A Decision Support System for IT Security Incident Management

Gerhard Rauchecker, Emrah Yaşasın, and Guido Schryen

Department of Management Information Systems, University of Regensburg,
Universitätsstraße 31, 93053 Regensburg, Germany
{gerhard.rauchecker,emrah.yasasin,guido.schryen}@wiwi.uni-regensburg.de
http://www.winfor.uni-regensburg.de/Home/index.html.en

Abstract. The problem of processing IT security incidents is a key task in the field of security service management. This paper addresses the problem of effectively assigning and scheduling security incidents to the members of the IT staff. To solve this problem, we propose an innovative approach to assign staff members to security incidents by applying mathematical programming to the field of IT security management. We formulate an optimization model and propose efficient solution methods. The numerical simulations show that our approach improves current best practice behaviour significantly.

Keywords: IT Security Incidents, IT Security Services, Decision Support System, Heuristics, Optimization, Computational Experiment

1 Introduction

According to a report by [3], Dun & Bradstreet refered that 59 % of Fortune 500 companies experience at least 1.6 hours of downtime of IT systems per week (about 83 hours per year). To illustrate the dimension of potential costs, [3] gives the following example: "Assume that an average Fortune 500 company has 10,000 employees who are paid an average of $56 per hour, including benefits ($40 per hour salary + $16 per hour in benefits). Just the labour component of downtime costs for such a company would be $896,000 weekly, which translates into more than $46 million per year." From these figures it can be concluded that severe downtimes of IT systems often cost a significant amount of money.

Therefore, IT security incidents or outages of possibly different types (e.g. server failure due to hijacking) require processing as soon as possible by the IT staff members. Thus, effective and efficient scheduling of their staff is regarded as one of the critical tasks for an organization's IT support. Interestingly, this challenge has only rarely been addressed in the literature (see section 2).

We address this identified research gap and propose an innovative approach by introducing methods of operations research (OR) to solve questions arising in the field of IT security management. To the best of our knowledge, there is no work including the powerful methods of OR to IT security management, although this is a very promising symbiosis in our opinion. Hence, we propose an

optimization model for optimally assigning and scheduling security incidents to IT staff members. A strong advantage of our approach is its widespread applicability, because the model can also be established in general IT incident management frameworks which are not security related. We show that our approach improves current best practice significantly. When designing our model, we were highly influenced by ITIL [6]. In ITIL there are several processes defined, one of them being "Incident Management" that manages the life cycle of all incidents. The main purpose of incident management is to return the IT service as soon as possible. In this investigation, we concentrate on the "Incident Resolution Time" that is the average time for resolving an incident grouped into severity priorities. Thereby, we focus on a first level support. Our paper contributes to the usability of security services in the form of decision analytics in the area of IT security management.

The paper is organized as follows: the next section introduces related work. The third section outlines the methodology. We propose our optimization model of the decision support problem and develop two heuristic algorithms to solve this model efficiently. In section 4, we explain our computational experiments and present our results in section 5. Finally, we outline our contribution and close with an outlook on future research.

2 Related Work

When researching about IT incident management, three main directions can be identified: conceptual, prototypical and quantitative approaches. Lots of investigations are conceptual like ITIL, CoBIT and other frameworks for incident management [4, 6, 7, 8]. The prototypical approaches describe the development and prototypic implementation of a documentation system for IT incidents. In [9], occurring IT incidents are documented within an prototypic implementation for saving efforts for the employees involved and supporting the adaptability of the resulting system. [11], for instance, present an approach to diagnose application incidents by effectively searching relevant co-occurring and re-occcurring incidents.

Regarding the context of quantitative approaches, there is only limited work existing. An algorithm to assign IT incidents is provided by [10]. Main shortcomings of this approach are that it neither takes account for the fact that incidents may have different levels of severity, nor give they a benchmark of their algorithm. Although [12, 13] consider an assignment of developers in the context of bug fixing and feature developement, they do not solve a combinated assignment and scheduling problem, which would be a more complicated problem class. In [1, 2], semi-automated approaches for the assignment of bug reports to a developer are considered using machine learning algorithms based on text classification. However, these approaches need "an open bug repository for some period of time from which the patterns of who solves what kinds of bugs can be learned" [2] or a lot of contextual knowledge [1] which, in practice, both often

do not exist. Hence, these investigations are probably suitable when focussing on large open source projects.

