A Hierarchical Heuristic Strategy for Hostel Space Allocation Problem

Adewumi, A.O.[1,*] & Ali, M.M.[1]

[1] School of Computer Sciences, University of KwaZulu-Natal, Durban, South Africa

[2] School of Computational & Applied Mathematics, University of the Witwatersrand, South Africa

Abstract

Hostel space allocation is a big concern for universities administration in developing countries where accommodation facilities are provided, given the recent rate of increase in admissions and subsequent requests for campus residence. This concern stems from many but sometimes conflicting factors. The allocation of limited hostel spaces available to competing students thus becomes a great challenge to university administration. The application of well-known operations research techniques to handle the allocation process has not been studied.

In this paper, we present a hierarchical-based heuristics solution for this new class of space allocation problem under domain specific constraints. We applied different inter-related heuristics at different levels of the allocation process in order to achieve given set of constraints and objectives. Genetic algorithm is used to drive the final allocation distribution at the lowest level. Comparative study of heuristics at some levels of allocations is presented. Simulation experiments are based on data set obtained from one of the largest universities in Nigeria. Results obtained show the viability of applying heuristics efficiently to this new class of problem.

Keywords: Space Allocation Problem, Hostel Space Allocation Problem, Heuristic, Hierarchical heuristics, Metaheuristics, Genetic Algorithm.

1. Introduction

Space allocation problem (SAP[1]) is a combinatorial optimisation problem (COP) with some similarities to the classical knapsack problems [1, 2]. It is also related to some scheduling

[*] *Corresponding Author. Email: laremtj@gmail.com*

[1] SAP = Space Allocation Problem, HSAP = Hostel Space Allocation Problem, COP = Combinatorial Optimization Problem, DSA = Dean of Student Affairs, GA = Genetic Algorithm, OR = Operations Research

problems, for example academic timetabling [3]. SAP and the associated resource efficiency issues impact on different institutions ranging from companies, government to educational institutions [2]. It finds relevant application in various areas such as disk storage space allocation [4] and transhipment storage space allocation [5]. The current study focuses on SAP as it applied to academic institutions. The distribution of available room space among various entities (staff, students, lectures, laboratories, examination venues, etc.) in academic institutions is a dynamic process that is carried out on regular basis [6]. The *scarcity* of space makes it necessary to think of evolving an efficient allocation strategy for the limited available one. This kind of efficiency can be said to be achieved when all demanding entities are given the minimum required space possible while observing pre-stated constraints to a high degree of satisfaction. Burke et al. [7] thus advocated the use of appropriate optimization strategy incorporated in an automated system which is believed to save a lot of time and effort in space administration. As with other COPs, exact and heuristic methods can be used to provide solutions to SAP. Exact methods seek to solve problems to guaranteed optimality but execution on large real-world problems usually requires huge computational time. Heuristics methods however, find near-optimal quality solutions to such problems within reasonable computational time. Burke and Varley [6] therefore suggested the application of heuristics and metaheuristics to tackle some domain-specific SAP.

A heuristic is an algorithm that is able to produce an acceptable solution to a problem in many practical scenarios [8]. Metaheuristic [9] is a leading edge heuristics technique [10] that guides the exploration of other heuristics in search of good solutions to a general class of computational problem. It seeks to combine user-defined black-box procedures, usually heuristics, in a hopefully efficient way in search of optimal solutions [11]. Usually, heuristics are applied where there are no satisfactory domain-specific algorithms or where their application can be computationally expensive. In this paper, we apply a hierarchical heuristic strategy driven at the lower end by a genetic algorithm (GA) to find solutions to the hostel space allocation problem (HSAP).

The rest of the paper is organized thus: section 2 gives further overview of SAP with specific emphasis on the new case of HSAP as obtained in our case study. The heuristics used for the HSAP are presented in section 3. Results obtained for the current problem case are discussed in section 4 while we made some concluding notes in section 5.

2. Space Allocation Problems

SAPs are domain-specific decision problem which have attracted attention among researchers working on metaheuristics within the past few years [1,2,5]. SAP seeks to reach the best objective under some operational domain constraints. In the context of higher institution, it is defined as the allocation of entities (staff, students, meeting rooms, lectures, special rooms, etc.) to areas of room so as to satisfy as many requirements and constraints as possible [2, 6]. The problem is highly constrained, has multiple objectives, varies greatly among different institutions and requires frequent modifications due to the addition or removal of entities and/or rooms [7]. It also has direct impacts on the functionality of institutions. Unfortunately, like other difficult tasks in higher institutions, especially in developing countries, space allocation is carried out manually which in most cases take time to complete [6]. Considering domain specific cases of SAP, the best practice in many institutions in developing countries is reliance on a form of database or spreadsheet that keep record of entities and available spaces without any recourse to any form of algorithm to determine optimal space allocation [12].

Several domain-specific cases of SAP have been studied in literature including office space allocation [2, 13], shelf space allocation [1, 14], and storage spaces allocation for transhipment tasks [5]. Annevelink and Broekmeulen [15] studied the application of GA into space allocation planning in pot-plant nurseries within the field of horticulture. This was a pioneer work that seeks the use and incorporation of GA metaheuristic into decision support system for the complex-natured space allocation planning in the nurseries [15]. Similarly, Dean [16] investigated the use of GA to optimize the staff scheduling within the domain of nurse rostering. His work entails a case of assigning employees to time slots in a way that satisfies given constraints. Specifically, the GA adapted employed two different chromosome representations, namely bit string and two-dimensional array for a data set of nurse shift duty rostering within a four-week schedule. Simulation results obtained proved the efficiency of the two-dimensional array structure over the bit string [16]. Presently, we consider a new case of SAP namely, the HSAP with data set as obtained in a foremost tertiary institution in Nigeria. Aside the two initial heuristics used for category allocation, GA with a multi-dimensional chromosome structure is employed to drive the final solution to the problem at the lowest level. HSAP has not been well studied in literature especially with reference to the application of heuristics and metaheuristics and in the context of developing countries.

2.1 Hostel Space Allocation Problem

Hostel space is one of the most important resources that influence the choice of higher institutions. Availability and efficient management of hostel space will not only aid concentration and academic performance of students but also assist in maximizing the institution's throughput in terms of graduation rate. Hostel space allocation and management are therefore critical issues in university administration. Most *first generation* (in terms of relative date of establishment) universities in developing countries have on-campus residence for students that were built years ago when the average number admission intakes were small compared to the current influx. The upsurge in student population is such that majority of the students reside off-campus with the attendant adverse effects on performance and throughput especially for the brilliant ones. In Nigeria, most of the higher institutions have hostel space enough to accommodate less than twenty percent of the student population [17]. In South Africa, it was recently reported nationally [18] that residence accommodation for students has reached crisis levels with the attendance effect on success and graduation rates. A forum of some tertiary institutions in the country noted that the high rate of student failures and increasing first-year dropout rates are worsened by the lack of student accommodation [18]. The recent upsurge coupled with the declining rate of fund availability therefore makes it necessary for university administration to reconsider many issues including space planning, most especially hostel space for students. Meanwhile, given the fact that provision of new and sufficient hostels to meet the current demand in developing countries is a long-term developmental goal; the emphasis should then be on finding an optimal way of utilizing and allocating the available hostels. The allocation of available hostel space can thus be seen as a competitive process with strict guidelines set by various institutions to steer the process.

Continual increased in the number of eligible applicants for hostel space with little or no subsequent increase in facility calls for an efficient optimization approach. HSAP can therefore be considered as a decision problem that aim at finding the best possible allocation of scarce resources (bed space within hostels) among many competitive 'customers' (eligible students) under given hard and soft constraints. It can be considered as an extension of the knapsack [19] or bin parking problems [20]. As currently obtained in the case study, the allocation process is done out every academic session as a multi-stage process (see Fig. 1). More insight is provided in section 2.1.2.

2.1.1 Basic Terminology

We provide some basic definitions and notations related to the HSAP.

1. *Bed Space*: A space created in the hostel to accommodate a single student.
2. *Hall (Hostel) (H):* A building meant to accommodate a number of students located at various zones within the campus. Each hall has a pre-defined capacity in terms of bed space and consists of set of rooms located at various floors levels and blocks (also referred to as wings). The duo terms of hall and hostel will be used interchangeably in this paper. A general assumption is that halls are located at different part of the campus vicinity.
3. *Capacity*: the maximum number of students that can be accommodated in a given entity (e.g. hall, floor, room). This is calculated as the total number of bed spaces in that entity.
4. *Category (C):* A grouping into which eligible applicants are divided for hostel allocation purpose. Each category has varying number of students with varying degree of allocation priority.
5. *Allocation (A)*: The assignment of students in a given category into an allocation entity (hall, block, floor). For example at the hall level, $a_{ij} \in A$ refers to the number of students in category *i* allocated to hall *j*. At the Floor/block level, it refers to the number of students allocated to the floor/block.
6. *Room Capacity (R)*: Total number of bed spaces in a given room within a given hall.
7. *Block Capacity (B)*: Summation of *R* for a given block in a given hall.
8. *Floor Capacity (F)*: Summation of all *B* for a given floor in a given hall.
9. *Hall Capacity (H)*: The capacity of all *F* within a given hall.

