

Convex optimization for shift scheduling in retail

Timo Räsänen

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 22.10.2022

Supervisor

Prof. Jukka Suomela

Advisors

PhD Paul Saikko

MSc Laur Norjama

Copyright © 2022 Timo Räsänen



Author	Timo Räsänen	
Title	Convex optimization for shift scheduling in retail	
Degree programme	Computer, Communication and Information Sciences	
Major	Computer science	Code of major SCI3042
Supervisor	Prof. Jukka Suomela	
Advisors	PhD Paul Saikko, MSc Laur Norjama	
Date	Number of pages 51+13	Language English

Abstract

Retail is a turbulent sector characterized by long opening hours, a multitude of products and services, varying employee availabilities and various contractual constraints. In retail workforce optimization we want to match the available workload to employees while minimizing task switches. In our problem formulation we have tasks that are interruptible and have different priorities and skill requirements. The scope of this thesis is the subproblem of evaluating the value of a set of daily shifts by assigning tasks to them. We show that this task assignment subproblem is NP-hard as it contains the interval scheduling problem with non-identical machines as a special case, which 3-SAT can be reduced to.

We have a pre-defined utility function for scoring the solutions and an existing baseline solution which we compare our results against. We also have 33 real problem instances and 100 artificially generated ones to run benchmarks on. We formulate and solve an integer program which optimizes the nonconvex quadratic utility function directly using the commercial solver GUROBI. This revealed that the baseline could be improved for the real world problem instances up to 4.2% (mean 0.7%) and for artificial instances up to 785.5% (mean 19.7%), with average runtimes being 451s and 5493s respectively.

Next we formulated a convex quadratic program which approximated the utility function. By optimizing this convex objective the results were still up to 4.2% (mean 0.6%) better than baseline on real instances and up to 401% (mean 7.1%) better on artificial instances. For this convex objective the average runtimes were 9.3s and 385s respectively.

To reduce runtime variance we removed the integer constraints and solved the model with the open source OSQP solver to get fractional solutions. For these fractional solutions we implemented an iterative rounding scheme, which improved the real instances up to 4.2% (mean 0.3%) and artificial instances up to 376% (mean 1%), with an average runtime of 2.4s and 16.3s respectively.

The convex relaxation could also be directly used as the utility function, as it also rewards consecutive task assignments. This would likely make the approach in this thesis perform even better.

Keywords convex optimization, employee scheduling, multi-activity assignment problem, mixed integer programming, combinatorial optimization

Författare	Timo Räsänen	
Titel	Konvex optimering för schemaläggning i detaljhandel	
Utbildningsprogram	Computer, Communication and Information Sciences	
Huvudämne	Computer science	Huvudämnets kod SCI3042
Övervakare	Prof. Jukka Suomela	
Handledare	PhD Paul Saikko, MSc Laur Norjama	
Datum	22.10.2022	Sidantal 51+13
		Språk Engelska

Sammandrag

Detaljhandel karaktäriseras av långa och varierande öppettider, flexibla arbetstider, flera produkter och olika tjänster. Då vi optimerar bemanningen i detaljhandeln vill vi para ihop tillgängligt arbete med personal och undvika överflödiga byten mellan arbetsuppgifter. I vårt problem är det möjligt att avbryta arbetsuppgifter och dessa uppgifter har olika prioritet och kompetenskrav. Detta arbete begränsar omfattningen till delproblemet, där vi vill värdera en samling av givna arbetspass genom att ange dem arbetsuppgifter. Vi visar att detta delproblem är NP-svårt, eftersom det innefattar ”intervall optimerings problemet” (interval scheduling problem) med icke-identiska maskiner som ett specialfall, vilket 3-SAT kan reduceras till.

Vi har en given nyttofunktion för att värdera lösningarna och det finns en bas lösning vilken vi jämför med våra resultat. Vi har 33 riktiga problemfall och 100 artificiellt genererade som vi använder till jämförelsen. Vi formulerar och löser ett heltalsprogram som optimerar den icke-konvexa kvadratiske nyttofunktionen direkt med hjälp av den kommersiella lösaren GUROBI. Detta visade att baslösningen kan förbättras i de riktiga fallen med upp till 4,2% (medeltal 0,7%) och i de artificiella fallen upp till 785,5% (medeltal 19,7%), med körtiderna i medeltal 451s respektive 5493s.

Sedan formulerade vi en konvex approximering för modellen. Optimering av denna modell ledde till resultat som var fortfarande upp till 4,2% bättre (medeltal 0,6%) för de riktiga fallen och 401% bättre (medeltal 7,1%) för de artificiella fallen. För denna modell tog optimeringen i medeltal 9,3s respektive 385s.

För att minska variansen i körtiderna tog vi bort heltalsrestriktionen och löste modellen med OSQP lösaren. För dessa bråktalslösningar gjorde vi olika iterativa avrundningsmetoder, varav den bästa förbättrade de riktiga fallen upp till 4,2% (medeltal 0,3%) och de artificiella fallen upp till 376% (medeltal 1%), med körtiderna 2,4s respektive 16,3s.

Denna konvexa omformulering kunde möjligtvis användas direkt som nyttofunktion, eftersom den också belönar på varandra följande arbetsuppgifter. I detta fall skulle metoden i detta arbete troligtvis prestera även bättre.

Nyckelord konvex optimering, schemaläggning, uppdragsproblem med flera aktiviteter, blandad heltalsprogrammering, kombinatorisk optimering, personalplanering

Preface

I would want to thank my employer for providing me an opportunity to do this thesis. Henrik Sjöström for introducing me to this interesting topic. My advisors Paul Saikko and Laur Norjama for providing encouraging feedback and always being available for discussion.

Finally I want to thank my supervisor Jukka Suomela for his support and keen interest in this topic.

Helsinki, 22.10.2022

Timo Räsänen

Contents

Abstract	3
Abstract (in Swedish)	4
Preface	5
Contents	6
Abbreviations	8
1 Introduction	9
1.1 Research question	10
1.2 Goals	10
1.3 Outline of this thesis	10
2 Background	11
2.1 Combinatorial optimization	11
2.2 Employee scheduling	11
2.2.1 Shift design and a multi-skilled workforce	14
2.2.2 Activities instead of tasks	15
2.3 Interval scheduling	16
2.4 Mathematical optimization	16
2.4.1 Linear optimization	17
2.4.2 Convex optimization	18
2.4.3 Integer optimization	20
2.4.4 Suboptimal solutions	22
3 The multi-activity assignment problem	23
3.1 Modelling the workload	23
3.2 Problem definition	24
3.3 Baseline solution	26
3.4 Shift placements	27
3.5 Measuring solution quality	27
3.5.1 Comparing solution quality	27
3.6 Problem analysis and special cases	28
3.7 Decomposition	29
4 Implementation	31
4.1 Integer program formulation	31
4.2 Task switch costs	32
4.2.1 Linear task switch costs	32
4.2.2 Quadratic task switch costs	33
4.3 Rounding schemes	34
4.3.1 Tick based rounding	35
4.3.2 N-best rounding	35

4.4	Extensions of the model	35
4.4.1	Modelling different types of tasks	35
4.4.2	Optimizing shift and break placement	36
5	Results	38
5.1	Data	38
5.1.1	Real instances	38
5.1.2	Artificial instances	38
5.2	Benchmark setup	38
5.3	Validation and optimal solutions	38
5.4	Fractional solutions and integrality gaps	41
5.5	Rounding results	42
5.6	Flexible shifts	42
6	Conclusion and future work	46
6.1	Future work	47
A	Appendix	52

Abbreviations

ISP	interval scheduling problem
MAAP	multi-activity assignment problem
DP	dynamic programming
DAG	directed acyclic graph
MIP	mixed integer program
QP	quadratic program
LP	linear program
3-SAT	boolean 3-satisfiability problem

1 Introduction

Retail is a turbulent sector characterized by long opening hours, a multitude of products and services, varying employee availabilities and various contractual constraints. It is important to keep the loyalty of customers by maintaining a good level of service as a single bad customer experience may lose the customer to a competitor. Employee costs in retail range from 10% to 20% of total gross revenue [21]. The trade-off between understaffing, impacting service quality, and overstaffing, impacting the bottom line needs to be carefully balanced. This makes designing and assigning good shifts to employees all the more important. Meanwhile the amount of data that is being collected from the retail processes is greater than ever. Access to sales forecasts and delivery schedules can give us unprecedented level of detail for estimating the workload at the store. Manual shift planning is no longer sufficient to leverage all this available data.

Typically there is one dedicated person, commonly a store manager, doing shift planning, usually on top of their other tasks. They strive to plan shifts which fulfill contractual obligations towards the employees, give them enough work and enough rest and breaks. All this while planning to have enough skilled labor at the store to do all the required tasks. This can be time consuming and often there is software with varying levels of automation and capabilities to help with the planning. While there are a multitude of software implementations available, there are always trade-offs involved related to model precision, flexibility and tractability. It is still an unsolved problem how to best balance these trade-offs.

On a high level a workforce optimization pipeline can be divided into four parts. First the workload is estimated based on sales forecasts and other sources, then the employees that will be working on each day will be determined for a longer time horizon, then for each day shifts with break placements are designed and finally activities are assigned to these shifts. This thesis will focus on the last part, which determines the assignment of multiple activities to each shift.

We will review scheduling in general and this task assignment problem specifically. Then we will implement a practical method to solve the problem. We will also explore the possibility to adjust the placement of shifts and breaks to improve the solution.

1.1 Research question

The problem of assigning multiple activities to shifts with a multi skilled workforce has been called the *multi-activity assignment problem* (MAAP) [35]. MAAP is a problem formulation that suits the retail context particularly well. In MAAP the planning horizon is discretized, usually into 15-minute intervals which we will be calling *ticks*. The activities need to be determined for each tick in a shift, so that excessive activity switches are avoided. We distinguish activities from tasks by the semantic that activities can have intermittent assignments and do not strictly need to be occupied for the full duration while tasks need to be completed from start to finish.