Thus, we integrate the quantitative methods of OR by giving a mathematical programming formulation for the assignment and scheduling of security incidents to IT staff members. We further develop and computationally evaluate heuristics to solve the mathematical program efficiently.

3 Optimization Model and Heuristics

In this section, we propose a linear programming model for optimally assigning and scheduling trouble tickets to members of the IT support, mainly influenced from [15] who propose a mathematical program for the allocation and scheduling of rescue units in the aftermath of natural disasters. In our formulation, a set of n tickets is available at time zero and tickets are assigned to staff members by one central agent at time zero. Tasks are non-preemptive, i.e., if a staff member starts to process an incident, he has to finish this incident without interruption. For tickets with similar resolution times, it is more convenient to solve tickets with higher priority first. Therefore, we introduce ticket weights (see subsection 3.2) and minimize the total weighted completion time of all tickets. Finally, this section closes with the proposition of two heuristic algorithms to solve the optimization model efficiently.

3.1 Dynamics of Ticket Occurrence

The fact that all tickets are available at time zero, while in real world scenarios we have a dynamic situation where tickets occur at different points of time, seems to be a shortcoming of our approach. But this, in fact, is not true and we will give a simple example why it can be better to wait a certain time until a bunch of tickets has arrived instead of myopically assigning tickets to staff members at the time of their occurrence. One reason surely is that often all staff members are currently occupied and therefore there is no loss of time when collecting a set of tickets before assigning them to the staff members. But even in the case that some of the staff members are ready to process a ticket immediately, it can be better (by means of the total weighted completion time) to collect some tickets and then assign them to the staff. We give a simple example of that by considering a scenario where the staff consists only of one member.

Assume that the first ticket arrives at time 0. The ticket has a moderate priority and a corresponding ticket weight $w_1 = 4$. The response time is $p_{\overline{t_1}} = 5$, the resolution time is $p_{t_1} = 20$ and the ticket has no setup time (remote ticket). The second ticket arrives at time 10. It has a critical priority and a corresponding ticket weight $w_2 = 16$. The response time is $p_{\overline{t_2}} = 5$, the resolution time is $p_{t_2} = 30$ and the ticket has no setup time (remote ticket). If we assign the first ticket immediately to the staff, then he needs a total time of 25 to respond to and solve

the ticket. After that[1], he can respond to and solve the second ticket. Therefore, we have a total weighted completion time of $4 \cdot 25 + 16 \cdot (25 - 10 + 35) = 900$. If we wait until the occurrence of the second ticket and choose the schedule (*respond to second ticket*, *solve second ticket*, *respond to first ticket*, *solve first ticket*), we get a total weighted completion time of $16 \cdot 35 + 4 \cdot (35 + 25) = 800$. Even if we add up the waiting time 10 for the first ticket with its weight 4, we still get a better total weighted completion time of 840 in the second schedule. Of course, this example can be adopted to much more complex scenarios.

Furthermore, we account for both the dynamics of the situation and the need for timely decisions by suggesting that the optimization model is applied in an iterative manner: if the central agent determines to update the current assignment and scheduling plan based on new incoming tickets, a new instance of the optimization problem is created and solved taking into account current assignments.

3.2 Model Setup

Before introducing our mathematical program, we have to give the definitions which are necessary for its formulation.

We have a set of staff members $K := \{1, ..., m\}$, a set of tickets $\{1, ..., n\}$, a set of incidents $T := \{t_1, \ldots, t_n\}$ that represent the solutions of the tickets and a set of incidents $\overline{T} := \{\overline{t_1}, \ldots, \overline{t_n}\}$ which represent the responses to the tickets. Furthermore, we use fictitious incidents t_0 and t_{n+1} for modeling purposes. We will further need the sets $I_0 := \{t_0\} \cup T \cup \overline{T}$, $I_{n+1} := T \cup \overline{T} \cup \{t_{n+1}\}$ and the set of real (non-fictitious) incidents $I_{real} := T \cup \overline{T}$.