2.1.2 Case Study

The allocation of bed space to various categories of students is performed by the student affairs unit at the beginning of the academic session in the case study. As at the time of study, there are twelve major hostels on the main campus for undergraduate students consisting of six male and six female hostels zoned according to their physical location (Tables 1 and 2) . The hostels have varying number of rooms spread across different floors and blocks. The number of floors and blocks also vary for different hostels (see Appendix A).

The allocation is carried out in an inter-related multi-stage manner (Fig. 1) under varying set of constraints and requirement considerations. This kind of arrangement made us to apply a sort of cooperative heuristics to handle different stages. The initial stage determines the

number of students to be considered for allocation in each category based on a set of given requirements and priority. The second stage takes the results of the category allocation and distributes them into various hostels based on another set of constraints. Finally, the generated hall allocation list is used to distribute students under each categories into various block/floor based on another set of constraints.

Table 1 – Hall Distribution with Capacity (Source DSA Office)

Zone (Area)	Hostel ID	Sex	Capacity
A (Main Campus)	HA1	Male	660
	HA2	Male	444
	HA3	Female	866
B (New Hall)	HB1	Male	800
	HB2	Female	764
	HB3	Female	276
	HB4	Female	524
	HB5	Male	968
C (Gate/Education)	HC1	Male	526
	HC2	Female	512
	HC3	Female	646
	HC4	Male	512

Table 2 – Summary of Hall Facilities

	ZONE			Total
	A	B	C	
Female	866	1 564	1 158	**3 588**
Male	1 104	1 768	1 038	**3 910**
Total	**1 970**	**3 332**	**2 196**	**7 498**

For hostel allocation purpose, students are grouped into eight different categories with different allocation priority. In the order of priority, we have: 1) Final year students (Fy), 2) Scholars - those with cumulative grade point average (CGPA) $\geq$ 4.20 of 5.00 scaling (Sc), 3) Foreign students (Fo), 4) Health students - physically challenged students or those with peculiar health conditions (Ht), 5) Fresher - first year and direct entry students (Fr), 6) Sports men and women (Sp), 7) Discretionary - based on individual special request (Ds), and 8) Others - students of other levels or years (Ot). Priority is assigned based on certain academic and/or administrative considerations by the institution authority. For example, the need for Fy to be accommodated in order to have full concentration on their studies and project work; and the allocation of all Fo category as they might not have a relative within the country. An optimal category allocation therefore should have a descending allocation curve in the given order of priority.

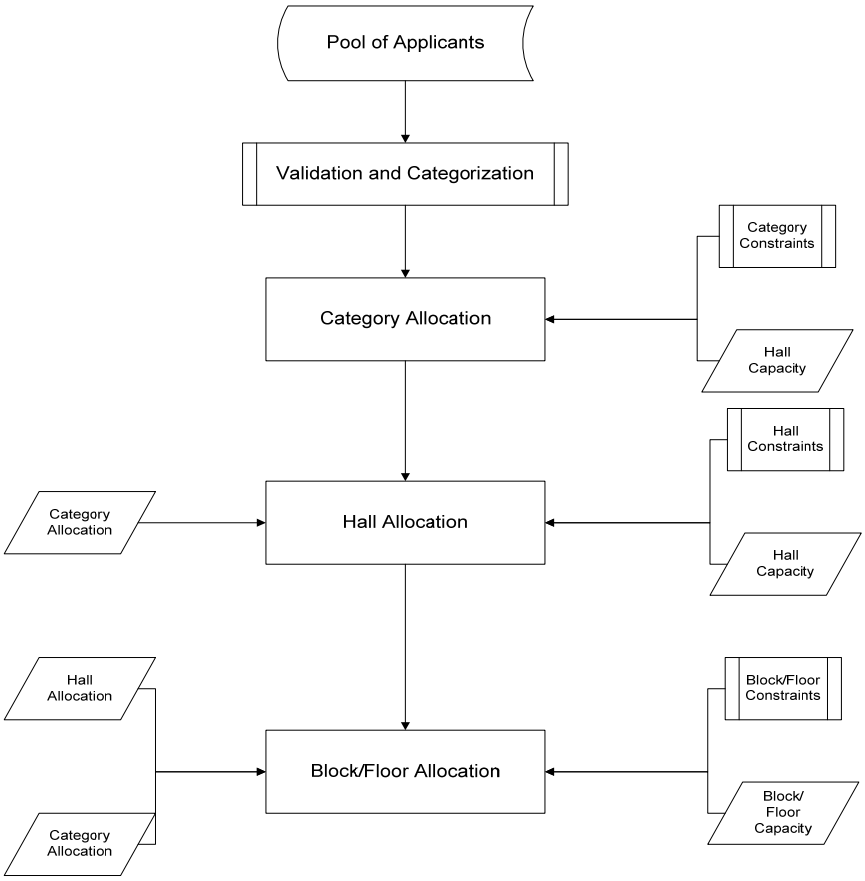

Fig. 1: Basic Conceptual view of the HSAP

2.1.3 Constraints

As stated earlier, HSAP has constraints imposed by system requirements which are to be satisfied in order to have an optimal allocation distribution and to achieve the overall goals of the process. We have classified these constraints into hard and soft. Hard constraints are to be rigidly enforced during the allocation process thus they must be satisfied. In the context of this work, feasible solutions are regarded as those that do not violate any of the hard constraints. On the other hand, soft constraints are imposed requirements which the system seeks to satisfy to a large extent, if not fully. They can be slacked with appropriate penalty when necessary. Table 3 provides a set of hard and soft constraints for each level of allocation. Furthermore, a major hard constraint is the allocations of some categories into pre-specified halls (see Table 4). This requirement was introduced by the institution authority because of certain factors. One factor is proximity to facilities that might be needed by certain categories, for example Ht category are to be allocated to hostels close to campus health centre while Sp are to be accommodated close to the sport centre. Another factor is the need to all eligible applicants in some categories (like Fo) to be accommodated given their peculiar

characteristics. It should be noted that though the allocation of Fy category is considered a soft constraint in terms of number of student accommodated yet its takes pre-eminence over other allocations (with the exception of Ht, Sp and Fo) because of the assigned allocation priority, that is 1) Fy, 2) Sc, 3) Fr, 4) Ds and 5) Ot, in this order.

Table 3: Summary of Constraints/Requirements

Level	**Constraints**		**Classification**
Category Allocation	a.	All students in Fo, Ht and Sp categories must be allocated.	Hard
	b.	As many Fy, Sc, Fr, Ds, and Ot as possible should be allocated in this order of priority.	Soft
Hall Allocation	a.	Students in Ht, Sc and Sp must be allocated to designated hostels (see Table 4)	Hard
	b	Allocation for the remaining categories must be in the stated order of priority	Soft
Block/Floor Allocation	a.	Ht category should be allocated to the lowest floor possible in their assigned hall	Soft
	b.	Fy category should be allocated to the highest possible floor in the hall	Soft

Table 4 – Specified Halls for certain Categories

CATEGORY	SPECIFIED HALLS	
	MALE	FEMALE
Ht	HA1	HA3
Sc	HA2	HA3
Sp	HC1	HC3

3.0 Methodology

Given the multi-level nature of the allocation process and the set of constraints in Table 3, we propose a hierarchical heuristics to handle the problems (Fig. 2). Different heuristics handles the allocation at each stage but the result of a succeeding stage depends on that of the previous stage(s). Two distinct heuristics are designed for each of the first two stages for study purposes. The final distribution into floors is then handled by GA. In the case study, the allocations of male and female students are considered mutually exclusive and are thus treated in our study. Our aim is to find an optimal allocation and distribution of students into hostels. This implies optimizing bed space utilization while satisfying specified constraints and requirements. Two classes of allocation choices are identified at some stages of the allocation process, namely the *fixed-choice* and *free-choice* allocations. At the category level, the fixed-choice allocations of all students in Ht, Fo and Sp categories must be achieved first before the *free-choice* allocation of the remaining categories according to assigned order of

priority. At the hall level, allocation of students in Ht, Sp and Sc is considered *fixed-choice* while the remaining categories allocation is *free-choice*.

