Designing Difficult Office Space Allocation Problem Instances with Mathematical Programming

Özgür Ülker and Dario Landa-Silva

Automated Scheduling, Optimisation and Planning (ASAP) Research Group
School of Computer Science, University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
oxu@cs.nott.ac.uk, dario.landasilva@nottingham.ac.uk

Abstract. Office space allocation (OSA) refers to the assignment of room space to a set of entities (people, machines, roles, etc.), with the goal of optimising the space utilisation while satisfying a set of additional constraints. In this paper, a mathematical programming approach is developed to model and generate test instances for this difficult and important combinatorial optimisation problem. Systematic experimentation is then carried out to study the difficulty of the generated test instances when the parameters for adjusting space misuse (overuse and underuse) and constraint violations are subject to variation. The results show that the difficulty of solving OSA problem instances can be greatly affected by the value of these parameters.

Keywords: Office Space Allocation Problem, Integer Programming, Mathematical Modelling, Data Instance Generation, Proof of Optimality.

1 Introduction

We develop a mathematical programming approach to design difficult test instances for the Office Space Allocation (OSA) problem, which is commonly encountered in universities, companies and government institutions. In simple terms, the OSA problem is the task of allocating office spaces (rooms, hallways, etc.) to entities (peoples, machines, roles, etc.) subject to additional constraints. OSA is related to the multiple knapsack [15] and generalised assignment problems [6]. In OSA, the primary goal is to maximise the space utilisation by reducing the misuse of rooms. Misuse of rooms refers to underusing space (under-utilisation of rooms) or overusing space (over-crowding of rooms). Usually, overuse is considered more undesirable than underuse of space. Additional constraints (e.g. people grouping conditions) can arise in different organisations when allocating office space. Any of such constraints can be *hard* (satisfied all the time) or *soft* (desirable but not necessary).

Office space allocation is usually a continuous process due to the constant changes in an organisational environment (departure/arrival of new personnel,

P.M. Pardalos and S. Rebennack (Eds.): SEA 2011, LNCS 6630, pp. 280–291, 2011.

maintenance/renovation of existing office space, restructuring in organisations etc). This process can involve many conflicting objectives and constraints difficult to tackle when using manual approaches. An automated allocation system can deal with such conflicting objectives and constraints better than a human decision expert especially if the size of the problem grows. An automated system can also provide alternative solutions for different scenarios more quickly. Then, instead of tackling the complex optimisation problem directly, the decision maker can focus on fine tuning the automatically generated allocation based on organisational preferences and requirements.

In this study, the office space allocation problem as described in Landa-Silva [13] is investigated and extended. A 0/1 integer programming model is developed and then Gurobi [10], a commercial integer linear programming (ILP) solver, is applied to solve it. Based on this model, we develop a test instance generator to further investigate the difficulty of the OSA problem through systematic experimentation. The paper is organised as follows: Section 2 outlines previous research on office space allocation. Section 3 presents the mathematical model proposed for OSA while Section 4 describes the test instance generator. Section 5 presents and discusses the results from our experimental study. Finally, Section 6 summarises our contributions and proposes future research directions.

2 Outline of Previous Related Work

One of the earliest works on the optimisation of office space utilisation is that of Ritzman et al. [17], who developed a linear programming model for the distribution of academic offices at the Ohio State University. Benjamin et al. [1] used a linear goal programming model for planning and improving the utilisation of the layout of floor space in a manufacturing laboratory at the University of Missouri Rolla. Giannikos et al. [8] developed a goal programming approach to automate the distribution of offices among staff in an academic institution.

Burke and Varley [5] reported on a questionnaire on the space allocation process in 38 British universities. The emphasis was on the scope of the problem, computing tools to solve it and the constraints in each university. Burke et al. [3] applied hill climbing, simulated annealing [11] and a genetic algorithm [9] to solve the *allocation* (task of creating a complete solution from scratch) and *reorganisation* (task of reallocating entities in a given solution) variants of the problem. The authors used *allocation*, *relocation*, and *swap* operators for moving entities between rooms. Burke et al. [2] later investigated a hybridisation of their previous approaches under a population based framework. The initial solutions were created using a hill climbing operator and then improved using simulated annealing with adaptive cooling schedule. Burke et al. [4] applied multi-objective optimisation [7] to the OSA problem comparing weighted aggregation to Pareto dominance for tackling two objectives: the total space misuse (under/over usage of rooms) and the sum of (soft) constraint violations. They found that these two objectives were conflicting in nature. Later, Landa-Silva and Burke [12] developed an asynchronous cooperative local search method in which local search threads in a population co-operate with each other asynchronously to improve

the overall solution quality. Based on their experiments, the soft constraint *'group by'* was regarded as the most difficult one to satisfy (high number of violations).