Next, we will propose our models decision variables. For incidents $\alpha \in I_0$, $\beta \in I_{n+1}$ and a staff member $k \in K$ we define

$$X^k_{\alpha\beta} := \begin{cases} 1, \text{ if } \alpha \text{ is executed directly before } \beta \text{ by } k \\ 0, \text{ else} \end{cases}$$

$$Y^k_{\alpha\beta} := \begin{cases} 1, \text{ if } \alpha \text{ is executed before } \beta \text{ by } k \\ 0, \text{ else} \end{cases}$$

In the following, we introduce the parameters used in our formulation. For a ticket $j = 1, \ldots, n$ and a staff member $k \in K$, $p^k_{t_j} \in \mathbb{R}_{\geq 0}$ (respectively $p^k_{\overline{t_j}} \in \mathbb{R}_{\geq 0}$) represents the time required by k to solve ticket j (respectively to respond to ticket j). Similarly, the parameters s^k_γ for $k \in K$ and $\gamma \in I_{real}$ denote the setup times, i.e., the time required to reach the location where the incident can be processed. Of course, this time is 0 if γ is a response to a ticket or a solution that can be executed remotely which means that the ticket can be solved by

[1] Note that it would be possible to respond to the second ticket between the response to and the solution of the first ticket, but because of the non-preemptiveness of tasks, the staff member cannot respond to the second ticket before time 25, because the solution of the first ticket happens in the time window between time 5 and time 25 while the second ticket occurs at time 10.

a staff member directly from its workplace. Note that the setup times do not depend on the incident which was processed before γ, because we assume that the staff member always returns to his office to catch the next incident from his computer. We also use a capability parameter cap_j^k which is 1 if staff member k is capable of processing ticket j and 0 otherwise.

We further have different priority levels (e.g. critical, high, medium, low, very low) that indicate the priority of the tickets. For each ticket $j = 1, \ldots, n$ we have a corresponding ticket weight w_j, a target response time r_j^{max} and a target resolution time c_j^{max}, each of them depending solely on the priority level of the ticket. Such times were suggested by ITIL [6] for example. The ticket weights w_j, assigned by the central agent, are based on the priority level which means the more urgent the priority level is, the higher is its ticket weight. The target response time represents the maximum acceptable time until the beginning of the response to a ticket whereas the target resolution time denotes the maximum acceptable time until the end of the solution of a ticket.

3.3 Model

With the above notations we can introduce the following linear programming model.

$$\min_{X,Y} \sum_{j=1}^{n} w_j \sum_{k=1}^{m} \left(\sum_{\alpha \in I_0} (p_{t_j}^k + s_{t_j}^k) X_{\alpha t_j}^k + \sum_{\gamma \in I_{real}} (p_\gamma^k + s_\gamma^k) Y_{\gamma t_j}^k \right) \tag{1}$$

$$s.t. \sum_{\alpha \in I_0} \sum_{k=1}^{m} X_{\alpha\gamma}^k = 1, \quad \gamma \in I_{real} \tag{2}$$

$$\sum_{\beta \in I_{n+1}} X_{t_0\beta}^k = 1, \quad k = 1, ..., m \tag{3}$$

$$\sum_{\alpha \in I_0} X_{\alpha\gamma}^k = \sum_{\beta \in I_{n+1}} X_{\gamma\beta}^k, \quad \gamma \in I_{real}; k = 1, ..., m \tag{4}$$

$$\sum_{k=1}^{m} \sum_{\gamma \in I_{real}} Y_{\gamma\gamma}^k = 0 \tag{5}$$

$$Y_{\alpha\gamma}^k + Y_{\gamma\beta}^k - 1 \leq Y_{\alpha\beta}^k, \quad \alpha \in I_0; \beta \in I_{n+1}; \gamma \in I_{real}; k = 1, ..., m \tag{6}$$

$$X_{\alpha\beta}^k \leq Y_{\alpha\beta}^k, \quad \alpha \in I_0; \beta \in I_{n+1}; k = 1, ..., m \tag{7}$$