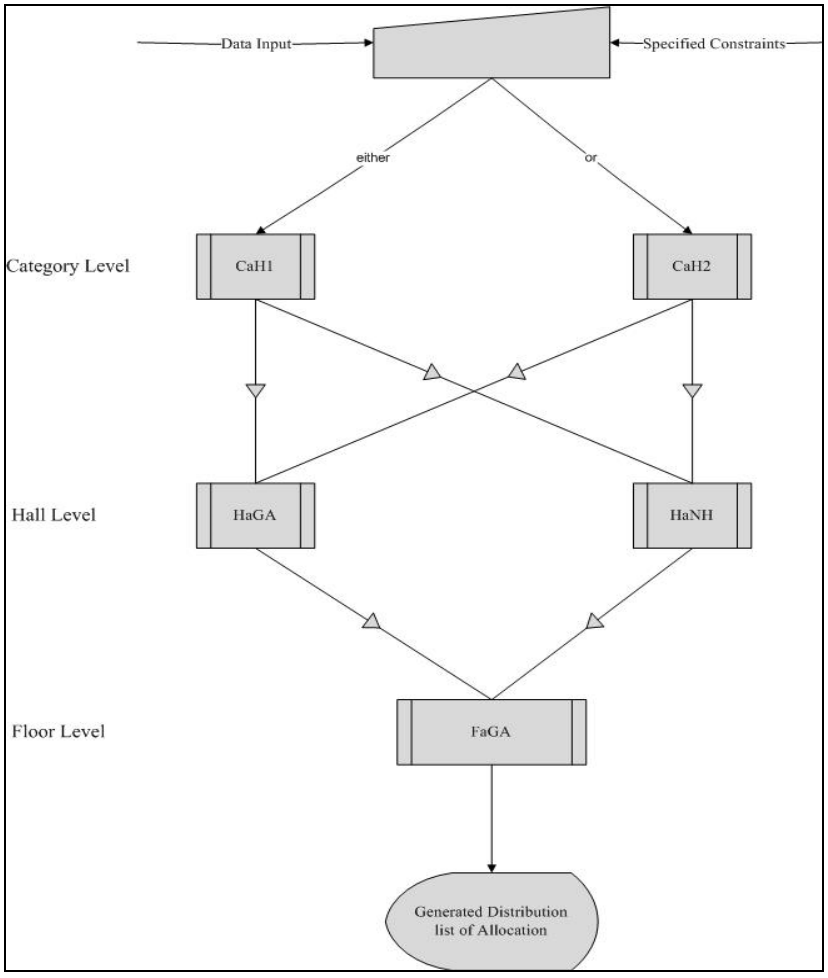

Fig. 2: The structure of the hierarchical heuristics

With respect to Fig. 2, necessary input such as the capacities for halls, number of floor for each hall, and number of applicants for each category are supplied to the system together with necessary specified constraints. The category level is implemented via *CaH1* or *CaH2* heuristics. Results obtained can then be processed at the hall level via either *HaGA* or *HaNH*. Finally, the floor allocation is driven by GA via *FaGA* to give the distribution of students into various wings and floors within each hall of residence. The descriptions of *CaH1*, *CaH2* and *HaNH* are provided in sections 3.1 to 3.3. The *HaGA* and *FaGA* presented in Fig. 2 are both GA metaheuristics with similar structures used at the hall and floor allocations respectively. Details of the structure and operation of the GAs are provided in section 3.5.

3.1 *CaH1* Heuristic

The applicants are divided into the eight given categories stated in section 2.1.2. The eight categories are grouped into the *fixed-choice* allocation consisting of Ht, Fo and Sp and the *free-choice* allocation consisting of Fy, Sc, Fr, Ds and Ot, in order of allocation priority. *CaH1* employs a greedy-like approach that first ensures that the *fixed-choice* categories are fully allocated while observing the hall capacity constraint. Initially, the allocation for each category is set to zero and the total number of applicants and the total capacity for all halls combined, *TH*, are noted. A variable, *TFc*, is set to keep track of the total number of applicants allocated so far. The category allocation (C_1, C_2, and C_3) for the *fixed-choice* is set to the number of applicants for each respective category and the variable, *TFc*, is set to the total number of applicants for all *fixed-choice* allocations. The total capacity of space remaining in all the halls, *rem*, is thus the difference between *TH* and *TFc*.

The heuristic then seeks to allocate the remaining five *free-choice* categories while observing the assigned allocation priority and the remaining capacity constraint for the halls. The categories are first sorted in order of allocation priority such that $C_4 > C_5 > \ldots > C_8$. A Boolean variable is used to detect and prevent negative allocation into any of the category. The heuristic randomly generate a number, *r*, between 0 and remaining hall capacity. And set each successive category (i.e. $C_4, C_5, \ldots, C_8$), as the minimum between *r* and total number of applicants for that category. For analysis purpose, we compute the number of unallocated applicants for each category as the difference between the number of applicants and the number allocated for the category.

The pseudo-code for *CaH1* is given below:

i. **Initialize:** Set the total capacity of all Halls to *TH*, $C_i = 0$ for all categories, *Appls[i]* = Total number of applicants for category *i*

ii. **Allocate Fixed Choice:** Set $C_i = Appls[i]$, $i = 1,2,3$
Sum up the *students* in Fo, Ht and Sp given *TFc* and subtract from *TH*.

iii. **Allocate Free Choice:**
Initial: Set *rem* = *TH* - *TFc* and Bool *Ok* ← *FALSE*;
Prioritize: Set free choice categories C_i, $i = 4,..,8$ in order of priority such that $C_4 > C_5 > \ldots > C_8$.
while *(NOT Ok)*
rem = *TH* – *TFc*
int *remNew* = *rem*;
Allocate: Set C_i = **Min{***remNew, Appls[i]***}**,
Set *remNew* = *remNew* - C_i, $i = 4,..,8$ in order of priority
CheckOk()
End while

iv. **Calculate Unallocated**: *Unallocated[i]* = *Appls[i]* - C_i, for $i = 4$ to 8

v. **CheckOk**: If $C_i >= 0$,for all i = 4,..,8 Set Ok = TRUE

We present a simple example below of the *free-choice* allocation of *CaH1* for illustration purpose. If we assume five categories, say, C4, C5, C6, C7, and C8 in that order of priority to have 300, 450, 600, 100 and 230 applicants respectively and that the remaining capacity of the hall, *rem,* after fixed-choice allocation is 1000. Table 5 below gives the results of a sample run of *CaH1* for this data set. All categories thus have a fraction of the number of applicants allocated based on the assigned allocation priority and the total capacity of the remaining hall space is observed.

Table 5: Sample Illustration for the *CaH1* heuristic

Category	*Appl[i]*	*rem*	*Min [Appli[i], rem]*	C_i
		1000		
C4	300	700	300	300
C5	450	250	450	450
C6	600	0	250	250
C7	100	0	0	0
C8	230	0	0	0
Total ==> 1680			**Sum ==> 1000**	

3.2 *CaH2* Heuristic

Similar to *CaH1*, applicants are divided into both *fixed-choice* and *free-choice* with the allocation of the fixed choice categories proceeding in the same manner. *CaH2* then allocates the remaining five *free-choice* categories based on the assigned allocation priority and the remaining capacity constraint for the halls. However, instead of a greedy-like approach of *CaH1*, *CaH2* employs a percentage range allocation distribution strategy which is set based on the allocation priority. The goal is to ensure that there is at least some allocation for each category of the *free-choice* assigned to some hall of residence.

The *free-choice* allocation categories are first sorted in order of allocation priority such that $C_4 > C_5 > ... > C_8$. A percentage range in $[Minp_i, Maxp_i]$, i=4,..,8, is assigned to all the categories such that the maximum percentage for the highest priority category is less than 100 and the minimum percentage for a higher category is greater than or equal to the maximum percentage for the subsequent category. Having assigned the percentage range for all categories, an allocation range $[MinA_i, MaxA_i]$, i=4,..,8, is computed for each category. A randomly number, x_i in $[MinA_i, MaxA_i]$, i=4,..,8, is generated and summed up to give, *sum,* for all the free-choice categories. This number x_i determines the number of applicants to allocate for the ith category. However, to prevent *space overuse* (section 3.4), we need to

balance up this sum with the remaining hall capacity constraint, *rem*. This is done by allocating exactly only *rem/sum* fraction of x_i for each category C_i, i=4,..,8. We assign

$$\omega_i = \frac{rem}{sum} x_i ,$$

where

$$sum = \sum_{4}^{8} x_i .$$

Clearly,

$$\sum_{4}^{8} \omega_i = rem$$

Note that even though x_i is random, making it possible for *sum* to have different values for different execution of the algorithm, the balancing we did ensures that the total number of students allocated is exactly the total number of space left to be filled in all the halls. Thus, the random nature of x_i will not have any negative effect on the GA used at the floor allocation level.

Similar to *CaH1*, the number of unallocated applicants for each category computed as the difference between the number of applicants and the number allocated for the category.

The pseudo-code for the *CaH2* is now given:

i.	**Initialize:** Set the total capacity of all Halls to *TH*, C_i = 0 for all categories, *Appls[i]* = Total number of applicants for category *i*
ii.	**Allocate Fixed Choice:** Set C_i = *Appls[i]*, i = 1,2,3 Sum up the students in Fo, Ht and Sp given *TFc* and subtract from *TH*
iii.	**Allocate Free Choice:** **Initial:** Set variable *sum* = 0; Set *rem* = *TH* - *TFc* **Prioritize:** Set free choice categories C_i, i = 4,..,8 in order of priority such that $C_4 > C_5 > ... > C_8$. **Assign Percent Range:** For each C_i, assign a percentage range [$Maxp_i$, $Minp_i$] such that $Maxp_4 < 100\%$ and $Minp_i >= Maxp_{i+1}$. **Calculate:** Calculate Allocation Range [$MaxA_i$, $MinA_i$] based on [$Maxp_i$, $Minp_i$] **Select:** For each C_i, randomly generate a number x_i in [$MaxA_i$, $MinA_i$] and add to *sum* **Balance:** For each generated random x_i, calculate new $\omega_i = x_i$ * *rem/sum*
iv.	**Allocate**: For each C_i, allocate exactly the new x_i, that is, $C_i = x_i$.
v.	**Calculate Unallocated**: *Unallocated[i]* = *Appls[i]* - C_i, for i = 4 to 8

We present a simple example below of the free-choice allocation of *CaH2* for illustration purpose. If we assume five categories, say, C4, C5, C6, C7, and C8 in that order of priority to have 300, 450, 600, 100 and 230 applicants respectively and that the remaining capacity of the hall, *rem,* after fixed-choice allocation is 1000. Table 6 below gives the results of a sample run of *CaH2* for this data set. All categories thus have a fraction of the number of

applicants allocated based on the assigned allocation priority and the total capacity of the remaining hall space is observed. Result obtained using the *CaH1* and *CaH2* are presented and compared in section 4.