For determining the shift placements there is an existing solution which we will call the baseline solution. In this baseline solution shift placements are determined with a heuristic search procedure. To guide this search the quality of a shift placement solution needs to be determined, that is solving the MAAP either optimally or approximately. The baseline uses a greedy heuristic to do this.

There are three reasons we are interested in solving this problem. Firstly, solving it optimally tells us how well the existing baseline heuristic is approximating the utility function. Secondly, solving it better in a reasonable time can be used as a final polishing step to get a better solution. Thirdly, if a solution to this problem performs sufficiently fast it can be used to guide the search algorithm.

As the solution quality of MAAP is highly dependent on the shift placements we will also look for some options to improve the shift placements provided by the baseline solution.

1.2 Goals

The main objectives of this thesis are:

- **O1:** Analyze the problem complexity and special cases.
- **O2:** Implement an algorithm that solves the *multi-activity assignment problem* and compare it to the baseline solution.
- **O3:** Look for opportunities to improve the shift placements.

1.3 Outline of this thesis

Section 2 will provide background for scheduling problems and mathematical optimization. Section 3 will define our problem in more detail. Section 4 will describe our implementation used for solving this problem. Section 5 compares the results of our implementation to the baseline. Section 6 concludes the thesis by presenting our main results and suggests some future work that could be done on this problem.

2 Background

Scheduling is about allocating resources to tasks. The simplest type of scheduling is the assignment problem which looks for optimal one-to-one task-resource assignments. Traditional scheduling problems consider the processing times of tasks and try to fit as many tasks as possible to a schedule. Interval scheduling adds fixed starting times to the tasks. Finally employee scheduling is a more practically oriented and heterogenous field where the goal is to plan and assign shifts to employees so that the demand for work is met. We will review scheduling in this section but first we will introduce combinatorial optimization as a tool to model problems.

2.1 Combinatorial optimization

Combinatorial optimization is one form of discrete optimization which provides a general framework that can be used to define a wide variety of problems. The goal is to find a combination of decisions that is feasible and also having the best value in case of an optimization problem. Many real world problems can be modelled as a set of decisions. In our problem we can define the decisions as whether worker i is assigned task j at tick k . A schedule can then be built based on these decisions. In the most general case these are hard problems where the worst case complexity with an exhaustive search is $\mathcal{O}(2^n)$, where n is the number of decisions that need to be made.

Research into specific problems typically aims to find ways to exploit the structure of the problem so that exhaustive search of all possible inputs is not needed. We will review literature related to employee scheduling and interval scheduling to find what specialized algorithms have been used for these problems. We will be looking for instances of problems that are easy to solve and identify any similarities to our problem. Later we will review what general approaches are used when no specialized algorithms exist.

2.2 Employee scheduling

Employee scheduling is also called personnel scheduling, workforce scheduling or staff scheduling [49]. In employee scheduling we would like to assign employees to shifts or decide which employees will work on a particular day. The aim is to construct timetables for the staff of an organization so that workforce demand can be met while abiding to legal constraints and considering employee satisfaction. In 1954 Dantzig [13] formulated an integer program for the toll booth problem, in which shifts and breaks are designed with half-hour precision to meet the workload demand. The first comprehensive survey of personnel scheduling was done by Baker in 1976 [3]. There he defined the shift scheduling, days off scheduling and tour scheduling problems, where tour scheduling is a combination of the two former.

In the shift scheduling problem there is a set of predetermined shifts with a certain staff requirement and the objective is usually to find the minimum staff size required. In case of non-overlapping shifts it is trivial to find this optimum.

When shifts overlap the problem gets more interesting, especially when the overlap is structured. One example of such structured overlap exists in the so called Kleen City Problem [3]. In this problem we have consecutive 8-hour shifts where the first half of the shift overlaps with the previous shift and the last half overlaps with the next shift. With this construction the workforce requirements of a shift can potentially be fully met by the surrounding shifts and a closed form expression can be derived for the minimum staff size. The more general problem where there are a couple fixed shift options, typically morning, evening and night shifts, and a certain staff coverage that needs to be achieved is called the *nurse rostering problem*. Baker also defines the *general shift scheduling* problem with m different shift patterns and n periods with their corresponding staff requirement r_i . Shift patterns are defined by a_{ij} indicator variables which equal 1 when shift pattern j is active in period i . By assigning different costs c_j to the shift patterns an optimal staff allocation can be solved by an integer program with the following form:

$$\begin{aligned} \text{minimize} \quad & \sum c_j x_j \\ \text{s.t.} \quad & \sum a_{ij} x_j \geq r_i, \forall i \in \{1, 2, \dots, n\}. \end{aligned}$$

The minimum staff size can be solved when all costs c_j are 1. This is the same formulation used by Dantzig [13]. To summarize, the shift scheduling problem aims to minimize the number of staff by allocating them to predetermined shifts so that the workload requirements are satisfied, this requires solving an integer linear program in the most general case.

Usually employees cannot work every day. In this case we have a separate problem where we need to decide which days the employees should or should not work on. This is called *days off scheduling*. A typical example would be employees with a 5 day working week while the operation should be available 7 days a week. An optimal workforce size for this case would be $W_{\min} = \sum r_i / 5$, where r_i is the staff requirement for day i . This can be achieved if the daily employee requirement can be met exactly with full-time employees. Part-time employees provide more flexibility so that the daily requirement can be partially fulfilled with full-time employees and then part-time employees can be used to fill the remaining requirements. In the general case a similar integer program can be constructed as was the case in the *shift scheduling* problem by replacing the daily shift patterns with week patterns.

To expand this scheduling to multiple weeks the trivial approach would be to use the same assignments every week, but in most cases it is desirable to rotate the off days between employees. In this case, the week patterns can be rotated among the employees. However, it is common that there is a lower or upper limit on the number of consecutive days an employee may be working. Week patterns that guarantee certain maximum consecutive days can be constructed, but with general patterns this cannot be guaranteed.

Brucker et al. [8] has explored the complexity of some variations of tour scheduling. However, in general there is a lack of research on how certain problem characteristics impact the complexity [49].

Van der Bergh et al. [49] reviewed over 300 employee scheduling articles published

between 2004 and 2012. They categorized articles by how the objective is defined, constraints and flexibilities, solution methods and the application area.

The objective is defined by varying definitions of employees, shifts and decision types. Employees can be full-time or part-time, which impacts how flexible shifts can be designed, skills can be binary or they can reflect productivity levels and sometimes the personnel needs to be scheduled part of a crew. Hojati and Patil [20] explored the problem with a part-time workforce and formulated an integer linear program to construct a schedule. Their solution also incorporated different employee skills and tasks but each shift was assigned a single task only.

When workload demand is distributed evenly for the whole day multiple non-overlapping shifts can be used, typically this is an early shift, a late shift and a night shift. This kind of workload is common in hospitals. Overlapping shifts are used to meet demand peaks. Incorporating shift design, by allowing flexible shift starts and lengths, makes the problem more complex but allows matching the workload demand more closely. This is important in fields like retail where the workload varies hourly and daily.

Van der Bergh et al. [49] continued their categorization further based on what different *constraints* are used and how *flexibilities* are added to deal with them. Almost 75% of the problems they reviewed had a hard coverage constraint on the workload meaning that understaffing is not allowed. When personnel skills are modelled a hard constraint is commonly used to ensure the presence of each skill in the schedule. Hierarchical or seniority based skills are used when more experienced employees can be used to do the tasks of those with less experience but at a higher cost. They recommend considering flexible skills with corresponding training costs.

Breaks can be either scheduled in advance or ad hoc in real-time. Considering break flexibility as part of tour scheduling is very effective in improving labor utilization [22]. Thompson et al. [48] argues that while scheduling breaks in advance makes the problem more costly, it is worth it to avoid problems with real-time break scheduling. When breaks are not scheduled in advance this can result in both costly overstaffing to ensure that all breaks can be taken or understaffing as some breaks have to be taken when there is no surplus of workforce available. During periods of high demand it may be that breaks are not taken at all if they are not scheduled. They found that only 18% of the 64 papers they reviewed considered breaks in their schedules.

Van der Bergh et al. [49] continues that ensuring employee fairness is core part of most rostering problems. This is done by applying different balancing constraints on the problem. Some undesirable shift properties that need to be balanced are working weekends, early and late shifts, consecutive working days or number of shift requests that are not met.

The most commonly used solution techniques are integer programming, meta-heuristics, simulation and constraint programming. The integer programs are usually based on Dantzig's [13] formulation, as additional constraints or multiple objectives can easily be added. These programs often end up with a huge amount of variables which requires specialized techniques to solve. These techniques include decomposition, cut generation, column generation and branch-and-price. We will describe these

in more detail in Section 2.4.3. Metaheuristics like tabu search, genetic algorithms or simulated annealing can typically find some good feasible solutions in reasonable time, but there is no guarantees on the solution quality. Brusco et al. [9] formulated a highly flexible tour scheduling problem which they solved efficiently using simulated annealing. Simulation approaches like discrete event simulation and Monte Carlo simulation shine when a stochastic component is needed. It can also be more natural to model a real life phenomenon with a simulation approach. However, simulation shares the drawbacks of metaheuristics in having no solution guarantees. In addition they may be more unpredictable as some chaotic behavior may emerge.

Five major application areas were identified in the review: healthcare services, transportation, manufacturing, retail and military personnel scheduling. Mirrazavi and Beringer presented a workforce management solution used at Sainsburys Supermarkets Ltd [38]. Several implementations have also been done for general retail [11, 24, 41, 44, 53].

In their review, Van der Bergh et al. [49] identified a lack of integrating all the decisions of the personnel scheduling problem, such as forecasting and adjusting the workload distribution, break placements, hiring or firing, training skills and considering employee preferences for holidays or shifts. It is clear that the future of personnel scheduling will strive for an multi-objective model and to integrate as many of these decisions as possible into one single personnel scheduling problem.

2.2.1 Shift design and a multi-skilled workforce

Many of the more recent and more flexible approaches incorporate *shift design* as is the case in the minimum shift design problem (MSD) [40, 17]. Here the task is to define when shifts start and end including breaks. However, MSD does not consider what tasks are assigned to those shifts. Shift design or shift generation is needed when the demand we want to satisfy is not static, as is common in retail.