$$\sum_{\gamma \in I_0} X_{\gamma\beta}^k + \sum_{\gamma \in I_{n+1}} X_{\alpha\gamma}^k \geq 2 \cdot Y_{\alpha\beta}^k, \quad \alpha \in I_0; \beta \in I_{n+1}; k = 1, \ldots, m \tag{8}$$

$$\sum_{k=1}^{m} \left(\sum_{\alpha \in I_0} (p_{t_j}^k + s_{t_j}^k) X_{\alpha t_j}^k + \sum_{\gamma \in I_{real}} (p_\gamma^k + s_\gamma^k) Y_{\gamma t_j}^k \right) \leq c_j^{max}, \quad j = 1, \ldots, n \tag{9}$$

$$\sum_{k=1}^{m} \sum_{\gamma \in I_{real}} p_\gamma^k Y_{\gamma \overline{t_j}}^k \leq r_j^{max}, \quad j = 1, \ldots, n \tag{10}$$

$$\sum_{\beta \in I_{n+1}} \left(X^k_{t_j \beta} + X^k_{\overline{t_j} \beta} \right) \leq 2 \cdot cap^k_j, \quad j = 1, \ldots, n \tag{11}$$

$$\sum_{k=1}^{m} Y^k_{\overline{t_j} t_j} = 1, \quad j = 1, \ldots, n \tag{12}$$

$$X^k_{\alpha\beta}, Y^k_{\alpha\beta} \in \{0, 1\}, \quad \alpha \in I_0; \beta \in I_{n+1}; k = 1, ..., m \tag{13}$$

The objective function (1) aims at minimizing the total weighted completion time. Constraint (2) guarantees that for each real incident there is exactly one incident processed immediately before. Constraint (3) ensures that each staff member k starts with the fictitious incident t_0 and then processes some incident β. Constraint (4) indicates for every staff member k that if there is an immediate predecessor for a specific real incident, there must be an immediate successor as well. Constraint (5) prohibits loops and constraint (6) is a transitivity constraint which assures that a staff member k, who processes incident α before a real incident γ and γ before incident β, also processes α before β. Constraint (7) means that any immediate predecessor is also a general predecessor. Constraint (8) secures that if a staff member k processes an incident α before an incident β, there has to be an incident which is processed by k immediately before β and an incident which is processed by k immediately after α. Constraints (9) and (10) list the target times of the tickets. Constraint (11) guarantees that tickets are only assigned to staff members that have the ability to solve them. Constraint (12) assures that before solving a ticket, the same staff member k has to respond to this.

3.4 Heuristics

The problem stated in this paper is computationally intractable and NP-hard. We will briefly explain this. If we drop a) the assumption that the response to a ticket has to be performed before the solution of a ticket and b) the target time constraints, we get a problem which is more simple than our problem but still a strong generalization of the 2-machine identical parallel machine scheduling problem with total weighted completion time as the objective function, which is known to be NP-hard. This fact has been proven by [5]. Furthermore, even for moderate instance sizes (e.g. 20 tickets and 10 staff members), we were not able to obtain optimal solutions within 12 hours. Therefore, we need to develop efficient solution heuristics to apply our approach in practical contexts.

Greedy Heuristic. Greedy heuristics are an often used technique in the context of IT incidents assignment in various contexts (e.g. [12, 13]). These heuristics make decisions about the construction of a solution based on local considerations such as preferring the choice that gives immediate best reward. In our context, this is a "first-come-first-serve" technique which is also current best practice to assign IT incidents to staff members [16]. The first heuristic we present is therefore a greedy heuristic, referred by **Greedy**, that models current best practice. The heuristic first sorts the tickets in descending order of their ticket weights and

then consecutively assigns the tickets to that staff member who has the shortest queue by means of current completion time. The response and resolution of the current ticket are both assigned in this succession to the end of that staff members current queue. The pseudocode is presented in table 1.