Table 6: Sample Illustration for the *CaH2* heuristic

Category	Appl[i]	Percentage Range [*Minpi, Maxpi*]	Allocation Range [*MinAi, MaxAi*]	Random x_i in [*MinAi,MaxAi*]	$x_i \rightarrow \omega_i$	C_i
C4	300	90% - 95%	[270, 285]	278	213	213
C5	450	80% - 88%	[360, 396]	395	302	302
C6	600	70% - 80%	[420, 480]	436	333	333
C7	100	60% - 70%	[60, 70]	66	50	50
C8	230	50% - 60%	[115, 138]	134	102	102
Total → 1680			***sum* ==> 1309**		**Total Allocation ==> 1000**	

3.3 *HaNH* Heuristic

At the hall level, the *HaNH* (see Fig. 2) aims at optimizing the spread of allocated category of students into available hostels. It starts by first allocating the *fixed-choice* categories. The heuristic then seeks to distribute the *free-choice* categories such that there will be maximal spread across all the hostels that are yet to be filled to capacity. The process is described as follows:

The category allocation obtained from either *CaH1* or *CaH2* is taken as input with the hall capacity constraints. The eight categories are divided into *fixed-choice* categories of Ht, Sp and Sc and the *free-choice* categories of Fy, Fo, Fr, Ds, and Ot. No order of priority is considered at this stage. The total capacity of the halls is computed as *TH*. The first part of the heuristic allocates the *fixed-choice* categories into designated halls, say, H_a, H_b, and H_c and calculates the total allocations as *TFc*. This is done by assigning number of students in each of these three categories to the respective designated hall. The capacity for each of H_a, H_b and H_c is thereafter decreased by the allocations for the category allocated to it respectively. We then set the space left to be allocated in each hall as H_j, j=1,…,*n*, where *n* is the number of halls. The total space left to be filled is thus:

$$THNew = \sum_{1}^{n} H_j = TH - TFc.$$

In order to ensure optimal spread in the distribution of the *free-choice* categories into available halls of residence, we compute a real-value hall ratio to drive the distribution. The hall ratio, HR_j, $j=1,...,n$, is computed as the ratio of the capacity of a hall to the total number space left to be allocated for all the halls (*THNew*), *i.e.* $HR_j = H_j/THNew$. Therefore, for each category in the *free-choice* and for each hall with a space left to be filled, the hall allocation for category i in hall j, $a_{i,j}$ is computed as the product of the hall ratio, HR_j, and the number of students in category i, C_i. The pseudo-code for the *HaNH* is given as:

i.	**Input:** *Ci, i=1..m* from Category allocation, Hall capacities for all hostels
ii.	**Initialize:** Set the total capacity of all Halls to *TH*.
iii.	**Allocate Fixed Choice:** Sum up the *students* in Ht, Sc and Sp given *TFc* Assign C_{Ht}, C_{Sc} and C_{Sp} to designated Halls and compute space left for the halls Let *Hj*, *j=1..n* be the space left to be fill for all hostels Calculate total space left *THNew = TH - TFc*
iv.	**Allocate Free Choice:** **Hall Ratio:** Compute HR_j, $j = 1..n$ as *HRj = Hj/THNew* **Reorder:** Set free choice categories C_i, $i = 1...5$ (in any order) **Allocate:** For each category in the Free choice For each hall, H_j, $j=1..n$ **Assign** $a_{i,j} = HR_j * C_i$

The heuristics discussed above together with the GA components are implemented for experimental study and comparative purpose. As shown in Fig. 2, there are four implementation options (paths) available at the first two stages. A succeeding stage however depends on the results obtained from its immediate predecessor. Thus GA at the floor level serves not only to distribute students into floor and wings but also to ensure that the results from hall allocation are well managed. This arrangement makes for progressive improvement in the achievement of the overall requirements of the entire allocation process.

3.4 Solution Quality

Some of the result obtained from simulation experiment are presented and discussed in section 4. In the real-life case, there are no well-defined criteria for measuring the quality of distribution obtained for hostel allocation. Coincidentally, there are no properly kept archives records that can be used as benchmark for our study since most of the allocation are carried out manually in a piece-meal manner. As part of a pioneer study in heuristic and metaheuristic application to HSAP therefore, we measure the quality of solutions obtained in our study based on the satisfaction of given constraints and requirements. The quality of an allocation (solution) at any stage in HSAP is thus measured in terms of 1) non-violation of given hard constraints, 2) number of students allocated for each category, 3) space utilization

factor, and 4) satisfaction of any additional requirements (soft constraints). We define *space utilization factor* (*u*) as a measure of bed space usage. This is taken as the average of the allocated space (*A*) over the available space when considering a given entity (e.g. hall). Thus

$$\text{Utilization factor } (u) = \frac{\text{Allocated Space } (A)}{\text{Capacity}} \qquad (1)$$

An allocation is expected to have $0 \leq u \leq 1$. A value of $u > 1$ would imply *space overuse* which is prevented by the GA through a repair algorithm. The utilization factor is appropriately indexed when used at different levels of the allocation process.

Given the application of heuristic, an optimal solution would be one in which all entities are allocated and the space utilization is the best possible, i.e. the amount of space wasted and overused has been reduced to the minimum and the additional requirements and constraints have been all satisfied [2].

3.5 Genetic Algorithm (*HaGA, FaGA*)

GA [21] is a probabilistic search algorithm that iteratively transforms a population set of mathematical objects, each having an associated fitness value, into a new population of offspring objects based on the Darwinian principle of natural selection and use of operations that are patterned after naturally occurring genetic operations of crossover and mutation [22]. GA metaheuristic seeks to evolve towards an optimum (global best) solution to a given problem. Possible solutions (*individuals*) are set of values represented as chromosomes. Set of solutions (population) are generated from an initial population of solution and these evolve until terminating conditions are met, for example, the optimal solution is found or a pre-defined maximum number of iterations (generations) has been performed. Evolution of generations involves selection of individuals (otherwise known as parents) to breed based on a pre-defined fitness function and production of offspring through the applications of genetic operators of crossover and mutation. More description on the nature and application of GA can be found in [21, 23, 24, 25]. We apply GA at both hall allocation (as an alternative heuristic) and floor level (as the final heuristic). Fig. 3 gives an overview of the GA.

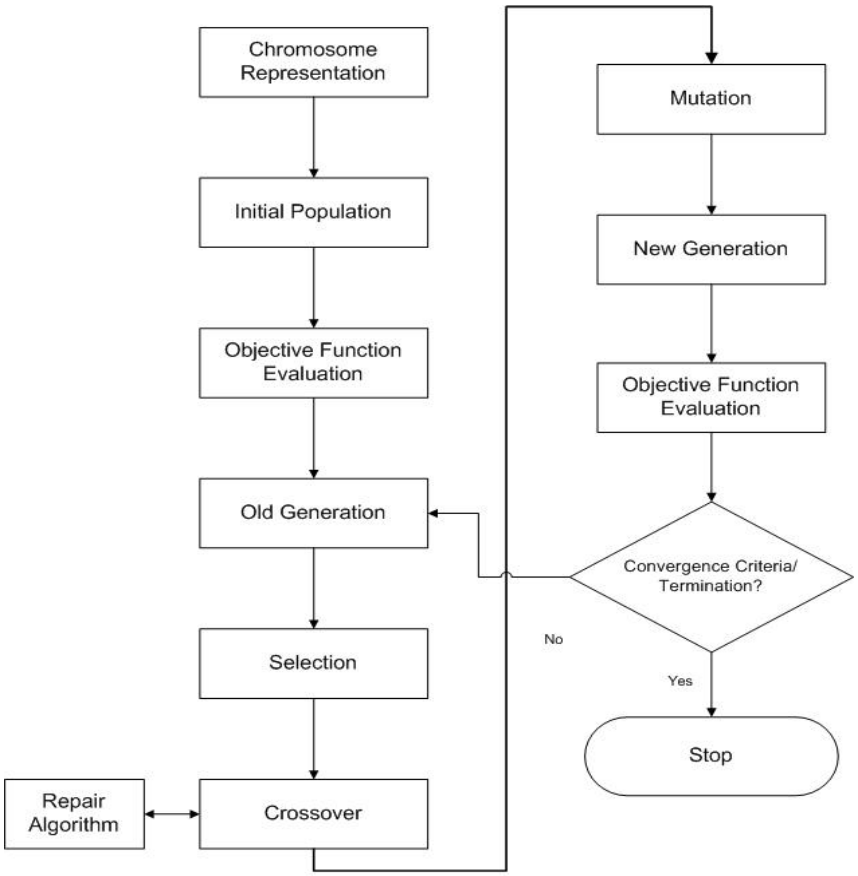

Figure 3: Graphical Illustration of the GA Steps

3.5.1 Chromosome representation

We design similar but different chromosome representations based on a multi-dimensional array structure for both hall and block/floor allocations (Figs. 4&5). This type of representation has proved to be effective in handling similar problems with large data set and constraints [16].