Pereira et al. [16] applied a greedy local search and tabu search algorithm to tackle an OSA problem where the goals are to minimise the distance between employees in the same organisation, minimise the office space misallocation and maximise the office space allocation. They reported that tabu search performed better on their OSA problem instances. Lopes and Girimonte [14] analysed a variant of the OSA problem (similar to the one described in [12]) arising in the European Space Agency (ESA). They implemented four types of metaheuristics: hill climbing, simulated annealing, tabu search and the hybrid metaheuristic in [2]. To improve the performance of these algorithms, variations to the local search, and new constraints management algorithms were designed by the authors.

3 Mathematical Programming Model

An earlier version of the following model was presented in [18]. The set of rooms is denoted by R and the set of entities is denoted by E. The size of entity e is S_e and the capacity of room r is C_r. There is a matrix X of $|R| \times |E|$ binary decision variables where each $x_{er} = 1$ if entity e is allocated to room r, otherwise $x_{er} = 0$. Let A be the adjacency list of $|R|$ adjacency vectors each one denoted by A_r and holding the list of rooms adjacent to room r. Similarly, let N be the nearby list of $|R|$ nearby vectors each one denoted by N_r and holding the list of rooms near to room r. The adjacency vector A_r for a room r is usually quite smaller compared to the nearby vector N_r, i.e. more rooms are considered to be 'near' to room r than considered to be 'adjacent' to the same room.

There are ten requirements or constraints handled here. Most of these constraints can be set as *hard* (must be satisfied) or *soft* (desirable to satisfy) in our formulation. In other words, when a constraint is set as soft, minimising its violation becomes an objective in the problem formulation. The exception here is the *'All allocated'* constraint (all entities must be allocated) which is always enforced. The next subsections present these alternative formulations in the constraint set and in the objective function. For each constraint type (defined below), HC^{al}, HC^{na}, HC^{sr}, HC^{nsr}, HC^{nsh}, HC^{ad}, HC^{gr}, HC^{aw}, HC^{cp} denote the corresponding constraint as *hard* while SC^{al}, SC^{na}, SC^{sr}, SC^{nsr}, SC^{nsh}, SC^{ad}, SC^{gr}, SC^{aw}, SC^{cp} denote the corresponding constraint as *soft*. Note that each *soft* constraint is associated with a binary indicator variable y^{cst} which is set to 1 if the respective *soft* constraint is violated. Some constraint types require additional binary variables (y_r^{cst}) over $r \in R$.

3.1 Modeling Hard Constraints

All allocated: each entity $e \in E$ must be allocated to exactly one room $r \in R$.

$$\sum_{r \in R} x_{er} = 1 \quad \forall e \in E \tag{1}$$

Allocation: entity e to be placed into room r. $((e,r) \in HC^{\text{al}})$.

$$x_{er} = 1 \tag{2}$$

Non allocation: entity e not to be placed into room r. $((e,r) \in HC^{\text{na}})$.

$$x_{er} = 0 \tag{3}$$

Same room: entities e_1 and e_2 to be placed into same room. $((e_1,e_2) \in HC^{\text{sr}})$.

$$x_{e_1r} = 1 \leftrightarrow x_{e_2r} = 1 \quad \forall r \in R \quad \text{i.e.}$$

$$x_{e_1r} - x_{e_2r} = 0 \quad \forall r \in R \tag{4}$$

Not in same room: entities e_1 and e_2 to be placed into different rooms. $((e_1,e_2) \in HC^{\text{nsr}})$.

$$x_{e_1r} = 1 \leftarrow x_{e_2r} = 0 \quad \forall r \in R \quad \text{i.e.}$$

$$x_{e_1r} + x_{e_2r} \leq 1 \quad \forall r \in R \tag{5}$$

Not sharing: entity e not to share a room with any other entity. $((e) \in HC^{\text{nsh}})$.

$$x_{er} = 1 \rightarrow \sum_{f \in E-e} x_{fr} = 0 \quad \forall r \in R \quad \text{i.e.}$$

$$\sum_{f \in E-e} x_{fr} \leq (|E|-1) - (|E|-1)x_{er} \quad \forall r \in R \tag{6}$$