To leverage a multi-skilled workforce we can add one more level of detail to MSD by also assigning different tasks to the shifts. This task assignment problem has been called the *Personnel Task Scheduling Problem* (PTSP) or *Shift Minimization Personnel Task Scheduling Problem* (SMPTSP) [30, 47]. In PTSP the shift start and end times have been decided in advance, then tasks need to be assigned to these shifts taking into account employee skill. This is similar to the interval scheduling problem with non-identical machines which we will consider more closely in Section 2.3. It is common to use a two-phase heuristic approach to first design the shifts for employees and then assign tasks to them [17, 33].

Nurmi et al. [42] approached the problem in reverse by first designing shifts with tasks and only then assign staff to the shifts in the staff rostering process. Here tasks are not fixed in time and tasks may have shift-local precedence constraints. They also considered transition times between tasks which depend on the mode of transport used by the employee. The number of feasible shift-employee pairs is maximized. They called this the *General Task-based Shift Generation Problem* (GTSGP). They used the PEAST algorithm [32] to optimize the objective. The core of PEAST is a greedy hill climbing mutation algorithm which builds a neighborhood of feasible

solutions by searching for improvements to current solution by a sequence of moves. This neighborhood is then explored using a combination of tabu search, shuffling operators which help escaping local optima and an adaptive genetic penalty method which turns hard constraints into adaptively weighted soft constraints.

2.2.2 Activities instead of tasks

When the demand is given as a multi-activity workload profile we do not have clearly defined start and end times for tasks. Thus it is reasonable to allow arbitrary switches between these activities. However, it is still preferred to have continuous segments assigned to the same activity, how this is done is then one of the key differentiators in how this problem is approached.

Quimper and Rousseau [46] performed large neighborhood search to assign activities to shifts. They built the neighborhood with a context free formal language. They had a minimum assignment duration of 1 hour and each activity switch was accompanied by a break. Thus the number of breaks limit the amount of task switches.

Lequy et al. [35] described the *multi-activity assignment problem* (MAAP). Their activities have minimum and maximum assignment durations. The minimum durations prevent the assignment of too short segments of activities. They also allowed overstaffing an activity which may enable assignment of a longer continuous segment at the cost of over assigning that activity. They considered both activity and sequence-dependent transition costs. The objective is to minimize overstaffing, understaffing and transition costs. Employee qualifications for tasks were considered binary so no skill based task rewards were used. They proposed a MIP model with column generation where the decision variables are all the possible sequences of activities that can be assigned to a shift. For finding variables with negative reduced cost they formulated a shortest path subproblem in a graph representing all the activity transitions for a shift. In this graph both activity-dependent and sequence dependent transition costs can be included. The arc weights in this graph utilize the dual variables of the master problem to allow modelling the impact of selecting an activity on the overstaffing and understaffing costs. Later Lequy et al. [34] also included fixed tasks into the model and called it the *multi-activity and task assignment problem* (MATAP).

Pan et al. [43] had a minimum and maximum assignment duration. They first built a solution satisfying the workload with a greedy heuristic. Then they used tabu search to reduce the number of violated activity duration constraints. No under or overstaffing was allowed.

Cuevas et al. [12] incorporated tour scheduling into the problem and called it the multi-skilled tour scheduling problem. They formulated a MIP for this problem and modelled tasks switches by defining a base activity for each employee and penalizing any deviations from this base activity.

Yi and Timothy [45] also incorporated shift design with up to two week time horizon. They used a 1 hour minimum activity duration and allowed overassignment with a penalty. They also introduced some benchmark instances with up to 150

employees and 19 tasks.

2.3 Interval scheduling

The interval scheduling problem (ISP) is closely related to the *task assignment problem* in employee scheduling. In the ISP the goal is to assign the maximum amount of jobs to a machine. The jobs have a strict starting and processing times. A machine can not do two jobs at the same time. This differs from traditional scheduling problems where the job starting time is not fixed.

The basic ISP can be optimally solved with a greedy earliest finish time first $\mathcal{O}(n \log n)$ -time algorithm. This greedy algorithm extends naturally to multiple machines by assigning any overlapping tasks to the next available employee.

Some useful extensions to the basic ISP that have been explored are adding weights, restricting jobs to certain machines and having machines available only for a certain interval of time.

The weighted ISP for single machines can be solved with dynamic programming (DP) in $\mathcal{O}(n \log n)$ time. With multiple machines it can be solved as a totally unimodular integer program or a min-cost flow problem in $\mathcal{O}(n^2 \log n)$ time [2, 10].

The machines are called non-identical when they are restricted to certain jobs. With k machines Arkin and Silverberg [2] used a longest path $\mathcal{O}(V + E)$ -time algorithm in a DAG with $V = \mathcal{O}(n^k)$ and $E = \mathcal{O}(n^{k+1})$ giving an $\mathcal{O}(n^{k+1})$ -time algorithm. When k is not fixed this problem is NP-complete and 3-SAT can be reduced to it.

A further extension to ISP is to consider machines that are only available for a specific period of time. Brucker and Nordmann [7] called this the k -track assignment problem. It has also been called interval scheduling with machine availabilities (ISMA) [28]. This problem reduces to the circular arc graph coloring which is NP-complete. Brucker and Nordmann [7] designed a longest path $\mathcal{O}(n^{k+1})$ -time algorithm in a DAG for this problem.

Kovalyov et al. [29] presented a general formulation of different interval scheduling problems, their relation to graph problems, computational complexity and solution algorithms.

These ISP variations are summarized in Table A2.

2.4 Mathematical optimization

In mathematical optimization we want to find the global optimum of a function and the value for the optimization variable $\mathbf{x} \in \mathbb{R}^n$ at this point. Usually there is also some constraints for the solution which define the feasible set $\mathbf{x} \in S$. Thus even for a convex function it is not sufficient to only find the point where the derivative is 0, as this point may be outside the feasible set. In this section we will review linear optimization and the more general convex optimization. Finally we will explore integer optimization which restricts the feasible set to be integer values.

2.4.1 Linear optimization

Linear optimization or more commonly called *linear programming* (LP) was first invented by Leonid Kantorovich in 1939 and was used in optimizing the five-year plans in the Soviet Union [25]. Later Danzig extended and made popular the theory of LP and invented the simplex method for solving them [15]. Von Neumann contributed by introducing duality into the LP theory [14] based on Farkas lemma [18].

The canonical form of a LP is a packing problem where the objective is to

$$\begin{aligned} & \text{maximize} && \mathbf{c}^T \mathbf{x}, \\ & \text{s.t.} && \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & && \mathbf{x} \geq 0, \end{aligned} \tag{1}$$

where $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$. This primal packing problem has a dual covering problem where the objective is to

$$\begin{aligned} & \text{minimize} && \mathbf{b}^T \mathbf{y}, \\ & \text{s.t.} && \mathbf{A}^T \mathbf{y} \geq \mathbf{c}, \\ & && \mathbf{y} \geq 0. \end{aligned} \tag{2}$$

The dual problem provides an upper bound for optimum value of the primal problem, which is called *weak duality*. The difference between the optimum value of the primal and dual problems is called the *duality gap*. For LPs the duality gap is 0 in which case we have *strong duality*.

To construct a dual linear program each variable in the primal problem becomes a constraint and each constraint becomes a variable. We can say that in the dual linear program we are optimizing the constraints of the primal problem. Thus the dual variables are often called *shadow prices* in practical optimization problems. The shadow price signifies the change in the optimum value for each unit of change in the constraint. We say that an inequality constraint is binding when its equality is satisfied, for these constraints any change in the right hand side of the constraint may change the optimum value of the problem. If a constraint is not binding then it has *slack* and its *shadow price* is 0 as any change in this constraint will not change the optimum until the constraint becomes binding and the slack becomes 0.

The *reduced cost* defines how much an objective function coefficient would have to improve until the corresponding variable would become positive in the optimal solution. This is equivalent to the *shadow price* of the non-negativity constraint of the variable. *Shadow price* and *reduced cost* are the core of performing sensitivity analysis on LPs. Duality is an useful concept for solving LPs as the dual problem can often be easier than the primal.

All LPs can be written in standard form:

$$\begin{aligned} & \text{maximize} && \mathbf{c}^T \mathbf{x}, \\ & \text{s.t.} && \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & && \mathbf{x} \geq 0. \end{aligned} \tag{3}$$

For each row i of the inequality constraint $\sum_j^n a_{ij}x_j \leq b_i$, we can introduce a slack variable $s \geq 0$ to turn it into an equality constraint $\sum_j^n a_{ij}x_j + s = b_i$.

LPs can be solved with either simplex or interior point methods. If the optimum of an objective function is restricted by the constraints, then the optimum value to the problem can be found on the surface of the feasible region or polytope. Simplex explores the extreme points, or corners, of the feasible region. It uses the fact that if the objective value is not optimal at the corner it is currently evaluating then there is an edge from this corner that leads to a corner with a better objective value. Simplex has polynomial running time in practice, but it can be exponential in worst case. Klee–Minty cube is an example of a worst case polytope where simplex ends up in exhaustive search [27].

While simplex explores the surface of the feasible region, interior point methods starts from inside the feasible region and move toward the optimum. Karmarkar [26] designed the first practical interior point algorithm with polynomial time complexity. These interior point methods are particularly well suited for problems where the constraints are no longer linear. This algorithmic power has opened new possibilities in convex optimization.

2.4.2 Convex optimization

Convex optimization extends mathematical optimization from LPs to a much larger domain of objective functions and constraints. In the most general sense convex optimization strives to minimize a convex objective function over a convex set. Figure 1 illustrates the hierarchy of problem formulations within convex optimization. Convex optimization is a useful categorization since while mathematical optimization in general is NP-hard, convex optimization can often be done with polynomial-time algorithms [6].

A convex function f satisfies

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y), \quad (4)$$

for all $x, y \in \mathbb{R}^n$ and $0 \leq \alpha \leq 1$. As a geometric representation in \mathbb{R}^2 , any two points on the graph of a convex function can be connected by a line such that the line stays above the function. A strictly convex function additionally requires strict inequality.