At the hall level, the chromosome has the structure (c_1 (a_{11} ... a_{1n}), c_2 (a_{21} ... a_{2n}) ... c_m (a_{m1} ... a_{mn})), where c_i represents category i and a_{ij} represents the allocation of category i in hall j, m is the number of categories and n is the number of halls (Fig. 4). The population for the hall allocation thus consists of Ψ individuals, where ψ is the population size. This is represented as a 3-dimensional array of integers for simulation purpose. An integer in the array with subscripts p, i, *and* j indicates the number of students in category i allocated to hall j in the solution represented by the individual p.

The chromosome at the block/floor level, has the structure: (c_1 (b_{11} (a_{111} ... a_{11x}) ... b_{1w} (a_{1w1} ... a_{1wx})) ... c_m (b_{m1} (a_{m11} ... a_{m1y}) ... b_{mw} (a_{mw1} ... a_{mwy}))) where c_i represents category i, b_{il} represents block l in category i's allocation, a_{ilk} represents category i's allocation on floor k of block l, m is the number of categories w is the number of blocks and x and y represent the

varying number of floors in the respective blocks. Fig. 5 presents four sections (one for each block), each with 8 rows representing the eight categories and the blocks having 3, 1, 2 and 4 floors respectively. The population consists of Ψ individuals. For implementation purpose, the population is represented as a 4-dimensional array of integers with subscripts *p, i, l* and *k* representing the number of students in category *i* allocated to floor *k* in block *l* in the solution of individual *p*.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}
a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}
a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}

Fig 4: A chromosome representation for hall allocation with m = 8 and n = 6.

a_{111}	a_{112}	a_{113}	a_{121}	a_{131}	a_{132}	a_{141}	a_{142}	a_{143}	a_{144}
a_{211}	a_{212}	a_{213}	a_{221}	a_{231}	a_{232}	a_{241}	a_{242}	a_{243}	a_{244}
a_{311}	a_{312}	a_{313}	a_{321}	a_{331}	a_{332}	a_{341}	a_{342}	a_{343}	a_{344}
a_{411}	a_{412}	a_{413}	a_{421}	a_{431}	a_{432}	a_{441}	a_{442}	a_{443}	a_{444}
a_{511}	a_{512}	a_{513}	a_{521}	a_{531}	a_{532}	a_{541}	a_{542}	a_{543}	a_{544}
a_{611}	a_{612}	a_{613}	a_{621}	a_{631}	a_{632}	a_{641}	a_{642}	a_{643}	a_{644}
a_{711}	a_{712}	a_{713}	a_{721}	a_{731}	a_{732}	a_{741}	a_{742}	a_{743}	a_{744}
a_{811}	a_{812}	a_{813}	a_{821}	a_{831}	a_{832}	a_{841}	a_{842}	a_{843}	a_{844}

Fig. 5: A chromosome representation for block and floor allocation with m = 8, and w = 4.

3.5.2 Fitness Evaluation

Given the diversity in the criteria imposed on and by various categories in the study, it apparently become difficult to design an evaluation function that incorporates all the criteria with adequate weighting. However, the possibility of breaking down the constraints into the various levels makes the goal achievable. The fitness of an individual in a given population is computed as a measure of the degree of satisfaction of given constraints that influence the allocation. To this end, we assigned weights accordingly to the various constraints at both the hall and block/floor levels. This guides the algorithm during the allocation process to ensure satisfaction of relevant constraints. The fitness evaluation function and the weights are combined to give a set of real number fitness values in [0, 1]. A fitness value of 0 indicates non-satisfaction of given all constraints for the allocation; while a value of 1 indicates total satisfaction of constraints. These fitness values are thus determined using the combination of the space utilization factor and assigned weight for respective constraints.

Based on the constraints stated in Tables 3 and 4, the fitness of an individual in a population at the hall level is computed as:

$$f = \sum_q w_q u_q \in [0,1]$$

where u is the utilization factor (eq. 1) at the hall level given as:

$$u_q = \frac{a_{ij}}{s_i},$$

w_q = weight assigned to constraint q, $q = 1, 2, 3$ representing constraints presented in Table 4 for the three *fixed-allocation* categories, a_{ij} = number of allocations of category i to hall j, s_i = total number of students in category i where i,j are fixed (Table 4). The computation is repeated for all individuals in the population and the computed values used to determine the average fitness for the generation. We take note of the best fitness values over the generations. The three constraints at this level are assigned weights as follows: $w_1 – 0.4$; $w_2 – 0.3$; $w_3 – 0.3$, where the subscript 1 represented Ht category, subscript 2 represents Sc and subscript 3 represents Sp.

Considering the two constraints at the block/floor level (Table 3), the fitness of an individual is computed thus:

$$f = \sum_q w_q u_q \in [0,1]$$

where u is the utilization factor (eq. 1) at the floor allocation level given as:

$$u_q = \frac{T_i}{s_{ij}},$$

and

$$T_i = \sum_{k=1}^{\rho} a_{ik},$$

w_q is the weight assigned to constraint q, q = 1,2 representing floor level soft constraints (highest and lowest floor possible) as presented in Table 3, a_{ik} represents the allocation of

category i to floor k, ρ = an integer representing the highest (or lowest) floor and s_{ij} = total number of students in category i allocated to hall j.

There are only two major constraints that affect allocation at this level, namely, for Ht (allocation to the lowest floors possible in hall) and Fy (allocation to the highest possible floors in the halls). We assign weight as follows: where the hall has an Ht allocation and no Fy allocation: $w_1 - 1.0;\ w_2 - 0.0$ with subscript 1 representing Ht category and subscript 2 representing the Fy category. Where the hall has Fy allocation and no Ht (typically hall is not a specified hall for Ht): $w_1 - 0.0;\ w_2 - 1.0$; and where the hall has a both Ht and Fy allocations, $w_1 - 0.7;\ w_2 - 0.3$. Thus Ht students are given higher priority because of their peculiar health status and thus more stringent hard constraints associated with the category at all levels (Table 3).

3.5.3 GA Operations

We apply GA in a similar but mutually exclusive fashion at the hall and floor levels. At both levels, a repair algorithm is incorporated to prevent space overuse. The pseudo-algorithm of the GA structure is depicted as follows:

```
i.    Initialize: Create initial population, NewPopulation, random
ii.   Evaluate: Calculate_Fitness (NewPopulation)
iii.  Set: Set Current Population = Initial Population
iv.   While NOT (Terminal conditions)
v.        For counter = 0 to PopulationSize do
vi.           Selection:
                 Parent1 = Heuristic_Select (Current Population)
                 Parent2 = Heuristic_Select (Current Population)
vii.          Crossover:
                 Heuristic_Cross (Parent1, Parent2, NewPopulation)
viii.         Repair:
                 Heuristic_Repair (NewPopulation)
ix.           Mutation:
                 Mutate_Population (NewPopulation)
x.            Evaluate: Calculate_Fitness (NewPopulation)
xi.           Replace:
                 Replace_Population (Current Population, New Population)
xii.  endwhile
xiii. Display Output
```

Initial population of solutions is generated stochastically for both *HaGA* and *FaGA* while observing stated constraints. At the hall level, a randomly generated number of students

between 1 and allocation for the current category is generated and allocated into a randomly generated hall. The resulting population is evaluated, and if found okay, set as the current population. During each successive epoch, a proportion of the existing population is selected to breed a new generation of population using the roulette wheel selection. We have established in an earlier study [26, 27] that roulette wheel gives good results for the case under study in comparison with other selection strategies. Individual solutions are selected through a fitness-based process, with better solutions are typically more likely to be selected. The algorithm however allows small proportion of less fit solutions to be selected sometimes. This helps to keep the diversity of the population large and prevents premature convergence on poor solutions. The relative fitness of each individual in a population is computed by dividing the cumulative fitness of the individual by the total fitness of the population. The first individual whose cumulative fitness is greater than a randomly generated number, $r \varepsilon [0, 1]$ is selected for recombination. For each category, a single-point crossover strategy is used to generate a new offspring from selected parents based on a pre-defined crossover rate and a randomly generated number, r in [0,1]. If r is less than 0.5, the first offspring inherits the allocation for the category from the first parent while the second offspring inherits from the second parent otherwise the reverse is performed. The single-point crossover has also been compared with double-point crossover in an earlier study [26, 27] and proved to be effective. The repair algorithm is invoked to ensure that generated chromosomes do not exceed the capacity constraint at the level under consideration. A similar process is carried out for the floor allocation while substitution floors for halls appropriately.