1	sort the incidents in descending order of their ticket weights $w_1 \geq \ldots \geq w_n$
2	initialize the current completion time $curr_k := 0$ and the current schedule
	$\sigma_k := \emptyset$ of every staff member $k \in K$
3	**for** $i = 1, \ldots, n$ **do**
4	define the set of feasible staff members to process i by
	$K^* := \{k \in K \mid cap_i^k = 1 \wedge curr_k \leq r_i^{max} \wedge curr_k + p_{t_i}^k + s_{t_i}^k + p_{t_i}^k \leq c_i^{max}\}$
5	**if** $K^* \neq \emptyset$ **then**
6	choose staff member k^* with the lowest current completion time
	$k^* := \arg\min_{k \in K^*}\{curr_k\}$ to response to and solve ticket i
7	update $curr_{k^*} := curr_{k^*} + p_{t_i}^{k^*} + s_{t_i}^{k^*} + p_{t_i}^{k^*}$ and $\sigma_{k^*} := (\sigma_{k^*}, \overline{t_i}, t_i)$
8	**else return** infeasible
9	**endfor**
10	**return** the list of feasible schedules $(\sigma_1, \ldots, \sigma_m)$

Table 1: Greedy pseudocode

Scheduling Heuristic. Although **Greedy** models best practice behaviour, it follows a very myopic assignment rule, because the staff member selection does not depend on the specific response times, the setup times or the resolution times of a ticket. Therefore, we suggest a scheduling heuristic, referred by **Sched**, which takes account for all of these times. We developed this heuristic based on the best performing algorithm in [14], referred to there as "Heuristic algorithm 7", which addresses a related problem from the scheduling literature. The pseudocode of the **Sched** heuristic is presented in table 2.

The main idea of this procedure is to select that pair of remaining tickets and staff members that minimizes the ratio of the total time to complete the ticket and its corresponding ticket weight. The response to that ticket (or the solution if a response has been given yet) is added to the end of the queue of the selected staff member. The selection criterion also takes care of the fact that before solving a ticket, a response to this ticket has to be performed by the same staff member (lines 8 and 14). We take account for keeping the timeframes (see table 4) for both the response to and the solution of a ticket with the use of the while-loop in lines 12 to 22. This loop addresses the case that the current assignment (i^*, k^*) makes it impossible to keep the timeframes for the remaining tickets. In this case we drop the current assignment (by setting $check := 1$ in line 13) and process urgent tickets first which are risky to miss their target times. In lines 15 to 21 (analogously in lines 23 to 29) we apply the current assignment by responding to the selected ticket or by solving the ticket (if the response has been given yet). At this point, we update the current completion time and the

current schedule of the selected staff member and add a response marker to the ticket (or remove the ticket from the remaining tickets if the ticket has been solved right now).