A convex set C satisfies

$$\alpha x + (1 - \alpha)y \in C, \quad (5)$$

for all $x, y \in C$ and $0 \leq \alpha \leq 1$.

Lagrangian duality extends the concept of duality to all convex and nonconvex objectives. We have an optimization problem in standard form:

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{s.t.} && f_i(x) \leq 0, \quad i = 1, \dots, m, \\ &&& h_i(x) = 0, \quad i = 1, \dots, p, \end{aligned} \quad (6)$$

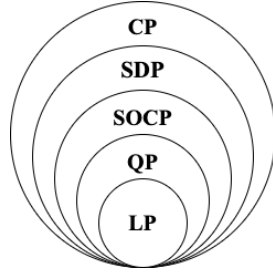


Figure 1: Hierarchy of convex programming problems. Linear program (LP) \subseteq quadratic program (QP) \subseteq second-order cone program (SOCP) \subseteq semi-definite program (SDP) \subseteq cone program (CP).

where $x \in \mathbb{R}^n$. We can then construct the *Lagrangian* L by extending the objective function with a weighted sum of the constraints:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{i=1}^p \nu_i h_i(x). \quad (7)$$

The λ_i and ν_i are called *Lagrange multipliers*. The *Lagrange dual function* is then defined as

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) = \inf_x \left(f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{i=1}^p \nu_i h_i(x) \right). \quad (8)$$

This dual function is a lower bound for the objective in (6). Furthermore for convex optimization strong duality nearly always holds. Finally the *Lagrange dual problem* is

$$\begin{aligned} & \text{maximize} && g(\lambda, \nu) \\ & \text{s.t.} && \lambda \geq 0. \end{aligned} \quad (9)$$

Deriving the analytical expression of the Lagrangian dual function for a LP gives us the dual problem in the same form as in (2) which we got by swapping the constraints and variables [6].

The *Karush-Kuhn-Tucker* (KKT) conditions are necessary and sufficient conditions for a primal-dual convex optimization problem solution (x^*, λ^*, ν^*) to be optimal. They are as follows:

$$\begin{aligned} \nabla L(x^*, \lambda^*, \nu^*) &= 0, \\ f_i &\leq 0, & i = 1, \dots, m, \\ h_i &= 0, & i = 1, \dots, p, \\ \lambda_i &\geq 0, & i = 1, \dots, m, \\ \lambda_i f_i &= 0, & i = 1, \dots, m. \end{aligned} \quad (10)$$

The KKT conditions are utilized when designing algorithms to solve optimization problems, particularly for interior point methods.

One restricted form of convex optimization is called quadratic optimization where the *quadratic program* (QP) is of the form:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^T P x + q^t x + r, \\ \text{s.t.} \quad & Gx \leq h, \\ & Ax = b, \end{aligned} \tag{11}$$

where P is a semi-definite matrix, $G \in \mathbb{R}^{m \times n}$ and $A \in \mathbb{R}^{p \times n}$. Wolfe [52] extended the simplex method to QPs in 1959. When also the constraints are quadratic the problem is called a *quadratically constrained quadratic program* (QCQP).

The advances in interior point methods have made semi-definite programs (SDP) a promising approach to formulate tight relaxed versions of NP-hard combinatorial optimization problems [1, 19, 36, 50]. A SDP can be represented in the following form:

$$\begin{aligned} \text{minimize} \quad & c^T x, \\ \text{s.t.} \quad & \sum_{i=1}^n x_i A_i - B \succcurlyeq 0, \end{aligned} \tag{12}$$

where the constraint is called a *linear matrix inequality*, A_i, \dots, A_n, B are symmetric $m \times m$ matrices and $c, x \in \mathbb{R}^n$. When A_i, \dots, A_n, B are diagonal matrices this becomes a generic LP.

2.4.3 Integer optimization

A useful extension of mathematical optimization is to restrict the optimization variable to be integer as this allows modelling discrete decisions. An optimization with integer constraints is called integer optimization or integer programming (IP). The term mixed integer programming (MIP) is more commonly used when continuous variables are also allowed.

Solving MIPs is hard since the integer constraint makes the feasible set of solutions nonconvex. When $x_i \in \{0, 1\}, i = 1, \dots, n$ the problem is called *pseudo-boolean optimization* [5] or *0-1 integer programming*. For small problems it may be possible to evaluate every possible input but for larger problems this becomes infeasible. The most common approach is to relax the integer constraints and solve the relaxed problem. The relaxed problem is then used to guide a *branch and bound* search. The quality of a LP-relaxation can be measured by the *integrality gap* which is the ratio between the integer solution and the fractional solution. Figure 2 shows an example of an LP-relaxation.

Branch and bound is a general approach to dynamically search the input space. It builds a binary search tree where branching is done at integer variables so that each branch has a different value for this variable. For each node an objective bound can be calculated by solving the relaxed LP problem. The minimum of a relaxed

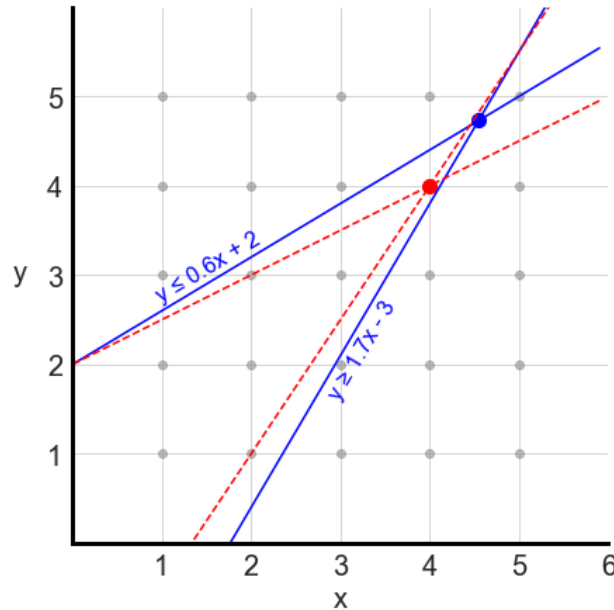


Figure 2: An IP with the objective to maximize y under the constraints $y \leq 0.6x + 2$, $y \geq 1.7x - 3$. The optimum of the LP-relaxation is at the blue dot. The red dashed lines $y \leq \frac{1}{2}x + 5$, $y \geq \frac{3}{2}x - 2$ are additional cuts to the problem which when added to the relaxed problem would yield the IP optimal solution at the red dot.

LP problem is always a lower bound for the discrete problem. The currently best found integer solution is called the *incumbent solution*. If the incumbent solution is better than the relaxed solution at some node then this node can be pruned from the search tree. In this way the algorithm implicitly searches the whole subtree of that node. Branch and bound benefits from a tight integrality gap [39] but suffers when the MIP has symmetries since these symmetric solutions cannot be pruned [37].

Cutting planes are additional constraints on the problem which do not cut out any valid integer solutions but tighten the bounds so that the LP relaxation is stronger. The tightest bound possible, which exactly includes all integer feasible solutions but cuts out all fractional solutions which are better than the integer solution is called the *convex hull* of the integer program. If the convex hull is found then solving the relaxed problem gives the integer solution.

Column generation or variable generation is often used to exploit the problem structure. The problem is reformulated into a *master problem* and subproblems which are called *pricing problems*. *Dantzig–Wolfe decomposition* is one way to do this reformulation. The master problem defines the complicating global constraints between the subproblems while the subproblems can then typically be solved by some dedicated combinatorial algorithm [4, 16]. First only a small subset of variables are considered in the master problem. The pricing problem is solved to find a variable that can be added which improves the objective value. Preferably this is a variable with the most negative reduced cost. The pricing problem is not necessarily solved optimally.

2.4.4 Suboptimal solutions

MIP is a doubled edged sword: it gives great flexibility in modelling a problem but there is no guarantee that the problem stays tractable. The difficulty can vary a lot based on input and in worst case the problem becomes intractable. There is no general way to predict if an input is hard before trying to solve it. Thus we need to implement safeguard heuristics to solve these in practice.

Another practical consideration is that rarely do we need the optimal solution since the modeling of the problem already introduces approximations which make the real optimal solution diffuse. Thus it is often desirable to find a suboptimal solution in less time.

For these suboptimal approaches it is desirable to have guarantees on solution quality or guarantees on runtime or dynamic parameters to adjust a quality versus runtime trade-off.

Approximation algorithms [51] give us solution quality guarantees. Often there may exist algorithms that have better time complexity and give better solutions on average but lack the guarantees for solution quality. These are typically called heuristics. Any algorithm where the time complexity is known to be polynomial gives us more predictable runtimes as the difference between the best and worst case runtimes are less drastic.

To get suboptimal MIP solutions, many of the same methods can be used as for solving MIPs optimally. Branch and bound can be interrupted with a time limit to get the current incumbent solution, column generation can be cut short and the pricing problems can be solved to different degrees of precision. We can also try to utilize the relaxed fractional solutions directly by rounding it to integer at the cost of a reduced objective value, but this is not guaranteed in general and depends on the constraints of the problem. Rounding schemes can be enhanced by formulating stronger relaxed problems with a tighter integrality gap. The relaxed solution may also be used to guide a discrete neighborhood search which only considers a subset of the solution space.

3 The multi-activity assignment problem

In the *multi-activity assignment problem* (MAAP) the objective is to assign activities to predetermined shifts so that the workload demand is met as well as possible. In our version of this problem we want to maximize task rewards and minimize task switch costs. Thus we have a *multi-objective optimization* problem. The task rewards are unique for each worker-task-tick combination which makes it possible to include time, skill or priority-dependent rewards without further modification. We also do not use a minimum assignment length but instead penalize sequence-dependent task switches.

3.1 Modelling the workload

How the workload is defined is one of the key differentiators in employee scheduling problems. In employee scheduling the workload has most commonly been defined without qualifying the type of workload. When the workload type is specified it is apparent that we need to consider the possibility to switch assignment types during a shift. The simplest option is to not allow switches in which case one type of workload is assigned for the whole shift. This has the downside that it may easily cause overassignment if the workload is not sufficient for the whole shift. To allow matching the workload more closely it is beneficial to allow switching assignments during the shift. Allowing this creates a new problem that needs to be considered: how to avoid making too many switches between assignment types.