The repair algorithm, for example at the hall level, ensures that for each hall, the allocation for all categories does not exceed the capacity of the hall. A boolean variable is set to track all possible overflows. If there is an overflow for any, the difference between the total allocation and the capacity is computed and redistributed into other halls stochastically. To achieve the redistribution, a category with non-zero allocation as well as a hall with non-zero allocation that has no space overuse is selected at random. We then compute the minimum, $m1$, between the overflow to be redistributed and the space remaining in the selected hall. Furthermore, the minimum, $m2$, between $m1$ and total allocation for the hall with overflow is computed. A random number, $r \varepsilon [1, m2]$ is generated. The allocation for the hall with overflow is then decreased by r while that of the selected hall is increased by r. This is repeated until all capacity constraints are satisfied.

Mutation was carried out based on pre-defined mutation rate and a randomly generated variable. At the hall level, two halls and two allocated categories are selected randomly. A

random number $r \varepsilon [1, m3]$ is generated where *m3* is the smaller between the allocations for the first category in the first hall and the second category in the second hall. The allocation of the first category in the first hall is them increased by *r* while that of the second category in the same hall is decreased by *r*. The reverse is performed for the second hall to ensure that the hall capacities are not exceeded. The same mutation process is used for the block/ floor allocation with floors substituted for halls.

Both *HaGA* and *FaGA* are terminated if 1) an optimal solution is found ((i.e. one with $\Sigma w_q u_q = 1$ for all the entities considered), 2) for fifty consecutive generations, the average fitness of the current population is at least 95% that of the previous average fitness and 80% of the best fitness found throughout execution, 3) execution has been carried out over a specified number of generation without (1) or (2) being satisfied.

4.0 Experimental Results

We present some of the results obtained from different experiments. All experiments are based on data set obtained from the case study which is presented in Appendix A. Simulation was done using an open-source eclipse® integrated development environment (IDE) for Java developer, release 3.3 on Microsoft Windows Vista Business operating system platform running on an Intel core 2 CPU at 1.66GigaHertz with 1Gigabyte of RAM.

First we compare the results for the two different heuristics at both category and hall levels. Figs. 6 & 7 are the distribution obtained at the category level using *CaH1* and *CaH2* respectively for male distribution. Recall that all students in Ht, Ot and Sp are to be accommodated (*u = 1)* while the allocation of other categories are to decrease with the given order of priority. While both heuristics give good results with the data set, we however observe that *CaH2* works better to achieve the overall goal of allocation prioritization. This is probably due to the unbiased strategy adopted by *CaH2* heuristic (see section 3.2 and Table 6). While the heuristic seeks to maximally enforce the allocation prioritization for the *free-choice* categories, it also ensures that the distribution is not biased either to the category with the highest priority or the category with the highest number of students to be allocated. The heuristic thus achieves dual goals of prioritization and unbiased distribution. Should there be a major increase in the number of students compared with available hall space, then *CaH1* may tend to achieve a more bias allocation than *CaH2*.

For the hall level heuristics, we compare results obtained by *HaGA* with that of *HaNH* after the application of *CaH1* at the category level. The results are presented in Tables 7 and 8 respectively where the total in the last row and the last column represents the total number of allocated students (equal to the total number of space available in all the halls). A careful observation of the two tables shows compliance with the hall level constraints. Both *HaGA* and *HaNH* satisfy the hard constraints for the *fixed-choice* allocation. However, *HaNH* heuristic gives better maximum distribution spread of the *free- choice* categories into the halls than the *HaGA* as shown by the distribution in bold typeface. For example, with *HaGA*, Fr category are concentrated more in HB5 and HC4 halls but are better spread out across the halls by *NaNH*. Note that the distribution achieve by both heuristics depends on the capacity constraint of the halls.

Table 7: Hall Distribution for *HaGA*

	HA1	HA2	HB1	HB5	HC1	HC4	Total
Fo	**2**	**6**	**4**	**0**	**0**	**8**	20
Ht	70	0	0	0	0	0	70
Sp	0	0	0	0	400	0	400
Fy	**531**	**6**	**473**	**139**	**17**	**74**	1240
Fr	**31**	**13**	**171**	**692**	**83**	**342**	1332
Sc	0	400	0	0	0	0	400
Ds	**8**	**14**	**61**	**0**	**16**	**1**	100
Ot	**18**	**5**	**91**	**137**	**10**	**87**	348
Total	660	444	800	968	526	512	**3910**

Table 8: Hall Distribution for *HaNH*

	HA1	HA2	HB1	HB5	HC1	HC4	Total
Fo	**4**	**0**	**5**	**6**	**1**	**3**	20
Ht	70	0	0	0	0	0	70
Sp	0	0	0	0	400	0	400
Fy	**241**	**18**	**326**	**395**	**51**	**209**	1240
Fr	**259**	**19**	**351**	**424**	**55**	**224**	1332
Sc	0	400	0	0	0	0	400
Ds	**19**	**1**	**26**	**32**	**4**	**17**	100
Ot	**68**	**5**	**92**	**111**	**14**	**59**	348
Total	660	444	800	968	526	512	**3910**

According to Petrovski et al [11], GAs possess explorative features, characterised by the population size and rate of mutation, and exploitative features characterised by the crossover probability. Mere use of these features does not guarantee efficiency of GA without

determining the domain specific values of these features that guarantee acceptable result. It is necessary therefore to strike a balance between the explorative and exploitative features of GAs. These features are quantified by the mutation rate; crossover rate, etc [11]. The effects of varying these factors on the efficiency of GAs are different. In our simulation experiment, we develop a methodology which allows identifying GA factors that most significantly affect the performance of the algorithm and we try to get the optimal values of such factors empirically.

Simulation experiments were conducted using the same data set for the case study (see Appendix A). These are supplied by the user at the start of the experiment. The data sets include hall capacity, floor capacity and the number of eligible students in each category.

We conducted series of experiments to find combinations of parameters, that is *popsize* (population size), *pr_cross* (crossover rate) and *pr_mut* (mutation rate), which consistently give good results when the algorithm is executed. The experiment also aimed at studying the effect of changes in these parameters on the algorithm performance. Generally, the range of values of the parameters were varies in 10 equal intervals, for example: *popsize* [10,100], *pr_cross* [0.1,1.0] and *pr_mut* [0.0,0.09]. In another experiment, we also varied the *pr_mut* in 10 equal intervals in the range [0.1,0.9].

First, we allowed the values of the parameters to varies in 10 successive interval in the order *popsize* [10,100], *pr_mut* [0.1,0.9], and *pr_cross* [0.1,1.0]. This means that for a single value of *pr_cross*, say, 0.1, *pr_mut* is fixed, say at 0.1 while *popsize* is varied from 10 to 100. Thereafter *pr_mut* is incremented to 0.2 and so on. For each set of values, the experiment was run five times and the average fitness is computed while keeping track of the best fitness found. Thus for a single value of *pr_cross*, there were 100 experimental runs. In Table 8, we present some selected best performing combination for this experiment. For each parameter set of *pr_mut*, *pr_cross* and *popsize*, the number of generations to get the final allocation is noted and the average number of generations over the five runs computed and shown in columns 4 and 6 for male and female allocations respectively. The summary of some results of the experiment are presented in Figs. 8 to 11.

The convergence of the algorithm for the problem was tested using the combination of best fitness and average fitness values. In Fig. 8, the values of the overall average fitness converge to the best fitness values and both move towards the optimal value of 1.0 as the number of generation progresses. Fig. 9 depicts the distribution of the average fitness as the value of *pr_cross* varies as described above. The best effects of *pr_cross* on the performance of the algorithm was observed around *pr_cross*=0.2 while values between 0.5 and 0.9 worsen

the performance of the algorithm with the exception of 0.8 after which the algorithm shows downward trend. The effects of *pr_mut* was studied in a similar experiment but with values of *pr_mut* in [0.0,0.09]. Fig. 10 represents results of average fitness plotted against *pr_mut* with *pr_cross* fixed at 0.1. Good solutions were observed with *pr_mut* in [0.06,0.08]. We went further to find the overall effects of *pr_mut* in the entire experiment as presented in Fig. 11. Here the overall average fitness values for the entire experiment at each value of *pr_mut* were taken into consideration. Generally, the algorithm improves in performance with increase in the value of mutation rate for the range studied. The algorithm performed most poorly at *pr_mut* in [0.0,0.02].

A sample of generated allocation distribution at the category and hall levels in one of the experimental runs is presented in Appendix B. A study of the results obtained shows a high degree of satisfaction of given constraints.