1	initialize the current completion time $curr_k := 0$, the updated completion
	time $\widetilde{curr_k}$ and the current schedule $\sigma_k := \emptyset$ of every staff member $k \in K$
2	initialize the remaining tickets $RT := \{1, \ldots, n\}$, the tickets yet
	responded to $Resp := \emptyset$ and a check parameter $check := 0$
3	**while** $RT \neq \emptyset$ **do**
4	reset $check := 0$
5	define the set of possible combinations by
	$F := \{(i,k) \in RT \times K \mid cap_i^k = 1\}$
6	**if** $F \neq \emptyset$ **then**
7	set $F_{res} := \{(i,k) \in F \mid i \notin Resp\}$ and
	$F_{sol} := \{(i,k) \in F \mid$staff member k has yet responded to ticket $i\}$
8	select the current ticket i^* and its processing staff member k^* as
	the best argument from the two minimization problems
	$\min_{(i,k)\in F_{res}} \left\{ \frac{curr_k + p^k_{t_i} + s^k_{t_i} + p^k_{t_i}}{w_i} \right\}$ and $\min_{(i,k)\in F_{sol}} \left\{ \frac{curr_k + s^k_{t_i} + p^k_{t_i}}{w_i} \right\}$
9	set $\widetilde{curr_k} := curr_k$ for all $k \in K$
10	**if** $(i^*, k^*) \in F_{res}$ **then** update $\widetilde{curr_{k^*}} := \widetilde{curr_{k^*}} + p^{k^*}_{t_{i^*}}$
11	**else** update $\widetilde{curr_{k^*}} := \widetilde{curr_{k^*}} + s^{k^*}_{t_{i^*}} + p^{k^*}_{t_{i^*}}$ **endif**
12	**while** exists $(i,k) \in F_{sol}$ with $\widetilde{curr_k} + s^k_{t_i} + p^k_{t_i} > c_i^{max}$ **or**
	exists $i \notin Resp$ with $\min_{k\in K \mid cap_i^k = 1} \{\widetilde{curr_k}\} > r_i^{max}$ **do**
13	set $check := 1$
14	select the current ticket i^* and its processing staff member k^* as
	the best argument from the two minimization problems
	$\min_{(i,k)\in F_{sol}} \{c_i^{max} - (curr_k + s^k_{t_i} + p^k_{t_i})\}$ (to select (i^*, k^*)) and
	$\min_{i \notin Resp} \max_{k\in K \mid cap_i^k = 1} \{r_i^{max} - curr_k\}$ (to select i^*)
	along with $\min_{k\in K \mid cap^k_{i^*} = 1} \{curr_k\}$ (to select k^*)
15	**if** $i^* \notin Resp$ **then**
16	**if** $curr_{k^*} > r_{i^*}^{max}$ **then return** infeasible **endif**
17	update $Resp := Resp \cup \{i^*\}$, $\sigma_{k^*} := (\sigma_{k^*}, \overline{t_{i^*}})$,
	$F_{sol} := F_{sol} \cup \{(i^*, k^*)\}$ and $curr_{k^*} := curr_{k^*} + p^{k^*}_{t_{i^*}}$
18	**else**
19	update $RT := RT \backslash \{i^*\}$, $\sigma_{k^*} := (\sigma_{k^*}, t_{i^*})$,
	$F_{sol} := F_{sol} \backslash \{(i^*, k^*)\}$ and $curr_{k^*} := curr_{k^*} + s^{k^*}_{t_{i^*}} + p^{k^*}_{t_{i^*}}$
20	**if** $curr_{k^*} > c_{i^*}^{max}$ **then return** infeasible **endif**
21	**endif**
22	**endwhile**
23	**if** $check = 0$ **then**
24	**if** $i^* \notin Resp$ **then**
25	update $Resp := Resp \cup \{i^*\}$, $\sigma_{k^*} := (\sigma_{k^*}, \overline{t_{i^*}})$ and

	$curr_{k^*} := curr_{k^*} + p^{k^*}_{t_{i^*}}$
26	**else**
27	update $RT := RT \backslash \{i^*\}$, $\sigma_{k^*} := (\sigma_{k^*}, t_{i^*})$ and
	$curr_{k^*} := curr_{k^*} + s^{k^*}_{t_{i^*}} + p^{k^*}_{t_{i^*}}$
28	**endif**
29	**endif**
30	**else return** infeasible
31	**endwhile**
32	**return** the list of feasible schedules $(\sigma_1, \ldots, \sigma_m)$

Table 2: Sched pseudocode

4 Computational Experiments

In order to evaluate the performance of our **Sched** heuristic, we investigated different problem sizes and randomly generated 10 instances per size. Our aim is to document the improvement of current best practice, which was modeled as **Greedy** heuristic. In order to reach this, we calculated the solutions of both heuristics in all instances.

We generated scenarios with 5, 10, 20, 40, 60 and 80 staff members, which should cover medium sized companies as well as large enterprises. We assume that there are at least as many and at most twice as many tickets as staff members. All times are expressed in minutes. The priority levels of a ticket can take one of the values *critical*, *high*, *medium*, *low* and *very low*. The target times for each priority level are presented in table 4 and are inspired by ITIL [6]. We have chosen normal distributions for generating the time parameters in our instances which is a common used approach in the academic literature, e.g. [15]. The response and setup times seem to be independent from the specific priority level of a ticket and are presented in table 3. We have chosen a 50% probability for a ticket to be solved remotely. Otherwise the normal distribution for the setup time is applied.