Adjacency: entities e_1 and e_2 placed into adjacent rooms. $((e_1,e_2) \in HC^{\text{ad}})$.

$$x_{e_1r} = 1 \rightarrow \sum_{s \in A_r} x_{e_2s} = 1 \quad \forall r \in R \quad \text{i.e.}$$

$$x_{e_1r} \leq \sum_{s \in A_r} x_{e_2s} \leq 1 \quad \forall r \in R \tag{7}$$

Group by: entities in a group placed near to the group head f. $((e,f) \in HC^{\text{gr}})$.

$$x_{er} = 1 \rightarrow \sum_{s \in N_r} x_{fs} = 1 \quad \forall r \in R \quad \text{i.e.}$$

$$x_{er} \leq \sum_{s \in N_r} x_{fs} \leq 1 \quad \forall r \in R \tag{8}$$

Away from: entities e_1 and e_2 to be placed in rooms away from each other. $((e_1,e_2) \in HC^{\text{aw}})$.

$$x_{e_1r} = 1 \rightarrow \sum_{s \in N_r} x_{e_2s} = 0 \quad \forall r \in R \quad \text{i.e}$$

$$0 \leq \sum_{s \in N_r} x_{e_2 s} \leq 1 - x_{e_1 r} \quad \forall r \in R \tag{9}$$

Capacity: Room r must not be overused. $((r) \in HC^{\text{cp}})$.

$$\sum_{e \in E} S_e x_{er} \leq C_r \tag{10}$$

3.2 Modeling Constraints as Objectives

Allocation: indicator variable $y^{\text{al}}(i)$ is set if $SC^{\text{al}}(i)$ is not satisfied.

$$y^{\text{al}}(i) = 1 - x_{er} \tag{11}$$

Non allocation: indicator variable $y^{\text{na}}(i)$ is set if $SC^{\text{na}}(i)$ is not satisfied.

$$y^{\text{na}}(i) = x_{er} \tag{12}$$

Same room: indicator variable $y^{\text{sr}}(i)$ is set if $SC^{\text{sr}}(i)$ is not satisfied.

$$2y_r^{\text{sr}}(i) - 1 \leq x_{e_1 r} - x_{e_2 r} \leq 1 - \epsilon + \epsilon y_r^{\text{sr}}(i) \quad \forall r \in R \tag{13}$$

$$y^{\text{sr}}(i) = \sum_{r \in R} y_r^{\text{sr}}(i) \tag{14}$$

Not in same room: indicator variable $y^{\text{nsr}}(i)$ is set if $SC^{\text{nsr}}(i)$ is not satisfied.

$$(1 + \epsilon) - (1 + \epsilon) y_r^{\text{nsr}}(i) \leq x_{e_1 r} + x_{e_2 r} \leq 2 - y_r^{\text{nsr}}(i) \quad \forall r \in R \tag{15}$$

$$y^{\text{nsr}}(i) = \sum_{r \in R} y_r^{\text{nsr}}(i) \tag{16}$$

Not sharing: indicator variable $y^{\text{nsh}}(i)$ is set if $SC^{\text{nsh}}(i)$ is not satisfied.

$$(|E| - 1)(2 - x_{er} - y_r^{\text{nsh}}(i)) \leq \sum_{f \in E - e} x_{fr} \quad \forall r \in R \tag{17}$$

$$\sum_{f \in E - e} x_{fr} \leq (|E| - 1)(1 - x_{fr} + \epsilon - (|E| - 1 + \epsilon) y_r^{\text{nsh}}(i) \quad \forall r \in R \tag{18}$$

$$y^{\text{nsh}}(i) = \sum_{r \in R} (1 - y_r^{\text{nsh}}(i)) \tag{19}$$

Adjacency: indicator variable $y^{\text{ad}}(i)$ is set if $SC^{\text{ad}}(i)$ is not satisfied.

$$y_r^{\text{ad}}(i) + x_{e_1 r} - 1 \leq \sum_{s \in A_r} x_{e_2 s} \leq x_{e_1 r} - \epsilon + (1 + \epsilon) y_r^{\text{ad}}(i) \quad \forall r \in R \tag{20}$$

$$y^{\mathrm{ad}}(i) = \sum_{r \in R} (1 - y_r^{\mathrm{ad}}(i)) \tag{21}$$

Group by: indicator variable $y^{\mathrm{gr}}(i)$ is set if $SC^{\mathrm{gr}}(i)$ is not satisfied.