One common solution to this is to define the workload as indivisible *tasks* that have a fixed start and end time and workload. The modelling assumption is that we know when the task needs to be done, how much time it takes and that it should be done by the same employee. These are quite generous assumptions, particularly for retail where the stochastic human factor is high and tasks that require a dedicated employee are not as common.

Another more flexible approach is to consider the workload as *activities* which are more easily divisible and does not have as strong assumptions about fixed workload as *tasks* have. This suits retail better since the workload is typically inherently divisible consisting for example of multiple separate customer service situations. Consider working at the cash register, here we cannot predict when customers exactly arrive and thus we do not model the individual customer encounters. Characteristic of these type of activities is that if they are left partially undone, the estimated workload not being fully met, the service quality might suffer depending on the realized workload.

One further consideration when modeling workload in retail is that some tasks are more time bound than others. For example opening the store needs to be done at the start of the day while administrative work may be done at any time. Then there are tasks like shelving work where it is preferred to be done as soon as possible after the delivery, but it can be postponed at the cost of service quality.

In our problem the workload will be modelled as time bound, divisible *activities* with a high variance in estimated workload. Thus meeting the workload will not be a hard constraint.

In our problem definition we will call all activities *tasks*.

3.2 Problem definition

We will use the following terminology:

- **Tick:** One day is divided into 96 ticks so that each tick represents 15 minutes.
- **Workload:** Each tick has a workload. The workload specifies for each task how many workers that should be available in that tick.
- **Workload profile:** The workload of the whole day.
- **Employee:** Employees have different skills for different tasks. If their skill for a task is 0 they cannot be assigned to that task.
- **Shifts:** We have one shift per employee. A shift starts at a given tick and has a given length in ticks. The shift can have a break which starts at a given tick and has a given duration.
- **Tasks:** Each input has their own set of tasks. But there are 2 special tasks that are used in all inputs, break and general work. General work is penalized but can be assigned for any tick. Since general work can always be assigned this means that there is always a feasible solution to the problem. Break tasks are determined in the input and their placement is fixed. Tasks have a task switch cost.
- **Task rewards:** For each employee-task-tick combination there is a reward which is given as input. This reward considers employee skill and task priority among other rewards/penalties. This also includes a penalty for shifts starting with general work.
- **Task switch costs:** A task switch happens if an employee does a different task at two consecutive ticks.
- **Maximum consecutive assignment:** Some tasks may have a restriction on how many consecutive ticks an employee can work on them.
- **General work:** General work can be done at any tick but it is penalized, thus the problem always has a feasible solution.
- **Break:** Break is considered a task that has fixed placement. Not all shifts have breaks.
- **Shifts starting with general penalty:** If a shift is assigned general work for the first tick, this accrues a penalty which can be included in the task reward for that tick. This “shift starting with general” penalty is special in the sense that if we formulate a model where shift starts are not fixed then this penalty is no longer tied to a specific tick and requires special considerations to be modelled.

Next we will formulate the problem formally. We define the following sets:

$$\begin{aligned} i \in \mathbf{E}, |\mathbf{E}| = N, & \text{ all employees in the instance.}, \\ j \in \mathbf{T}, |\mathbf{T}| = M, & \text{ all tasks in the instance.}, \\ k \in \mathbf{Q}, |\mathbf{Q}| = K, & \text{ all ticks in the instance.} \end{aligned}$$

The problem input $\forall i \in \mathbf{E}, \forall j \in \mathbf{T}, \forall k \in \mathbf{Q}$:

$$\begin{aligned} q_{ijk}, & \text{ reward for employee } i \text{ doing task } j \text{ at tick } k, \\ w_{jk}, & \text{ workload for task } j \text{ at tick } k, \\ p_j, & \text{ task switch penalty for task } j, \\ d_j, & \text{ maximum assignment duration for task } j, \\ l_i, & \text{ shift length of employee } i, \\ s_i, & \text{ shift start tick for employee } i, \\ bl_i, & \text{ break length for employee } i, \\ b_i, & \text{ break start tick for employee } i, \\ r, & \text{ reward for staying in same task.} \end{aligned}$$

We want to maximize the utility function

$$U = \sum_{i \in \mathbf{E}} \sum_{j \in \mathbf{T}} \sum_{k \in \mathbf{Q}} \left(q_{ijk} x_{ijk} + x_{ijk} \sum_{p \in \mathbf{T}} a_{jp} x_{ip(k+1)} \right), \quad (13)$$

where $x_{ijk} = 1$ when employee i is assigned to task j at tick k , otherwise $x_{ijk} = 0$ and a_{jp} is the task switch cost from task j to task p so that

$$a_{jp} = \begin{cases} p_j, & \text{if } j \text{ is general work and } p \text{ is general work.} \\ 0, & \text{if } j \text{ is break or } p \text{ is break.} \\ r, & \text{if } j = p. \\ \max(p_j, p_p), & \text{otherwise.} \end{cases} \quad (14)$$

This can be written in matrix form as

$$U = \mathbf{q}\mathbf{x} + \mathbf{x}^T \mathbf{A}\mathbf{x}. \quad (15)$$

The optimization output is

$$\mathbf{x}^* = \arg \max_{\mathbf{x}} U(\mathbf{x}), \quad (16)$$

which determines the activities for all employee shifts and $U(\mathbf{x}^*)$ which gives a score for the solution. We also need to satisfy the following constraints:

- An employee can only do one task per tick.
- An employee cannot be assigned to a task that is outside their shift.

- The number of employees assigned to a task in a tick cannot be more than the workload of that task in that tick, no overstaffing is allowed.
- A task i can only be worked on for d_i consecutive ticks.

An example instance of this problem can be seen in Figure 3. Two variations of this instance are listed in the appendix, one with 3 employees available in Figure A1 and another with higher priority for bakery in Figure A2.

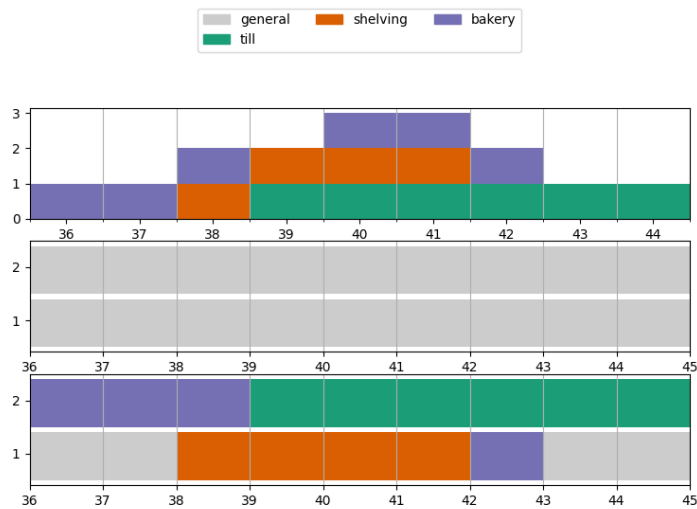


Figure 3: A problem instance with 2 employees and workload at ticks $[36,44]$. The first graph shows the stacked workload for each tick. The second graph shows the shifts of the 2 employees. The last graph shows one possible assignment of tasks to shifts. Employee skill and task switch costs are not shown here, but in this example all employees have same skill for all tasks and all task switches have the same cost.

3.3 Baseline solution

The baseline solution selects shift placements with simulated annealing, which is a probabilistic search method that looks for local improvements to a solution while also moving out of local optima with a certain probability. To determine the quality of shift placements it needs to solve the MAAP subproblem. It does this by solving an assignment problem at each tick. Between each tick the costs are updated to include task switch costs. This is a greedy approach which does not consider future workload when assigning tasks for a tick. Thus it may miss future task switches or commit to a task that gives less total utility when considering the whole shift. This same heuristic is also used to determine the final task assignments for a shift.

3.4 Shift placements

For the *shift placement problem* the input shift and break start times, s_i and b_i , need to be additionally decided for each employee. The shift for each employee has to start within an interval defined by the input. Similarly breaks are restricted to start within an interval which is offset from the shift start time.

3.5 Measuring solution quality

It is always possible to visually inspect the solution but this is subjective, laborious and non-quantifiable. So there needs to be a metric that can be used to measure the quality of a solution. The existing utility function (13) is the best we have for measuring overall solution quality. We can try to build a better or different utility function but without any way to validate it with domain expertise or feedback from the end users this becomes a shot in the dark.

However, we know that an important quality of the utility function is to value consecutive assignments of tasks. This gives us room to consider some variations of the utility as long as it values assigning tasks into continuous blocks. This is advantageous since a different formulation of the task switch cost may be easier to optimize, while still managing to keep tasks in continuous segments.

If we can reason that some utility function we come up with is better, then it is reasonable to use that to evaluate solution quality and also use it to compare to the existing solutions. However, as task rewards are given in the input we have to assume these represent the real utility of doing those tasks. The interpretation of task switch costs assigned to each task in the input are more flexible but they roughly represent the loss of effective working time switching to and from that task. The relative magnitude of the task rewards versus task switch costs are also important to keep in mind. Adjusting this balance changes the level of incentive to avoid task switches.

We can calculate some interesting metrics for the solution like number of task switches or amount of general work but these should not be used to measure the quality of solution. In mathematical optimization the utility function is the quality and if we want to measure the quality differently then we use that as a utility function. However, sometimes it may be that the quality we want to measure is not a suitable utility function. Then we may use a *proxy utility* function that we hope would capture the same effects as the real utility. We hope because in general it is very difficult to quantify the similarity of two high-dimensional functions. So when possible we should use the same measure for utility and quality of solution. Otherwise the solution quality, achieved by the optimization, is dependent on input and we can construct inputs that create any extremes of this quality metric if we abuse a component of the utility function that is not measured by the quality metric.