Table 8: Cross-section of some best performing combinations during a set of runs

pr_cross	*pr_mut*	Ψ	Male Hall Allocation		Female Hall Allocation	
			Avg # of Gen	Best fitness	Avg # of Gen	Best fitness
0.1	0.9	80	240.4	1.0	179.6	0.9999
0.2	0.5	90	262.6	1.0	224.6	0.9998
0.2	0.7	90	179.2	1.0	152.6	0.9999
0.2	0.8	90	186.2	1.0	202.2	1.0
0.3	0.4	100	227.2	1.0	191.4	1.0
0.3	0.5	90	230.6	1.0	217.2	0.9999
0.3	0.7	70	240.4	1.0	197.8	1.0
0.3	0.8	80	215.4	1.0	156.4	1.0
0.3	0.9	100	154.0	1.0	147.2	1.0
0.4	0.5	100	216.6	1.0	204.8	1.0
0.5	0.8	100	220.0	1.0	201.6	1.0
0.5	0.9	100	201.2	1.0	175.6	1.0

5. Conclusion and Further work

SAP finds application in many real-world scenarios. Of recent, heuristics and metaheuristics are being applied to some domain-specific SAPs which led to many interest in the area. However, a lot of domain-specific SAP exists in developing countries that need equal attention. We identified one of such SAP domain, namely the HSAP as it affects tertiary institutions. The scarcity of hostel space is reported having adverse effect on success and graduation rate in developing countries. However, we believe effective and efficient management of available hostel space can help to achieve these goals. We proposed a GA-driven hierarchical heuristic solution to the problem. This is part of our initial application of metaheuristics to HSAP. Results obtained show the feasibility and efficiency of metaheuristic to HSAP. Incorporating this new idea into hostel space allocation software

system will thus greatly help to overcome bottlenecks that characterises the ad-hoc approach presently in place. Further works include improvement on these early results and incorporation of more advanced metaheuristics.

Acknowledgements

The Authors appreciate the financial supports of the University of Lagos (Nigeria) and the School of Computational and Applied Mathematics of the University of Witwatersrand (South Africa). The effort of Mr. O. Bastos towards data collection and prosecution of the work is acknowledged.

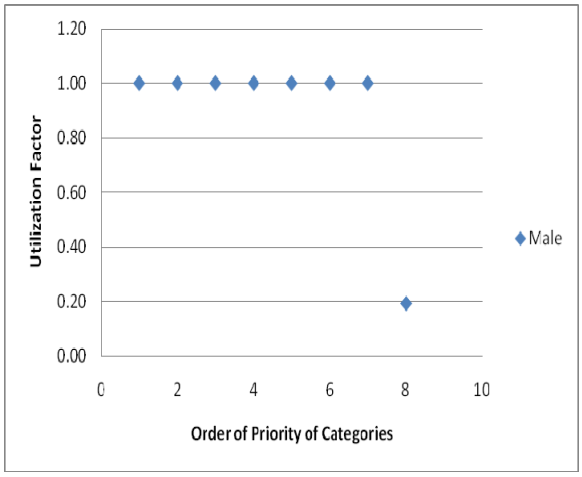

Fig. 6: Category Allocation distribution for CaH1

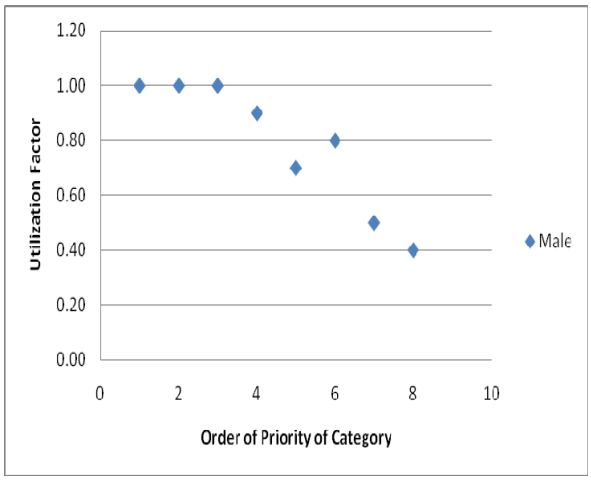

Fig. 7: Category Allocation distribution for CaH2

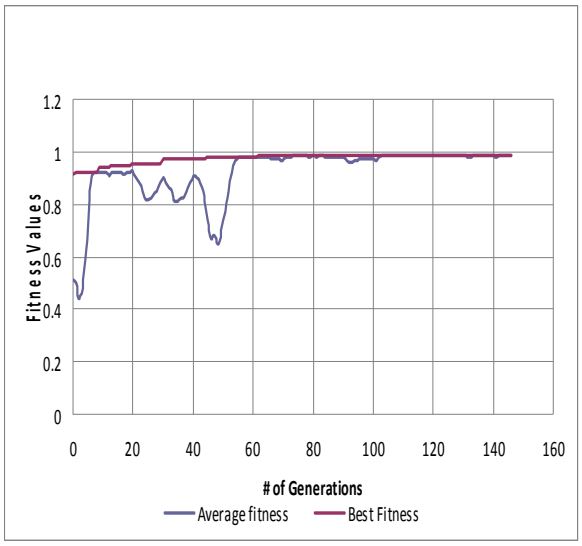

Fig.8: Test of Convergence over 200 Generations

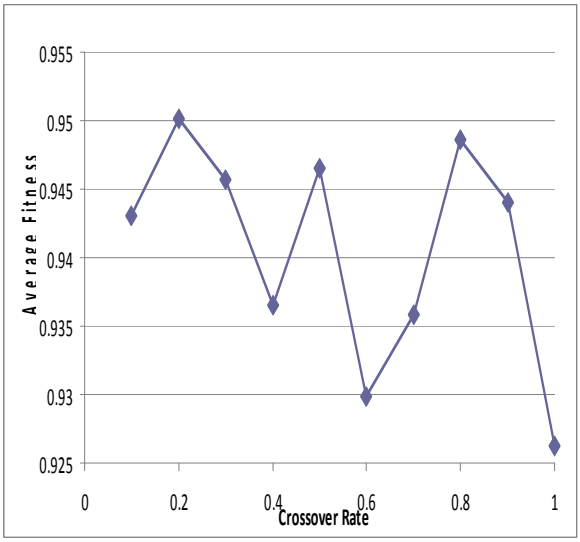

Fig. 9: Crossover rate vs average fitness

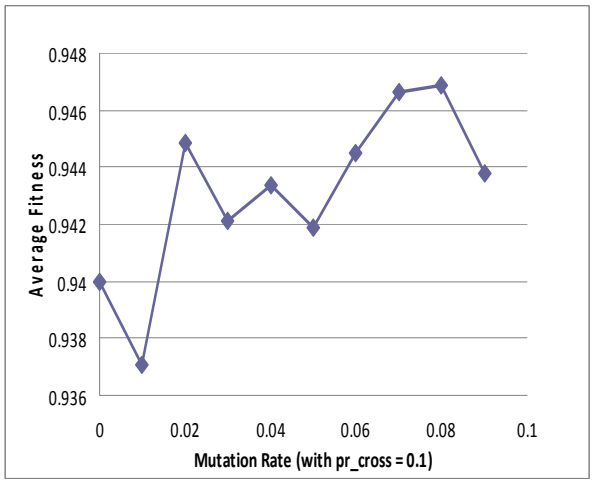

Fig. 10: Mutation rate vs average fitness *(pr_cross = 0.1)*

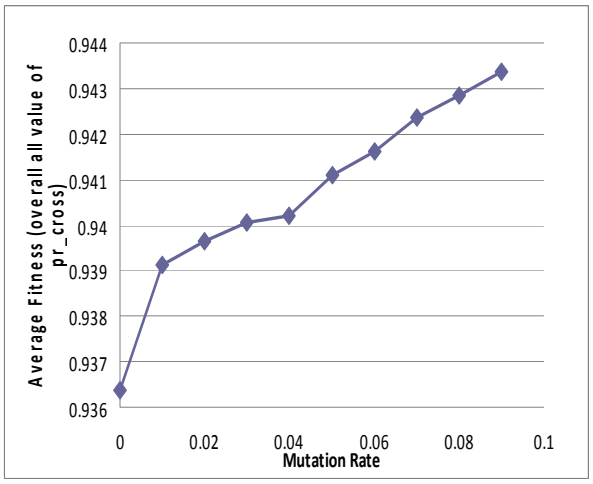

Fig. 11: Mutation rate vs average fitness (pr_cross in [0.1,1.0]

References

[1] R. Bai, An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation. PhD Thesis. School of Computer Science and Information Technology, University of Nottingham, UK, 2005.

[2] J.D. Landa-Silva, Metaheuristics and Multi-objective Approaches for Space Allocation. PhD Thesis. School of Computer Science and Information Technology, University of Nottingham, UK, 2003.

[3] A.O. Adewumi, N, Ihemedu and J.O.A. Ayeni, An Evolutionary-based Approach to University Course Time-Tabling Problem. First Annual Research Conference and Fair, University of Lagos, October 26, (2005) No. 9.11

[4] H.L. Morgan, Optimal Space Allocation on Disk Storage Devices. *Communications of the ACM*, ACM New York, NY, USA, 17 (3) (1974) 139-142.

[5] D.C. Mattfeld, Allocation of Storage Space. In. D.C. Mattfeld, *The Management of Transshipment Terminals*, Operations Research/Computer Science Interfaces Book Series 35 (2006) 83-108.