Input parameter	**Value, distribution**
Setup times	$s^k_j \sim N(5,1)$ or $s^k_j = 0$ if remote
Response times	$p^k_{t_j} \sim N(5,1)$

Table 3: Data independent from priorities

In contrast to the response and setup times, the solution times are indeed dependent on the specific priority level, see table 4. This is obvious because a critical ticket, such as a server blackout, tends to require more time to be solved than a lower priority ticket, such as a crashed virus scanner at a single workplace.

Priority level	**critical**	**high**	**moderately**	**low**	**very low**
Resolution times $p^k_{t_j} \sim$	$N(30, 10)$	$N(25, 10)$	$N(20, 10)$	$N(15, 10)$	$N(10, 10)$
Occurrence probability	5%	10%	15%	30%	40%
Corresp. ticket weight	16	8	4	2	1
Target response time	0	10	60	240	1440
Target resolution time	60	240	480	1440	10080

Table 4: Data dependent from priorities

We also consider the fact that tickets with a higher priority tend to occur more seldomly than tickets with a lower priority, see table 4. The probalibility of a staff member to be capable of processing a certain ticket was set to 50%.

5 Results

In this section, we evaluate the results drawn from our computational experiments. The results are presented in table 5. The first and fourth row contain the ratio #Staff/#Tickets of the number of staff members to the number of tickets. For every instance size, we average the ratios $SchedSol_i/GreedySol_i$ for all ten instances $i = 1, \ldots, 10$, which results in the values $Sched/Greedy$ listed in rows 2 and 5 (where $SchedSol_i$ and $GreedySol_i$ denote the total weighted completion time of the **Sched** and the **Greedy** heuristic solution for instance i). The numbers *CoeffVar* stand for the coefficients of variance of the ratios $(SchedSol_i/GreedySol_i)_{i=1,\ldots,10}$ and are a measure for robustness.

#Staff/#Tickets	5/5	5/10	10/10	10/15	10/20	20/20	20/30	20/40	
Sched/Greedy	0.91	0.86	0.72	0.68	0.66	0.61	0.59	0.56	
CoeffVar	0.15	0.15	0.15	0.13	0.08	0.09	0.10	0.06	
#Staff/#Tickets	40/40	40/60	40/80	60/60	60/90	60/120	80/80	80/120	80/160
Sched/Greedy	0.52	0.49	0.49	0.46	0.47	0.44	0.40	0.43	0.43
CoeffVar	0.10	0.10	0.06	0.11	0.06	0.07	0.10	0.07	0.06

Table 5: Results of computations

The computational results show that our developed heuristic **Sched** improves the current best practice behavior (modeled as **Greedy** heuristic) from 9% up to 60%. In larger companies, with an IT support consisting of 20 or more staff members, the improvement is even greater than 39%. The reason for the **Sched** heuristic being more dominant over the **Greedy** heuristic in larger instance sizes is explained in the following. If we have a small IT support with about 5 or 10 staff members, the number of employees that can solve a specific ticket is rather low. This number increases with the size of the IT support staff and therefore the response, setup and solution times of a ticket, that depend not only on tickets

but also on the staff members, become more relevant. This is the point where our **Sched** heuristic performs much better than the **Greedy** heuristic which does not account for any of these times. The coefficient of variation is a measure for the relative dispersion of data. In our context, this parameter describes the average percentile deviation of the ratios $SchedSol_i/GreedySol_i, i = 1, \ldots, 10$, from the mean value Sched/Greedy. This relative spread reaches from 6% in the best to 15% in the worst scenario. These values are thus at a good range. Further, runtimes show that the developed **Sched** heuristic needs only few seconds in all tested cases and is thus very applicable in practice.

6 Conclusion and Future Work

In this paper, we address the problem of effectively assigning IT security incidents to IT staff members, which is a crucial task in IT security management. First, we introduced a mathematical programming model to optimally assigning and scheduling these incidents. By doing this, we bridged the gap between the quantitative methods of OR and the field of IT security management. Although we could not solve relevant scenarios optimally due to the model complexity, we showed the practical applicability of our approach by developing efficient solution heuristics. Second, we showed that our **Sched** heuristic improves current best practice, modeled as **Greedy** heuristic, by up to 60% and at the same time can be used in practice because of the very low execution times of the algorithm.