3.5.1 Comparing solution quality

We also need a method to compare two utility scores U and U_{ref} to each other. We can start by taking the difference between our actual utility U and the baseline

reference utility U_{ref} . This difference $\Delta = U - U_{\text{ref}}$ preserves the sign which is positive when our utility is better. The drawback with measuring the difference is that this quantity scales proportionally with the size of the problem instance as the achievable utility score increases. We would want negate this effect by dividing the difference with a suitable quantity. We divide the difference by U_{ref} and get the formula

$$\delta = \frac{U - U_{\text{ref}}}{U_{\text{ref}}} \cdot 100\%. \quad (17)$$

However we cannot use this directly as our utility can get both negative and positive values. A percentage difference is not defined between values of different signs. Nevertheless these situations where the compared utilities are of different sign are quite rare since most instances achieve positive utility as long as there is not too much general work assigned. We choose to keep the sign of difference by taking the absolute value in the divisor. Another consideration is that as the reference value gets close to 0 this metric approaches infinity, but as all our scores are integer this is not an issue. Finally this metric is not defined if reference score is 0, this needs to be handled separately in our case by setting the divisor to 1 in these cases. Our final modified percent difference formula then becomes

$$\delta_* = \frac{U - U_{\text{ref}}}{\max(1, |U_{\text{ref}}|)} \cdot 100\%. \quad (18)$$

In all our results the reference value will be from the baseline solution with a positive value being better.

An alternative approach to dealing with the negative values would be to remove them by moving the zero point of our utility so that all scores remain positive. This type of instance specific scaling could be done by setting the zero point to be the value where all shifts are assigned with general work. After this we could use these translated scores to measure the change using the relative difference formula (17).

3.6 Problem analysis and special cases

Our problem can be reduced to the interval scheduling problem (ISP) when the input workload can unambiguously be mapped to intervals. This requires that the workload for any task does not exceed 1 at any point and that task switches are penalized sufficiently that they are only done when it would not be possible to continue with the task otherwise. The instance in Figure 3 is an example of a problem where there is such a mapping for the workload. For the ISP solution we would exclude any task assignments where the whole interval has not been assigned. Furthermore all employees are identical and able to do all tasks. For the standard ISP we have one employee with a shift that covers the whole day. As we saw in Section 2.3 many extensions of the ISP exist which makes it approximate our problem better but the relevant extensions of having employees available only for a certain time of day or restricting employees to certain tasks both make the problem NP-hard. Thus we can conclude that since these special cases are NP-hard also our task assignment problem is NP-hard.

Arkin and Silverberg [2] showed that 3-SAT can be reduced to the interval scheduling problem with multiple non-identical machines. Figure A3 in the appendix shows one example 3-SAT instance solved using this reduction.

If we drop the task switch costs the problem can be solved optimally by solving the assignment problem at each tick. The assignment problem can be solved in polynomial time by the Hungarian algorithm [31] and derivations of it [23]. The baseline solution builds a heuristic out of this by updating the assignment rewards with the task switch costs at each tick.

We can represent our problem as a longest path search in a DAG that has a node for each combination of task assignments at each tick. With N employees and K tasks, with unlimited workload, this gives us $\mathcal{O}(K^N)$ nodes at each tick and $\mathcal{O}(K^{2N})$ arcs between the nodes at two ticks giving us a $\mathcal{O}(K^N + K^{2N})$ -time algorithm to find the longest path. This is not a practical formulation but illustrates the decision space we are dealing with.

3.7 Decomposition

A common way to approach hard problems is to divide them into smaller more manageable problems and then combining the solutions to these smaller problems. If this division can be done until we are only left with many small trivial problems which do not depend on each other it is said that the problem has optimal substructure. This approach is called divide-and-conquer. When subproblems overlap, dynamic programming is used to avoid repeated solves for the overlapping segments.

We can decompose the problem by employee-task when there are sets of employees that are disjoint in the tasks they are able to do. These sets of employees and their corresponding tasks can be solved individually. Similarly, we can decompose by employee-tick when there are sets of employees with shifts that are disjoint with the rest of the employees. For this decomposition we can consider shifts with breaks as two separate shifts or employees. However, both of these decompositions only help in selected cases.

As subsequent ticks effect each other whenever they are part of the same shift, due to task switch costs and maximum consecutive duration constraints, we can not do any subdivisions where shifts are split into separate problems. The only exception to this is if there is a tick with no workload as this forces a general work assignment on this tick for all shifts. If such a tick exists, the whole day can be split into separate problems at this point.

However in general and when decomposing the problem further we end up with subproblems that are dependent on each other. Each tick subproblem is dependent on the nearby ticks and through the resulting dependency chain potentially on all ticks in a shift. Each task subproblem is dependent on how other task have been assigned since those effect which employees are available for a task assignment. Each employee subproblem is also dependent on what tasks have been assigned to previous employees. A greedy algorithm can be built based on any of these three subdivisions.

As the baseline solution solves the problem by solving one tick at a time, it could be interesting to try solve it one task at a time starting with the most valuable task

or one employee at a time sorting the employees first by number of skills and then by what is the maximum assignment value for the employee with remaining tasks.

We could also try to divide the problem into dependent subproblems and then forcibly combine these by resolving any dependency conflicts as another optimization. As the problem does not have optimal substructure we have to avoid doing too many of these divisions since each dependency conflict will make the solution worse. We can try to find such subproblems where the dependency is minimized by some approximation of sparsest cut, for example the shifts could be decomposed by finding two sets of shifts with minimal overlap.

4 Implementation

As our problem, is NP-hard we know that solving it with an integer program is at least reasonable compared to a more specialized algorithm. To test solver capabilities more widely we will formulate both linear and quadratic objectives. We also want to implement a rounding scheme based on this model. Thus we also need to consider how well the fractional solutions approximate the integer objective.

4.1 Integer program formulation

We will be optimizing the decision variable $\mathbf{x} = \{x_{ijk} \mid \text{employee } i \in \mathbf{E} \text{ is able to do task } j \in \mathbf{T} \text{ at tick } k \in \mathbf{Q}\}$. Employee i cannot do task j at tick k if tick k is outside the assigned shift, there is no workload for task j at tick k , the employee has a break at tick k or the employee has skill 0 in task j .

Our objective is then to

$$\begin{aligned} \text{maximize } & U = \mathbf{q}\mathbf{x} + \mathbf{x}^T \mathbf{P}\mathbf{x}, \\ \text{where } & x_{ijk} = 1 \text{ if employee } i \text{ is doing task } j \text{ at tick } k, \\ & q_{ijk} \text{ is the reward for employee } i \text{ doing task } j \text{ at tick } k, \\ & \mathbf{P} \text{ is a matrix with task switch penalties which may be } \mathbf{0}. \end{aligned}$$

The utility U consists of a linear part $\mathbf{q}\mathbf{x}$ which models the task rewards and a quadratic part $\mathbf{x}^T \mathbf{P}\mathbf{x}$ which models the task switch penalties. See Section 4.2 for different variations of the task switch cost objective.

The core constraints of our problem are

$$x_{ijk} \in \{0, 1\} \quad \forall i \in \mathbf{E}, \forall j \in \mathbf{T}, \forall k \in \mathbf{Q}, \quad (19)$$

$$\sum_{j \in \mathbf{T}} x_{ijk} \leq 1 \quad \forall i \in \mathbf{E}, \forall k \in \mathbf{Q}, \quad (20)$$

$$\sum_{i \in \mathbf{E}} x_{ijk} \leq w_{jk} \quad \forall j \in \mathbf{T}, \forall k \in \mathbf{Q}. \quad (21)$$

Here (19) restricts the problem to be binary, (20) restricts that for each employee i and tick k at most one task can be selected and (21) restricts that for each task j and tick k the number of assigned workers cannot exceed the available workload. Strictly speaking (20) could be an equality constraint but this formulation allows empty assignments when shift positions are flexible.

In addition to these core constraints we will also be limiting the maximum consecutive assignments for those tasks j that have such a limit d_j . We will set the constraint

$$\sum_k^{k+d_j+1} x_{ijk} \leq d_j, \quad (22)$$

for each $i \in \mathbf{E}$ and for each j, k such that task j starting at k can be assigned more than d_j consecutive ticks.

4.2 Task switch costs

As we want a model that is flexible enough to allow task switches at any tick we need to penalize or prevent excessive switches so that tasks would be assigned in continuous blocks.

The input determines for each task j a task switch cost penalty p_j , which we interpret as the magnitude of loss in effective work when switching to or from this task. Noteworthy is that we do not model task switches as actually consuming time in the schedule, which would be the realistic way to model task switches. Nevertheless this interpretation is compatible with how we consider tasks to have flexible workload as is described in Section 3.1.

To allow comparisons with the existing solution we are particularly interested in modelling the existing utility function so that scores are comparable. Measuring the solution quality in general will be considered in more detail in Section 3.5.

Since the model allows switching tasks between every tick, which is undesirable for a proper schedule, we want the objective function to value assigning continuous blocks of each task.

For the task switch cost we will present a **linear objective** in Section 4.2.1 and a **nonconvex objective** and **convex objective** in Section 4.2.2.

4.2.1 Linear task switch costs

We can penalize task switches with a linear objective function if we add some auxiliary variables \mathbf{z} which are true when a task switch happens and can then be associated with a linear task switch cost in the objective function. Then $\mathbf{P} = 0$ in our objective and we have a linear program. Our objective then becomes

$$U = \mathbf{q}^T \mathbf{x} + \mathbf{a}^T \mathbf{z}, \quad (23)$$

where \mathbf{a} contains all the task switch costs between tasks and \mathbf{z} is defined by the constraints

$$0 \leq x_{ijk} + x_{ip(k+1)} - 2z_{ijpk} \leq 1, \quad (24)$$

for each $i \in \mathbf{E}, j, p \in \mathbf{T}, k \in \mathbf{Q}$. As there is a reward when staying in the same task we cannot rely on \mathbf{z} always having penalty coefficients and thus z_{ijpk} may only be true iff x_{ijk} and $x_{ip(k+1)}$ are both true, otherwise z_{ijpk} may give superfluous rewards. When \mathbf{x}, \mathbf{z} are both binary vectors (24) is a sufficient constraint to ensure this.