[6] E.K. Burke and D.B. Varley, Space Allocation: An Analysis of Higher Education Requirements. In: E.K. Burke and 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 PATAT 1997, *Lecture Notes in Computer Science*, Springer, 1408 (1998) 20-33.

[7] E.K. Burke, P. Cowling, J.D. Landa-Silva, B. McCollum, Three Methods to Automate the Space Allocation Process in UK Universities, Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling, PATAT 2000, Konstanz, Germany, (2000) 374-393.

[8] Wikipedia. Heuristic algorithm. Accessed via http://en.wikipedia.org/wiki/Heuristic algorithm in October 2009.

[9] F. Glover, Future paths for integer programming and links to artificial intelligence. *Computer & Operations Research,* 13 (5) (1986) 533–549.

[10] M. Gendreau, An Introduction to Tabu Search. In. F. Glover and G.A. Kochenberger (eds.). *Metaheuristic Handbook*, Kluwer Academic Publishers, Boston, MA, (2003) 37-54

[11] A. Petrovski, A. Wilson and J. Mccall. Statistical Identification and Optimisation of Significant GA Factors. Technical paper, The Robert Gordon University, School of Computer and Mathematical Sciences, Scotland, 2007

[12] A.O. Adewumi, B.A. Sawyerr and M.M. Ali, A Heuristics Approach to the University Timetabling Problems. *Engineering Computations*, Emerald Publishers, UK (2009) To appear.

[13] E.K. Burke and D.B. Varley. Automating Space Allocation in Higher Education. Selected Papers from the 2nd Asia Pacific Conference on Simulated Evolution and Learning SEAL 98, *Lectures Notes in Artificial Intelligence*, Springer, 1585 (1998) 66-73.

[14] M. Yang and W. Chen, A Study of Shelf Space Allocation and Management. *International Journal of Production Economics*, Elsevier, 60-61 (1999), 309-317.

[15] E. Annevelink and R.A.C.M. Broekmeulen, The Genetic Algorithm applied to Space Allocation Planning in Pot-Plant Nurseries. XII International Symposium on Horticultural Economics *Acta Hort. (ISHS)* 340 (1995) 141-148.

[16] J.S. Dean, Staff Scheduling by a Genetic Algorithm with a Two-Dimensional Chromosome, In E.K. Burke, M. Gendreau, (Eds.), *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling – PATAT 2008,* Montreal, Canada, August 18-22, 2008, 15 pgs

[17] Futminna, Student Affairs Department. Federal University of Technology, Minna, www.futminna.org/currentstudents/student_affairs.htm. Retrieved in January, 2009.

[18] M&G. Varsities run out of housing. Mail & Guardians (South African) Newspaper article on higher learning, September 23, 2009 pg 31.

[19] D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operations Research.* 144 (1999) 528–541.

[20] A. Federgruen, G.V. Ryzin. Probabilistic Analysis of a Generalized Bin Packing Problem and Applications. *Operations Research*. 45(4) (1997) 596-609.

[21] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[22] J.R. Koza, 2007. Genetic Algorithms and Genetic Programming. Presentation. Department of Electrical Engineering, School of Engineering, Stanford University, Stanford, California. http://www.smi.stanford.edu/people/koza/, Retrieved in August.

[23] D.E. Goldberg, *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison Wesley, 1989

[24] M. Mitchell, *An Introduction to Genetic Algorithms.* The MIT Press, 1998.

[25] K.A. Dowsland. Genetic Algorithms - a tool for OR? *Journal of the Operational Research Society* 44 (1996), 550-561 [ISSN 0160-5682]

[26] A.O. Adewumi. Some Improved Genetic-Algorithms Based Heuristics for Global Optimization with Innovative Applications. PhD thesis to be submitted in 2009, School of Computational and Applied Mathematics, University of Witwatersrand, South Africa.

[27] A.O. Adewumi and M.M. Ali. A Multi-level Genetic Algorithm for a Multi-Stage Space Allocation Problem. *Mathematical and Computer Modeling*. To Appear.

Appendix A – Sample Data Set

The first table contains the number of applicants per category while the rest give the number of block/floor (with capacity) for the various halls. A label ***A 0*** implies block A floor 0 (ground floor). *Cap* stands for the capacity. *Cap* with value 0 implies that the block/floor is reserved and not to be allocated. The total capacity for the hall is given in bracket.

Category	Fy	Fo	Fr	Ht	Sp	Sc	Ds	Ot
Male	1 240	20	1 332	70	400	400	100	1 800
Female	1 420	25	1 367	80	500	230	60	1 000

HA1 (660)		HA2 (440)				HB1 (800)			
Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap
A 0	18	**A 0**	0	**C 0**	0	**F 0**	20	**R 0**	20
A 1	100	**A 1**	40	**C 1**	40	**F 1**	60	**R 1**	60
A 2	100	**A 2**	40	**C 2**	40	**F 2**	60	**R 2**	60
A 3	100	**A 3**	40	**C 3**	40	**F 3**	60	**R 3**	60
A 4	12	**B 0**	0	**D 0**	0	**G 0**	20	**S 0**	20
B 0	18	**B 1**	40	**D 1**	28	**G 1**	60	**S 1**	60
B 1	100	**B 2**	40	**D 2**	28	**G 2**	60	**S 2**	60
B 2	100	**B 3**	40	**D 3**	28	**G 3**	60	**S 3**	60
B 3	100								
B 4	12								

HB5 (968)				HC1 (526)				HC4 (512)	
Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap
E 0	20	**J 2**	60	**1 0**	40	**11 0**	40	**1 0**	80
E 1	60	**J 3**	60	**2 0**	40	**12 0**	40	**1 1**	144
E 2	60	**T 0**	12	**3 0**	40	**13 0**	46	**1 2**	144
E 3	60	**T 1**	60	**4 0**	40			**1 3**	144
H 0	12	**T 2**	60	**5 0**	40				
H 1	60	**T 3**	60	**6 0**	40				
H 2	60	**W 0**	12	**7 0**	40				
H 3	60	**W 1**	60	**8 0**	40				
J 0	12	**W 2**	60	**9 0**	40				
J 1	60	**W 3**	60	**10 0**	40				
HA3 (866)				**HB2 (764)**				**HB3 (276)**	
Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap
A 0	0	**D 2**	60	**D 0**	0	**U 3**	80	**C 0**	0
A 1	30	**D 3**	60	**D 1**	32	**V 0**	40	**C 1**	30
A 2	30	**E 0**	60	**D 2**	32	**V 1**	40	**C 2**	30
B 0	40	**E 1**	60	**D 3**	32	**V 2**	40	**L 0**	40
B 1	40	**E 2**	60	**K 0**	48			**L 1**	40
B 2	40	**E 3**	60	**K 1**	80			**L 2**	40
C 0	40	**F 0**	0	**K 2**	80			**P 0**	0
C 1	40	**F 1**	30	**K 3**	80			**P 1**	24
C 2	40	**F 2**	30	**U 0**	20			**P 2**	24
D 0	0	**G 0**	43	**U 1**	80			**S 0**	0
D 1	60	**H 0**	43	**U 2**	80			**S 1**	24
								S 2	24

HB4 (524)				HC3 (646)		HC2 (512)	
Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap	Blk/Flr	Cap
A 0	30	**M 2**	32	**A 0**	60	**1 0**	80
A 1	40	**M 3**	32	**A 1**	60	**1 1**	144
A 2	40	**N 0**	32	**A 2**	60	**1 2**	144
A 3	40	**N 1**	32	**B 0**	40	**1 3**	144
B 0	30	**N 2**	32	**B 1**	100		
B 1	40	**N 3**	32	**B 2**	100		
B 2	40			**C 0**	26		
B 3	40			**C 1**	100		
M 0	0			**C 2**	100		
M 1	32						

Appendix B: Sample Generated Allocation Distribution for Category and Hall

CATEGORY ALLOCATION DISTRIBUTION

	Allocated		Unallocated	
	Male	Female	Male	Female
foreign:	20	25	0	0
health:	70	80	0	0
sport:	400	500	0	0
final:	1 240	1 420	0	0
first:	1 332	1 333	0	34
scholar:	400	230	0	0
discretion:	100	0	0	60
other:	348	0	1 452	1 000
TOTAL:	3 910	3 588	1 452	1 094
GROSS:	7 498		2 546	

HALL ALLOCATION DISTRIBUTION

	HA1	HA2	HB1	HB5	HC1	HC4
foreign	3	0	12	0	2	3
health	70	0	0	0	0	0
sport	0	0	0	0	400	0
final	227	24	285	687	14	3
first	358	7	274	165	77	451
scholar	0	400	0	0	0	0
discretion	0	4	48	21	0	27
other	2	9	181	95	33	28

	HA3	HB2	HB3	HB4	HC3	HC2
foreign	6	2	8	8	0	1
health	80	0	0	0	0	0
sport	0	0	0	0	500	0
final	307	258	174	450	133	98
first	243	504	94	66	13	413
scholar	230	0	0	0	0	0
discretion	0	0	0	0	0	0
other	0	0	0	0	0	0