As future work, we are going to extend our approach by considering dependencies between the tickets. For instance, some tickets cannot be solved before having solved other tickets first. We will also expand our approach to higher support levels. Further, our current approach assumes to wait a certain time until a bunch of tickets has arrived. This tradeoff between waiting time and immediate assignment requires further research with real data. In further researches, the assumption that tasks are non-preemptive can be dropped in order to pause the task when appropriate. For example, this is likely to become necessary when all staff members are currently occupied and a critical ticket arrives. Finally, we will develop other heuristics and adopt metaheuristics to cover these extensions and to gain further benchmarks for the quality of the heuristics.

Acknowledgments. The research leading to these results was supported by the "Bavarian State of Ministry, Education, Science and the Arts" as part of the FORSEC research association (`https://www.bayforsec.de`) and by the "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (`http://www.secbit.de`).

References

[1] Anvik, J.: Automating Bug Report Assignment. In: ICSE '06 Proceedings of the 28th International Conference on Software Engineering. pp. 937–940 (2006)

[2] Anvik, J., Hiew, L., Murphy, G.: Who should fix this bug? In: ICSE '06 Proceedings of the 28th International Conference on Software Engineering. pp. 361–370 (2006)
[3] Arnold, A.: Assessing the Financial Impact of Downtime. Vision Solutions (White Paper) (2010), http://www.strategiccompanies.com/pdfs/Assessing%20the%20Financial%20Impact%20of%20Downtime.pdf
[4] Bernard, P.: COBIT 5 - A Management Guide. Van Haren Publishing (2012)
[5] Bruno, J., Coffman Jr., E.G., Sehti, R.: Scheduling Independent Tasks to Reduce Mean Finishing Time. Communications of the ACM 17(7), 382–387 (1974)
[6] Cabinet Office, Steinberg, R., Rudd, C., Lacy, S., Hanna, A.: ITIL Service Operation, 2nd edition. TSO, London (2011)
[7] Cichonski, P., Millar, T., Grance, T., Scarfone, K.: Computer Security Incident Handling Guide. National Institute of Standards and Technology Special Publication 800-61, Revision 2 (2012)
[8] ISO/IEC: ISO/IEC 27035 - Information Technology - Security Techniques - Information Security Incident Management (2011)
[9] Kurowski, S., Frings, S.: Computational Documentation of IT Incidents as Support for Forensic Operations. In: Proceedings of the 2011 Sixth International Conference on IT Security Incident Management and IT Forensics. pp. 37–47. IEEE Computer Society, Washington, DC, USA (2011)
[10] Li, X., Zhan, Z., Guo, S., Zhang, L.: IT Incident Assign Algorithm Based on the Difference Between Support Groups. In: International Conference on Advanced Intelligence and Awarenss Internet (AIAI). pp. 319–323 (2010)
[11] Liu, R., Lee, J.: IT Incident Management by Analyzing Incident Relations. In: Proceedings of the 10th international conference on Service-Oriented Computing (ICSOS). pp. 631–638. Springer Verlag, Berlin, Heidelberg (2012)
[12] Rahman, M., Ruhe, G., Zimmermann, T.: Optimized Assignment of Developers for Fixing Bugs: An Initial Evaluation for Eclipse Projects. In: IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 439–442 (2009)
[13] Rahman, M., Sohan, S.M., Maurer, F., Ruhe, G.: Evaluation of Optimized Staffing for Feature Development and Bug Fixing. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (2010)
[14] Weng, M.X., Lu, J., Ren, H.: Unrelated Parallel Machine Scheduling with Setup Consideration and a Total Weighted Completion Time Objective. International Journal of Production Economics 70(3), 215–226 (2001)
[15] Wex, F., Schryen, G., Feuerriegel, S., Neumann, D.: Emergency Response in Natural Disaster Management: Allocation and Scheduling of Rescue Units. European Journal of Operational Research 235(3), 697 – 708 (2014)
[16] Zitek, N.: ITIL Incident Management - How to separate roles at different support levels. ITIL & ISO 20000 Blog (2013), http://www.20000academy.com/Blog/November-2013/ITIL-Incident-Management-How-to-separate-roles-at-different-support-levels