$$y_r^{\mathrm{gr}}(i) + x_{er} - 1 \leq \sum_{s \in N_r} x_{fs} \leq x_{er} - \epsilon + (1 + \epsilon) y_r^{\mathrm{gr}}(i) \quad \forall r \in R \tag{22}$$

$$y^{\mathrm{gr}}(i) = \sum_{r \in R} (1 - y_r^{\mathrm{gr}}(i)) \tag{23}$$

Away from: indicator variable $y^{\mathrm{aw}}(i)$ is set if $SC^{\mathrm{aw}}(i)$ is not satisfied.

$$1 - x_{e_1 r} + \epsilon - (1 + \epsilon) y_r^{\mathrm{aw}}(i) \leq \sum_{s \in N_r} x_{e_2 s} \leq 2 - x_{er} - y_r^{\mathrm{aw}}(i) \quad \forall r \in R \tag{24}$$

$$y^{\mathrm{aw}}(i) = \sum_{r \in R} (1 - y_r^{\mathrm{aw}}(i)) \tag{25}$$

Capacity: indicator variable $y^{\mathrm{cp}}(i)$ is set if $SC^{\mathrm{cp}}(i)$ is not satisfied.

$$\sum_{e \in E} S_e x_{er} + (C_r + \epsilon)(1 - y^{\mathrm{cp}}(i) \geq C_r + \epsilon \tag{26}$$

$$\sum_{e \in E} S_e x_{er} + (\sum_{e \in E} S_e - C_r)(1 - y^{\mathrm{cp}}(i) \leq \sum_{e \in E} S_e \tag{27}$$

3.3 Objective Function

The objective function is the weighted sum of the space misuse ($underuse + 2 \cdot overuse$) and the soft constraints violation penalty. The penalties associated to each soft constraint type are: w^{al}, w^{na}, w^{sr}, w^{nsr}, w^{nsh}, w^{ad}, w^{gr}, w^{aw}, and w^{cp}. The objective function Z to minimise is given by:

$$\begin{aligned} Z = \sum_{r \in R} max \left(C_r - \sum_{e \in E} x_{er} S_e \ , \ 2 \sum_{e \in E} x_{er} S_e - C_r \right) + w^{\mathrm{al}} \sum_{i=1}^{|SC^{\mathrm{al}}|} y^{\mathrm{al}}(i) \\ + w^{\mathrm{na}} \sum_{i=1}^{|SC^{\mathrm{na}}|} y^{\mathrm{na}}(i) + w^{\mathrm{sr}} \sum_{i=1}^{|SC^{\mathrm{sr}}|} y^{\mathrm{sr}}(i) + w^{\mathrm{nsr}} \sum_{i=1}^{|SC^{\mathrm{nsr}}|} y^{\mathrm{nsr}}(i) + w^{\mathrm{nsh}} \sum_{i=1}^{|SC^{\mathrm{nsh}}|} y^{\mathrm{nsh}}(i) \\ + w^{\mathrm{ad}} \sum_{i=1}^{|SC^{\mathrm{ad}}|} y^{\mathrm{ad}}(i) + w^{\mathrm{gr}} \sum_{i=1}^{|SC^{\mathrm{gr}}|} y^{\mathrm{gr}}(i) + w^{\mathrm{aw}} \sum_{i=1}^{|SC^{\mathrm{aw}}|} y^{\mathrm{aw}}(i) + w^{\mathrm{cp}} \sum_{i=1}^{|SC^{\mathrm{cp}}|} y^{\mathrm{cp}}(i) \end{aligned} \tag{28}$$

4 Test Instance Generator for OSA

We have access to some real-world data for the OSA problem but in order to systematically investigate this problem, we developed a test instance generator based on the mathematical programming approach. The generator currently supports the nine types of constraints described in the previous section, and the generation of entities, rooms, and floor layout. An outline of the generator is shown in Algorithm 1. The generator algorithm starts with the creation of entities, groups (set of entities) and initial sets of *hard* and *soft* constraints. Then it creates or modifies the floor layout and/or the room sizes by means of a constructive heuristic. The generator tries to 'plant' a core solution into the instance by allocating entities into rooms according to the initial constraint sets. In order to experimentally investigate the difficulty of the created test instances, four parameters directly related to space misuse (overuse/underuse) and to soft constraint violations were devised. These parameters are:

Algorithm 1. OSA Test Instance Generator Algorithm

Input: input file of parameters.
Output: data instance.
— Create Groups, Entities, Floor Layout and Constraints.
— Placement of entities according to the *hard* constraints.
— Calculate space that is minimally required for each room.
— Placement of entities according to the *soft* constraints.
— Room size adjustments via (positive or negative) slack amounts.
— Post processing

1. *Slack Space Rate:* After all the entities are allocated to rooms, this parameter determines whether the room size will be modified. This parameter adjusts the amount of space misuse.
2. *Negative Slack Amount:* Is the amount by which room capacity is reduced and is determined by a percentage of the total sizes of the entities already allocated to rooms. This parameter adjusts the amount of space overuse.
3. *Positive Slack Amount:* Is the amount by which room capacity is increased and is determined by a percentage of the total sizes of the entities already allocated to rooms. This parameter adjusts the amount of space underuse.
4. *Violation Rate:* When allocating entities to rooms as indicated by the soft constraints, there might be some violations of constraints, i.e. conflicts. A soft constraint is removed from the constraint set with this rate if such a conflict occurs. This parameter adjusts the violation of soft constraints.

5 Experiments and Results

Six real-world benchmark instances (called Nott) taken from [13] were used for experimentation here. Additional experiments were carried using test instances created with our generator as well.

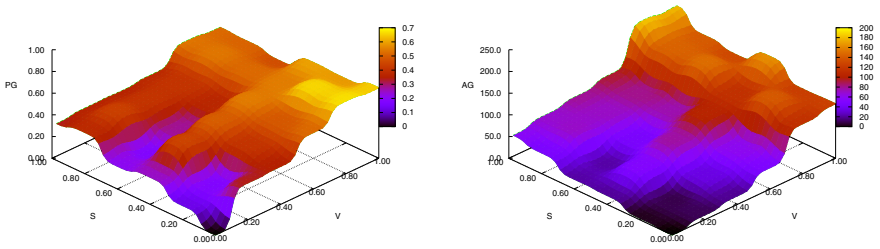
Table 1. Constraint penalties for the best results obtained for each problem instance of the Nott Dataset

	Nott1	**Nott1***	**Nott1b**	**Nott1c**	**Nott1d**	**Nott1e**	**Wolver**
Allocation	40.00	20.00	0.00	40.00	0.00	0.00	0.00
Same Room	0.00	0.00	80.00	0.00	0.00	0.00	0.00
Not Sharing	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Adjacency	10.00	10.00	0.00	10.00	0.00	0.00	0.00
Group by	11.18	22.36	11.18	11.18	11.18	0.00	0.00
Away From	20.00	30.00	0.00	20.00	0.00	40.00	0.00
Constraint Penalty	81.18	82.36	91.18	81.18	11.18	40.00	0.00
Overuse	130.80	106.40	64.20	182.90	164.70	13.80	486.04
Underuse	134.20	122.00	87.90	41.65	26.85	123.90	148.15
Usage Penalty	265.00	228.40	152.10	224.55	191.55	137.70	634.19
Total Penalty	346.18	310.76	243.28	305.73	202.73	177.70	634.19
Lower Bound	201.86	273.16	131.45	305.73	202.73	177.70	634.19
Percentage Gap	%41.70	%12.10	%46.00	%0.00	%0.00	%0.00	%0.00

To solve the 0/1 IP formulation, Gurobi 3.0.1 [10] was used on a PC with a Core 2 Duo E8400 3Ghz processor and 2GB of RAM. Each problem instance was given 30 minutes of solver runtime. The following penalties were used for each soft constraint violation: $w^{\text{al}} = 20$, $w^{\text{na}} = 10$, $w^{\text{sr}} = 10$, $w^{\text{nsr}} = 10$, $w^{\text{nsh}} = 50$, $w^{\text{ad=10}}$, $w^{\text{gr}} = 11.18$ for the Nott instances, $w^{\text{gr}} = 10$ for the generated test instances, $w^{\text{aw}} = 10$, and $w^{\text{cp}} = 10$.