As we want to implement a rounding scheme for the relaxed version of this problem we need to pay special attention to these auxiliary variables. With the constraint in (24), z_{ijpk} would be optimized to 0.5 when a task switch happens. Thus we need to change the constraint coefficient on z_{ijpk} and ensure that we do not get superfluous rewards from z_{ijpk} with two additional constraints

$$0 \leq x_{ijk} + x_{ip(k+1)} - z_{ijpk} \leq 1, \quad (25)$$

$$0 \leq x_{ijk} - z_{ijpk}, \quad (26)$$

$$0 \leq x_{ip(k+1)} - z_{ijpk}, \quad (27)$$

for each $i \in \mathbf{E}, j, p \in \mathbf{T}, k \in \mathbf{Q}$. However, it is noteworthy that in a fractional solution (25) will not start increasing z_{ijpk} until $x_{ijk} + x_{ip(k+1)} = 1$. Thus the fractional solution would strive to assign half tasks to avoid the task switch penalty. So while the integral solutions values are the same as for the utility (13) and the **nonconvex objective** which will be used to match the utility, the fractional solution values will be different for the same fractional assignments. Thus we will call this the **linear objective** when emphasizing what objective has been used to calculate the value of a fractional solution.

To finish this section we will propose two additional ways to define the linear task switch penalties which we have not pursued in this implementation. For a fractional solution constraints $x_{ijk} + x_{ip(k+1)} = 2z_{ijpk}$ could be used. This makes the task switch penalty proportional to the amount of assignment done, but it also gives a half task switch penalty when a task switch is not happening. The second alternative would be to add constraints $x_{ij(k+1)} + z_{ijk} \geq x_{ijk}$ which works similar as the convex qp by striving to keep the same task in next tick or otherwise penalize a switch with z_{ijk} .

4.2.2 Quadratic task switch costs

Nonconvex. We can use the same task switch cost definition as was used in the utility (13), by assigning the task switch costs to products of two binary assignment variables. However these bilinear terms makes this objective nonconvex and the corresponding matrix P that determines this quadratic form is indefinite. This limits the tools available to optimize this type of objective.

Convex. The utility function in our problem is nonconvex but to be able to solve the problem more efficiently we want to formulate a convex objective which might approximate the utility reasonably well. As we mused in section 3.5 the reason for the task switch penalties is to keep tasks assigned into continuous segments. Here we will formulate a convex objective which achieves this.

We can construct a quadratic form

$$\sum_{i \in \mathbf{E}} \sum_{j \in \mathbf{T}} \sum_{k \in \mathbf{Q}} c_j (x_{ijk} - x_{ij(k+1)})^2, \quad (28)$$

which rewards assigning the same task in consecutive ticks. The sum of convex functions is convex making this quadratic form convex as well. It adds the cost $c_j + c_p$ to the objective when a task switch from j to p happens. This formulation has the beneficial property that it will also work in a fractional solution by encouraging two consecutive assignment to have the same fractional value weighted by the task switch cost. If we then force an assignment to 1 by rounding then this assignment cascades to nearby ticks by this objective.

The quadratic form can be encoded into a matrix P with $\mathbf{x}^T P \mathbf{x}$. This matrix can be made symmetric $\mathbf{P}' = \frac{1}{2}(\mathbf{P}^T + \mathbf{P})$ without changing the value of the quadratic form for a particular \mathbf{x} . The resulting \mathbf{P}' is positive semi-definite and the quadratic form is convex. This convex objective allows numeric optimization using first and second order methods. The drawback here is that we cannot express the baseline

utility function with this formulation exactly. The end result in utility of switching from task a to task b is then $p_a + p_b$. General to general is not assigned a task switch cost and staying in same task does not get an additional reward. We will call this the *proxy utility*, when it is used to optimize the objective the final solution will still be evaluated with the utility in equation (13).

We can also turn the nonconvex task switch cost matrix from the utility (13) into a convex one with a sufficiently high constant C as follows:

$$\mathbf{P}' = \mathbf{x}^T(\mathbf{A} + C\mathbf{I})\mathbf{x} - C\mathbf{x}.$$

This will retain the same optimum value as \mathbf{P} when \mathbf{x} are binary, however it drives \mathbf{x} towards 0.5 in the fractional solution and widens the integrality gap, thus we will not be using this in our implementation.

Convex with adjustments The task switch penalties and rewards in the utility (14) are more extensive than what the previous convex objective can capture. It does not penalize general work as much as the utility due to lack of general to general task switch costs, the task switch costs are higher due to taking the sum instead of max, however the lack of same task switch rewards does offset this cost in the other direction. We create another convex objective, which we call *convex2* or the adjusted convex objective. With this objective we try to balance and adjust these differences. This is done by moving these special task switch rewards partly to the task rewards.

- The general to general task switch cost is added to each general task assignment and general task cost p_j is set to 0.
- The same task reward is added to each task assignment and subtracted from each task switch cost.
- The rewards of a shifts first and last tick are not adjusted
- The task switch penalties are averages $\frac{1}{2}(p_a + p_b)$ instead of the sum $p_a + p_b$

4.3 Rounding schemes

Thus far this model can provide us with optimal results when solved with a commercial MIP solver. However we are also interested in suboptimal solutions which can be reached in a reasonable time. We will do this by relaxing the integer constraint and then implement a iterative rounding scheme for the fractional solutions that uses an open source LP or QP solver as a subroutine.

The core of a rounding scheme is high quality fractional solutions which are able to guide the rounding in the right direction. For our goal of assigning consecutive tasks a good property is an objective which drives consecutive assignments to be the same. Another favorable property is if the fractional solution can be driven towards integer values.

The basic building block of these rounding schemes is procedure that selects a task for a worker at a specific tick. It rounds the decision variable for the selected

task to 1 and all other task variables to 0. Two different schemes were implemented which we call tick and n -best in short. Both the convex and convex2 objectives were used for these schemes giving us a final four procedures tick, n -best, tick2 and n -best2.

4.3.1 Tick based rounding

In tick based rounding we will round all fractional assignments within one tick before moving to the next one. We choose to iterate ticks in order starting from the start of the day. We can expect that this method can work reasonably well as the input shifts have been selected with a greedy algorithm which iterates the ticks in this order as was described in Section 3.3. We could also choose some other order of ticks for example by first solving the tick with the tightest match between workload to available skilled workers.

The rounding proceeds by selecting the most confident task for a worker. After rounding the problem is solved again. This repeats until all assignments are integer. The amount of solves can be reduced with a fast-lane rounding where all values that are close enough to 1 will also be rounded so that we select multiple tasks between solves.

4.3.2 N-best rounding

As an alternative to the previous more heuristic guided approach, which can be wrong when the assumptions the heuristic is based on does not exist in the input, we would also like to try a more principled approach. This more general approach finds the most decisive variable globally and performs task selection based on it. Then the updated problem is solved again and this is repeated until all variables are integer. We can parameterize the number of variables n being rounded between each solve and we get what we call *n -best rounding*. Too aggressive rounding may lead to infeasible solutions. We combat this by limiting for each round that a *worker-tick*, *task-tick* and *task-tick* combination cannot be repeated in the same round. We can also limit the absolute amount of rounding done to keep it more sane. As a last safeguard, if we reach an infeasible rounding we backtrack and do rounding less aggressively. In our implementation we will set the starting value n to half of the number of employees N in the instance. We will also limit the absolute amount of rounding to be $0.4n$ per iteration. Once the best candidate variables are below 0.1 they are rounded to general work which will never turn the solution infeasible. We will also use a special case of this scheme with $n = 1$ called *1-best rounding* which is optimized for solution quality instead of runtime.

4.4 Extensions of the model

4.4.1 Modelling different types of tasks

As was discussed in Section 3.1 there are a few alternative ways that the workload could be defined which might be more suitable for certain type of tasks.

It is natural to have some tasks that do not need to be done at a fixed time, it could improve the resulting assignment significantly if we allow this instead of arbitrarily fixing them to some specific time span. For those tasks we can replace the constraint (21) with

$$\sum_{i \in \mathbf{E}} \sum_{k \in \mathbf{Q}} x_{ijk} \leq w_k \quad \forall j \in \mathbf{T}. \quad (29)$$

This could fill problematic gaps in the schedule but it also makes the problem more globally connected as these tasks then become a shared resource in every assignment decision.

When tasks have a strict workload we can replace (21) with equality constraints for those tasks that need to be fully completed.

4.4.2 Optimizing shift and break placement

We can try to take full advantage of the modelling capabilities of mathematical optimization by modelling the shift and break placements as well. As this drastically enlarges the search space and thus the worst case runtime, we can approach this more carefully by still leaving in some restriction on how far from the initial shift placement the shifts can be moved.

When shift and break positions are not fixed we need some additional constraints. Shift length and break length of employee i needs to be limited to length l_i and d_i respectively. We can do this with the constraints

$$\sum_{k \in \mathbf{Q}} \sum_{j \in \mathbf{T}} x_{ijk} \leq l_i \quad \forall i \in \mathbf{E}, \quad (30)$$

$$\sum_{k \in \mathbf{Q}} x_{i(\text{break})k} \leq d_i \quad \forall i \in \mathbf{E}. \quad (31)$$

When we allow flexible start times for shifts we model this by having a baseline start time and a certain amount of slack s which defines how many ticks backwards or forwards the shift start can be moved. We create the following constraints

$$0 \leq \sum_{k=t}^{t+l_i} \sum_{j \in \mathbf{T}} (x_{ijk}) - l_i s_{is} \leq l_i - 1, \quad (32)$$

$$\sum_{s \in \mathbf{S}_i} s_{is} = 1, \quad (33)$$

for each $s_{is} \in \mathbf{S}_i$, for each $i \in \mathbf{E}$, where \mathbf{S}_i are new variables that encode every possible shift start we want to allow for employee i . (32) sets s_{is} to true when l_i ticks that belong to the shift s_{is} have been assigned. (33) constrains that only one of the possible shifts can be selected per employee.

