22	B	1 (B)			SL	17.69	MASS COM	1	4	1	1	1	-	-	-
23	C	1 ©		(23) 24,25,26,27,28,29,30,31,22,21,18,17,16,14,	L2	8.56	MASS COM	1	3	1		2	-	-	-
24	C	2 ©	OR 1 (B) W	(24) 25,26,27,28,29,30,31,23,22,21,18,17,16,15,14	L1	13.38	MASS COM	1	5	1		1	-	-	-
25	B	1 (B)			-	17.69							-	-	-
26	C	2 ©	OR 1 (B) W	(26) 27,28,29,30,31,32,33,34,35,36,25,24,23,22,21	SL	13.38	COM SC	1	2	1			-	-	-
27	B	1 (B)		(27) 28,29,30,31,32,33,34,35,36,26,25,24,23,22,21	READER	13.38	COM SC	1	3	1	1	2	-	-	-
28	C	2 ©	OR 1 (B) W	(28) 29,30,31,32,33,34,35,36,37,38,27,26,25,24,23,22,21	SL	13.38	COM SC	1	2	1		2	-	-	-
29	C	2 ©	OR 1 (B) W	(29) 30,31,32,33,34,35,36,37,38,28,27,26,25,24,23,22	L1	13.38	COM SC	1	3	1	-	1	-	-	-
30	C	2 ©	OR 1 (B) W	(30) 31,32,33,34,35,36,37,38,29,28,27,26,25,24,23,22	L1	13.38	COM SC	1	2	2	-	-	-	-	-
31	B	1 (B)		(31) 32,33,34,35,36,37,38,39,40,30,29,28,27,26,25,24,23,	SL	13.38	COM SC	1	3	1	1	1	-	-	-
32	C	2 ©	OR 1 (B) W	(32) 33,34,35,36,37,38,39,40,1,31,30,29,28,27,26,25,24	2(AL)	13.72	ICS	2	4	1		1	-	-	-
33	C	2 ©	OR 1 (B) W	(33) 34,35,36,37,38,39,40,1,32,31,30,29,28,27,26	L1	13.38	ICS	1	3	1	1	1	-	-	-
34	C	1 ©		(34) 35,36,37,38,39,40,1,2,3,4,5,33,32,31,29,28,27,26	L2	8.56	ICS	1	2	1	1	1	-	-	-
35	C	2 ©			READER	13.38	ICS	1	3	1	1	1	-	-	-
36	B	1 (B)		(36) 37,38,39,40,1,2,3,4,5,6,35,34,33,32,31,30,29,28,27,26	SL	17.69	COM SC	1	3	1	1	2	-	-	-
37	C	2 ©	OR 1 (B) W	(37) 38,39,40,1,2,3,4,5,6,7,8,36,35,34,33,32,31,30,29,28,27,26	L2	13.38	COM SC	1	3	1			-	-	-

38	C	2 ©	OR 1 (B) W	(38) 39,40,1,2,3,4,5,6,7,8,37,36,35,34,33,32,31	L1	13.38	LIB	1	3	1		2	-	-	-
39	C	2 ©	OR 1 (B) W	(39) 40,1,2,3,4,5,6,7,8,11,12,38,37,36,35,34,33,32,31	GA/L2/AL	13.38	ICS	2	4	1		3	-	-	-
40	C	1 ©		(40) 1,2,3,4,5,6,7,8,11,12,39,38,37,366,35,34,33,32,31	AL/GA	7.7	ICS	2	4	1		1	-	-	-
41	B	1 (B) w		(41) 42, ics, c5, 43, 45	2(L2)	27.69	ICS	2	5	1		1			
42	B	1 (B)	1 (A) W	(42) ICS, CS, 43, 45, 46, 41	AL	26.76	ICS	1	3	1					
43	C	2 ©	1 (B)	(43), 45, 46, 47, Telcomm, Mass comm.	2(AL)	13.38	ICS	2	4	1		2			
44	C	2 ©			-	13.38									
45	B	1 (B)	or (A)		L1	26.76	LIB	1	3	1		2			
46	B	1 (B)		(46) 47,Telcomm, Mass Comm, 43, 42, 41, CS, ICS	READER	26.76	TELCOM	1	3	1	1				
47	C	2 ©	or 1 (B) w	(47) 52, 53, 54, 55, 56, 57, 46, 43, 42, 41, CS, ICS	AL	13.38	TELCOM	1	2	1					
48	C	2 ©	or 1 (B) w	48	-	13.38									
49	B	1 (B)	or 1 (A) w	49	-	26.76									
50	B	1 (B)	or 1 (A) w	50	-	26.76									
51	B	1 (B)	or 1 (A) w	(51), 52, 53, 54, 55, 56, 57, 60	L2	26.76	MASS COM	1	3	1		1			
52	B	1 (B)	or 1 (A) w	(52), 53,54,55,56,57,60	AL	21.08	MASS COM	1	3						
53	A	1 (A)		(53), 54, 55, 56, 57, 60, 61, 52, 51	2(AL), L2	26.76	COM SC	3	8	2	1	1	1		
54	C	2 ©	or 1 (B) w	(54),55, 56, 56, 57, 60, 61, 53, 52, 51	L2	13.38	MASS COM	1	2	1		1			
55	A	1 (A)		(55) 56, 57, 60, 61, 62, 63, 54, 53 ,52, 51	PROF	13.38	COM SC	1	5	1	1	1		1	
56	A	1 (A)		(56) 57, 60, 61, 62, 63, 64, 55, 54, 53, 52, 51	-	26.76	ICS								
57	C	2 ©	or 1(B) w penalty	(57) 60, 61, 62, 63 , 64, 56 ,55, 54, 53, 52, 51	L1	13.38	MASS COM	1	3	1		2			
58	B	1 (B)		58	-	26.76									

59	C	1 ©		59	-	7.7									
60	C	2 ©	or 1(B) w penalty	(60) 61, 62, 63, 64, 57, 56, 55, 54	2(AL)	13.38	COM SC	2	4	1		2			
61	B	1 (B)		(61) 62, 63, 64, 65, 60, 57, 56, 55	2(ST)	26.76		2	4	2		1			
62	A	1 (A)		(62) 63, 64, 61, 60, 57, 56, 55	PROF	26.76	LIB	2	4	1	1	1		1	
63	C	2 ©	or 1(B) w penalty	(63) 64, 62, 61, 60, 57, 56	L1	13.38	LIB	1	4	2		2			
64	A	1 (A)		(64) 63, 62, 61, 60, 57, 56	PROF	26.76	LIB	2	5	1	1	2		1	
65	C	1©			L2	7.7	LIB	1	3	1		1			
66	C	2©	or 1(B) w		READER	13.38	LIB	1	3	1	1	1			
67	C	2 ©	or 1(B) w			13.38									
68	C	1 ©				7.7									
69	B	1 (B)				26.76	COM SC								
70	B	1 (B)			SL	13.38	COM SC	1	3	1	1	4			
71	A	1 (A)			PROF	26.76	COM SC	2	3	5	1	1		1	
72	B	1 (B)				26.76									
73	B	1 (B)				26.76									
74	B	1 (B)				26.76									
75	B	1 (B)				26.76									
76	B	1 (B)				26.76									
77	B	1 (B)				26.76									
78	C	2 ©	or 1(B) w			13.38									
79	B	1 (B)				26.76									
80	C	2 ©				13.38									
81	A	1 (A)				40.14									

DATA SET FOR FACULTY OF COMMUNICATION AND INFORMATION SCIENCES AS USED BY OSAP