Table 1 summarises the best results obtained after a run of 30 minutes on each problem instance (from Nott1 to Wolver). Note that these dataset instances do not contain *non-allocation*, *not in same room*, or *capacity* constraints, the other six constraint types are present in these real-world instances. The penalties for each constraint violation are given in rows 2-7 of the table. Two different experiments were run on the largest problem instance Nott1. It was observed during experiments that minimising *same room* constraint violations is the most difficult, especially for the Nott1b instance (value of 80.00). So, we conducted an additional experiment for tackling the *same room* constraint in the Nott1 instance (largest one). In Nott1 column, *same room* constraints were all set as soft, whereas in column Nott1*, *same room* constraints were all set as *hard*. Notice that this latter setup achieved a lower usage penalty by roughly 35 square meters. We can see that in all these instances, the constraint penalty turned out to be significantly lower than the usage penalty. For all these real-world instances, our model and solution approach produced the best results in the literature so far [18,12]. Table 1 also shows that for instances Nott1c, Nott1d, Nott1e and Wolver we obtained optimal results while instances Nott1 and Nott1b remain very challenging.

Our next experiments focused on studying the difficulty of the generated test instances by changing the four generator parameters described in Section 4: *slack space rate* (S), *positive slack amount* (P), *negative slack amount* (N) and *violation rate* (V). The term 'difficulty' in this paper refers to the difficulty of



(a) Percentage Gap (PG) (b) Absolute Gap (AG)

Fig. 1. The effect of changing S and V on the percentage and absolute gaps

the optimality proof for the ILP solver (i.e. the gap or difference between the best found solution and the best found lower-bound on the optimal solution). The aim of this experimentation was to check if incrementing the above four parameters had any effect on this gap and hence the optimality proof difficulty of the test instances. In generating all our test instances, the generator used the same entity set (with 150 entities), same initial hard constraint set, and same initial soft constraint set (subject to modifications of the V parameter). The S, P, and N parameters were varied to adjust the size of the rooms (92 rooms in each instance) to obtain various amounts of space misuse (underuse/overuse). We created two different datasets. In the $SVe150$ dataset, the *slack space rate* (S) and *violation rate* (V) were varied between 0.0 to 1.0 with 0.2 increments (i.e. 36 test instances). In the $PNe150$ dataset, the *positive slack amount* (P) and *negative slack amount* (N) were varied between 0.00 to 0.25 with 0.05 increments (i.e. 36 more instances).

Table 2(a) shows results for the $SVe150$ dataset. Columns S and V represent the amount of *slack space* and *violation* rates respectively. Columns C, B and % represent the objective value achieved, the lower bound on the objective value and the percentage gap between the objective value and the lower bound respectively. The *positive slack* (P) and *negative slack* (N) amounts were fixed at 0.10 for this experiment. It was observed that although increasing S and V individually increases the percentage gaps, this increase tends to stabilize (and in fact decreases) after certain levels of S and V. It was observed that the percentage gaps tend to peak around $S = 0.4$ and $V = 0.8$. The absolute gaps (the raw difference between the bound and obtained objective value) exhibit a somehow expected smooth increase with larger S and V values. Table 2(b) shows results for the $PNe150$ dataset. The effect of changing the *positive slack* (P) and *negative slack* (N) amounts on the achieved objective values, lower bounds and percentage gaps obtained is observed in this table. The *slack rate* (S) and *violation rate* (V) were fixed at 0.5. Figures 1 and 2 illustrate graphically the effect of S, V, P and N in our experimental results.

The percentage gaps obtained in our experiments serve as an evidence that our generator is able to create difficult test instances. i.e. with significant high percentage gaps between the achieved objective values and the lower bounds

Table 2. The objective values, the lower bounds and the percentage gaps obtained when solving the $SVe150$ and $PNe150$ generated instances under different values for parameters S, V, P and N

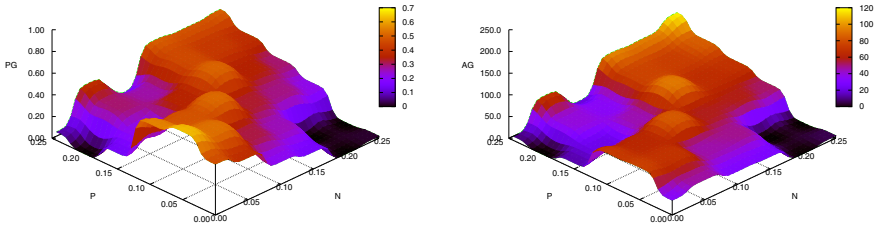
(a) Changing Slack Space Rate (S) and Violation Rate (V)

S	V	C	B	%
0.00	0.00	0.00	0.00	0.0%
0.00	0.20	32.00	21.10	34.0%
0.00	0.40	57.00	37.30	34.5%
0.00	0.60	95.50	46.20	51.6%
0.00	0.80	171.00	55.90	67.3%
0.00	1.00	193.00	67.40	65.1%
0.20	0.00	28.10	24.40	13.2%
0.20	0.20	52.90	43.20	18.3%
0.20	0.40	86.60	51.40	40.6%
0.20	0.60	122.80	62.20	49.3%
0.20	0.80	212.70	72.10	66.1%
0.20	1.00	210.30	86.20	59.0%
0.40	0.00	82.80	62.00	25.1%
0.40	0.20	116.30	63.70	45.2%
0.40	0.40	155.10	77.20	50.3%
0.40	0.60	188.80	84.70	55.1%
0.40	0.80	208.70	94.10	54.9%
0.40	1.00	271.50	107.80	60.3%
0.60	0.00	109.70	87.10	20.6%
0.60	0.20	129.70	107.00	17.5%
0.60	0.40	168.20	121.90	27.5%
0.60	0.60	205.20	129.50	36.9%
0.60	0.80	289.10	138.80	52.0%
0.60	1.00	278.70	147.60	47.1%
0.80	0.00	124.70	76.80	38.4%
0.80	0.20	160.30	89.20	44.4%
0.80	0.40	173.60	101.90	41.3%
0.80	0.60	195.90	114.70	41.5%
0.80	0.80	267.80	122.20	54.4%
0.80	1.00	276.10	134.80	51.2%
1.00	0.00	169.10	111.00	34.4%
1.00	0.20	194.20	124.10	36.1%
1.00	0.40	221.40	141.40	36.1%
1.00	0.60	243.40	149.90	38.4%
1.00	0.80	340.40	157.50	53.7%
1.00	1.00	345.30	165.90	51.9%

(b) Changing Positive Slack (P) and Negative Slack (N) Amounts

P	N	C	B	%
0.00	0.00	73.00	38.59	47.13%
0.00	0.05	119.40	72.70	39.11%
0.00	0.10	145.20	114.29	21.28%
0.00	0.15	186.00	156.88	15.65%
0.00	0.20	210.20	209.90	0.14%
0.00	0.25	250.80	244.74	2.42%
0.05	0.00	119.90	44.90	62.55%
0.05	0.05	130.80	61.06	53.32%
0.05	0.10	141.40	95.47	32.48%
0.05	0.15	185.00	136.33	26.31%
0.05	0.20	202.40	202.40	0.00%
0.05	0.25	232.70	232.70	0.00%
0.10	0.00	130.40	57.67	55.78%
0.10	0.05	134.00	70.09	47.70%
0.10	0.10	162.60	78.64	51.64%
0.10	0.15	176.30	118.26	32.92%
0.10	0.20	205.00	146.51	28.53%
0.10	0.25	240.60	188.26	21.76%
0.15	0.00	96.00	81.50	15.10%
0.15	0.05	105.40	83.25	21.01%
0.15	0.10	124.50	79.35	36.27%
0.15	0.15	183.50	90.22	50.83%
0.15	0.20	192.90	126.09	34.63%
0.15	0.25	229.50	160.13	30.23%
0.20	0.00	108.60	108.60	0.00%
0.20	0.05	135.40	95.37	29.56%
0.20	0.10	129.90	95.47	26.51%
0.20	0.15	167.20	90.65	45.78%
0.20	0.20	177.20	102.75	42.02%
0.20	0.25	219.50	134.46	38.74%
0.25	0.00	126.00	116.21	7.77%
0.25	0.05	180.30	110.82	38.54%
0.25	0.10	132.90	113.27	14.77%
0.25	0.15	192.40	96.33	49.93%
0.25	0.20	195.30	100.71	48.43%
0.25	0.25	233.00	115.48	50.44%

provided by the solver. One interesting observation is that the difficulty of the test instances is not necessarily affected by increasing or decreasing P and N independently. The percentage gaps were usually highest when P and N were set equal or close to each other. Also, the percentage gaps were usually lower



(a) Percentage Gap (PG) (b) Absolute Gap (AG)

Fig. 2. The effect of changing P and N on the percentage and absolute gaps

when the absolute value gap between P and N was increased and the gaps were minimal when $N - P$ was the highest. The absolute value gap between the objective value and the bounds exhibited a similar pattern to the percentage gap case. This can be attributed to the fact that when N is increased, the resulting test instance has a lot of rooms with less available space than required (high overuse) but not enough rooms with extra capacity to compensate the lack of space if P is kept low. At this setting, the solver is expected to immediately allocate any extra space it can find and the remaining time when solving the instance is concentrated on minimizing the overuse. However, when P and N are kept close to each other, there are enough rooms with both overuse and underuse to compensate for each other, hence the solver has to make choices to minimize overuse, underuse and constraint violations, spending more computation time as a result.

6 Conclusions

In this work, a 0/1 IP formulation was proposed to model various *hard* and *soft* constraints in the office space allocation (OSA) problem. This model was implemented in the Gurobi solver and we improved the best results obtained so far for the existing Nott dataset. A test instance generator was also described here and further experiments were carried out on new test instances generated. Our experiments focused on studying the effect that four different parameters of the generator, which affect the space misuse (underuse/overuse) and the soft constraint violations, have on the percentage and absolute gaps between the achieved objective value and the lower-bound on the optimal solution found by the solver. It was observed that an important factor affecting the optimality proof difficulty of the test instances, was the difference between *negative slack* (N) and *positive slack* (P) amounts, which adjust the *overuse* and *underuse* of rooms respectively in the generated test instances. Although raising the *slack space rate* (S) and *violation rate* (V) increased the percentage gaps, their effect was less prominent than the effect of N and P. Future research will concentrate on a detailed study of the effect of these four parameters on the overuse, underuse and constraint violation penalties. We also intend to develop more effective solution techniques to tackle the most difficult instances like Nott1b and Nott1.

References

1. Benjamin, C., Ehie, I., Omurtag, Y.: Planning facilities at the university of missouri-rolla. Interfaces 22(4) (1992)
2. Burke, E.K., Cowling, P., Landa Silva, J.D.: Hybrid population-based metaheuristic approaches for the space allocation problem. In: Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001), pp. 232–239 (2001)
3. Burke, E.K., Cowling, P., Landa Silva, J.D., McCollum, B.: Three methods to automate the space allocation process in UK universities. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 254–273. Springer, Heidelberg (2001)
4. Burke, E.K., Cowling, P., Landa Silva, J.D., Petrovic, S.: Combining hybrid metaheuristics and populations for the multiobjective optimisation of space allocation problems. In: Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 1252–1259 (2001)
5. Burke, E.K., Varley, D.B.: Space allocation: An analysis of higher education requirements. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 20–33. Springer, Heidelberg (1998)
6. Cattrysse, D.G., Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem. European Journal of Operational Research 60(3), 260–272 (1992)
7. Coello Coello, C.A., Lamont, G.B., Van Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Springer, Heidelberg (2006)
8. Giannikos, J., El-Darzi, E., Lees, P.: An integer goal programming model to allocate offices to staff in an academic instituition. Journal of the Operational Research Society 46(6), 713–720 (1995)
9. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, 1st edn. Addison-Wesley, Reading (1989)
10. Gurobi Optimization: Gurobi (2010), `http://www.gurobi.com`
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
12. Landa-Silva, D., Burke, E.K.: Asynchronous cooperative local search for the office-space-allocation problem. INFORMS J. on Computing 19(4), 575–587 (2007)
13. Landa-Silva, J.D.: Metaheuristics and Multiobjective Approaches for Space Allocation. Ph.D. thesis, School of Computer Science and Information Technology, University of Nottingham (2003)
14. Lopes, R., Girimonte, D.: The office-space-allocation problem in strongly hierarchized organizations. In: Cowling, P., Merz, P. (eds.) EvoCOP 2010. LNCS, vol. 6022, pp. 143–153. Springer, Heidelberg (2010)
15. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York (1990)
16. Pereira, R., Cummiskey, K., Kincaid, R.: Office space allocation optimization. In: IEEE Systems and Information Engineering Design Symposium (SIEDS 2010), pp. 112–117 (2010)
17. Ritzman, L., Bradford, J., Jacobs, R.: A multiple objective approach to space planning for academic facilities. Managament Science 25(9), 895–906 (1980)
18. Ülker, O., Landa-Silva, D.: A 0/1 integer programming model for the office space allocation problem. Electronic Notes in Discrete Mathematics 36, 575–582 (2010)