We also need to encode the break starts in a similar way

$$0 \leq \sum_{k=t}^{t+d_i} (x_{ijk}) - d_i b_{ib} \leq d_i - 1. \quad (34)$$

$$\sum_{\forall s} b_{ib} = 1, \quad (35)$$

for each $b_{ib} \in \mathbf{B}_i$, for each $i \in \mathbf{E}$, where \mathbf{B}_i is the set of all possible break starts for an employee. (34) sets the b_{ib} variable to true when a break is selected. (35) limits that only one of the breaks can be selected.

Break timings are constrained to an offset from shift start and thus we also need to specify which breaks are allowed for a certain shift start. We create the constraint

$$-1 \leq s_{is} - \sum_{b \in B_{s_{is}}} b \leq 0, \quad (36)$$

for each $i \in \mathbf{E}$, for each $s_{is} \in \mathbf{S}_i$, where $\mathbf{B}_{s_{is}}$ are the breaks that are compatible with shift s_{is} . This constraint sets at least one of the compatible breaks to be true when s_{is} is true. These break starts can be allowed in multiple different shifts and thus we have the lower bound -1 .

Finally the “shift starting with general” penalty now requires special consideration. When shift starts are fixed a static penalty can be added to all the shift start ticks that assign general work. With flexible shifts we do not know when the shift will start and thus we have to create a new decision variable p_{is} which is true when a selected shift is starting with general task. This variable can then be penalized in the cost function. We encode p_{is} with the constraint

$$0 \leq x_{ijk} + s_{is} - 2p_{is} \leq 1, \quad (37)$$

for each $i \in \mathbf{E}$, for each $s_{is} \in \mathbf{S}_i$, where j is general work and shift s_{is} starts at tick k .

Constraints (32) and (34) rely heavily on s_{is} and b_{ib} being integer. Thus any rounding schemes for flexible shifts need to be designed around these decisions. We will leave this scenario out of scope for our rounding scheme.

5 Results

5.1 Data

5.1.1 Real instances

A set of 44 instances representing real-world optimization cases was provided for this thesis. For each input instance we have generated a shift placement with the baseline solution. At this point 3 instances were excluded in which some shifts were breaking the shift placement constraint for an employee. Then a further 8 inputs were excluded from the results as they represented trivial cases with for example no compatible workload for employees and thus the optimal solution only assigned general work to all shifts. This left us with 33 real instances. A summary of the instances can be seen in Table 1. One of the real instances, AD, was an order of magnitude slower to optimize than the others. See Figure A7 in the appendix for an example of a real instance workload distribution and the optimal solution to it.

5.1.2 Artificial instances

Some artificial instances were generated with the following fixed parameters: 10 task types, 96 ticks of employee availability and workload and an average of 4 skills per employee. The employee count N was varied. We generated 10 different instances for each multiple of 10 employees up to 100 employees. Thus we have a total of 100 artificial instances for 10 different values of N . Unlike in the real instances, the total workload per tick in artificial instances is uniformly distributed throughout the day and individual task workload varies more randomly so longer continuous task assignments are less likely to exist. See Figure A8 in the appendix for an example of workload distribution and optimal solution to one artificial instance.

5.2 Benchmark setup

All benchmarks were run with a 2,6 GHz 6-Core Intel Core i7 32GB machine. The integer programs were solved with GUROBI 9.5.1 with a 30000 second time limit which was hit in some exceptional cases. Fractional convex QP and LP programs were solved with the open source OSQP 0.6.2 solver.

5.3 Validation and optimal solutions

First we validated that the baseline solutions matched the score our model gave with the same assignments. This uncovered some discrepancies between the initially assumed utility and the actual utility function which we presented in (13). After fixing these discrepancies we solved the instances optimally to reveal how much room there is to improve the solution.

The optimal solutions from our integer program was compared to the baseline solution on real instances. First we ran the benchmark with both the LP and nonconvex QP objectives which both should get the same solution. This was confirmed

Instance	N	Tasks	Ticks	Total wl	Average wl	Average skill
A	29	16	65	716	11	3
B	35	14	65	1010	16	4
C	29	16	65	715	11	3
D	29	16	65	717	11	3
E	35	14	65	1010	16	4
F	28	16	65	725	11	3
G	29	16	65	707	11	3
H	41	13	67	929	14	1
I	18	13	54	493	9	2
J	36	11	67	611	9	1
K	20	10	96	487	5	6
L	19	10	96	495	5	6
M	19	9	96	498	5	6
N	18	9	96	482	5	6
O	17	10	96	520	5	6
P	17	9	96	482	5	6
Q	12	6	76	267	4	3
R	20	10	96	520	5	6
S	40	13	62	1169	19	2
T	39	13	62	1179	19	2
U	22	11	53	776	15	2
V	40	13	62	1179	19	2
W	21	12	52	712	14	3
X	21	11	53	796	15	3
Y	22	12	53	699	13	2
Z	41	13	62	1158	19	2
AA	27	11	49	865	18	3
AB	8	2	96	284	3	2
AC	39	10	59	1202	20	5
AD	14	7	51	814	16	7
AE	35	10	59	1202	20	5
AF	40	10	59	1220	21	5
AG	34	10	58	856	15	1

Table 1: Real instances summarized. Showing N employees, number of unique tasks in workload, number of ticks with workload, average workload per tick and average number of skills per employee.

in the benchmark results. However the LP objective performed worse in terms of runtime. Results for the slowest LP runtimes can be seen in Table 2 and for QP respectively in Table 3. The mean runtimes were 15.5s vs. 5.7s when the slowest instance AD was excluded. Thus the slower LP objective will not be used for benchmarking the optimal artificial instances. We also benchmarked the integer solutions for the convex QP objective. It only approximates the reference utility but

it performed exceptionally well compared to the LP and nonconvex QP objectives as can be seen from Table 4.

	runtime(s)	score	baseline score	relative difference	optimal
AD	30004.2	11209	10975	2.13	False
B	382.33	21767	21648	0.55	True
E	50.74	21825	21716	0.5	True
AC	7.16	7818	7767	0.66	True
F	7.07	8350	8300	0.6	True

Table 2: 5 slowest runtimes for LP objective.

	runtime(s)	score	baseline_score	relative difference	optimal
AD	14705	11229	10975	2.31	True
B	91.39	21767	21648	0.55	True
E	32.53	21825	21716	0.5	True
AC	6.07	7818	7767	0.66	True
AF	5.51	6745	6587	2.4	True

Table 3: 5 slowest runtimes for nonconvex QP objective.

	runtime(s)	score	baseline score	relative difference	optimal
AD	52.75	11089	10975	1.04	True
B	12.46	21732	21648	0.39	True
E	6.96	21812	21716	0.44	True
T	3.33	27326	27300	0.1	True
V	3.3	29649	29617	0.11	True

Table 4: 5 slowest runtimes for convex QP objective, the results are optimal in respect to this objective. The convex2 objective had similar runtimes.

All the optimal results are summarized in Table 5 for real instances and in Table 6 for artificial instances, these also include the relaxed convex and convex2 objectives. As the artificial instances have a wide range of instance sizes the overall runtime distribution is not that interesting as the smallest instances will have short runtimes and largest will have the longest runtimes.

The optimal results on individual real instances can be seen in Table A1. Figures A4, A5 and A6 graphs the the relative improvement and runtimes on individual artificial instances grouped together by N. This figure also shows the instances that did not reach the optimal solution due to the time limit.

The convex objective is faster to optimize so it was solved for all 100 artificial instances. We observe that the optimal solution for the convex objective is often worse than the baseline.

We also did a comparison where we evaluated the baseline solutions with the convex objective and then compared this to our optimized convex objective. These results are in Table 7. Here the optimum result is on average 24.7% better, it is clear

that the convex objective is not able to capture all the same features as the target utility and this difference is emphasized in the artificial instances.

	LP	QP	QP-convex	QP-convex2
N	33	33	33	33
mean (%)	0.7%	0.7%	0.4%	0.6%
median (%)	0.5%	0.5%	0.2%	0.3%
range (%)	[0.0%, 4.2%]	[0.0%, 4.2%]	[-0.1%, 3.1%]	[-0.4%, 4.2%]
mean (s)	924.3 s	451.2 s	3.4 s	9.3 s
median (s)	1.9 s	2.0 s	1.0 s	1.2 s
range (s)	[0.1 s, 30004.1 s]	[0.1 s, 14705.0 s]	[0.1 s, 52.8 s]	[0.1 s, 246.3 s]

Table 5: Real instances optimal summary

	LP	QP	QP-convex	QP-convex2
N	-	82	100	100
mean (%)	-	19.7%	-1.6%	7.1%
median (%)	-	6.7%	-0.2%	2.2%
range (%)	-	[1.5%, 785.5%]	[-130.8%, 83.1%]	[-5.2%, 401.2%]
mean (s)	-	5493.1 s	118.1 s	384.7 s
median (s)	-	289.5 s	4.4 s	5.9 s
range (s)	-	[0.4 s, 30005.0 s]	[0.3 s, 5067.8 s]	[0.3 s, 7903.9 s]

Table 6: Artificial optimal summary

	Real instances	Artificial instances
N	33	100
mean (%)	1.0%	24.7%
median (%)	0.5%	14.3%
range (%)	[0.0%, 5.3%]	[2.1%, 257.4%]
mean (s)	3.4 s	134.9 s
median (s)	1.0 s	12.1 s
range (s)	[0.1 s, 52.8 s]	[0.4 s, 5351.6 s]

Table 7: Optimal convex objective solution vs. baseline evaluated with the convex objective.

5.4 Fractional solutions and integrality gaps

We can optimize the models without the integrality constraints to obtain fractional solutions. While this will not give task assignment for shifts, the score can be used to approximate the value of a shift assignment. As the utility is nonconvex, we can not solve it efficiently even without the integer constraint. In fact the fractional solution is integer due to the boolean bilinear terms. Instead we evaluate the LP

objective (23), the convex QP objective (28) and the adjusted convex2 QP objective. We measure the difference between the integer and fractional solutions as

$$\text{IG} = \frac{S_{\text{frac}} - S_{\text{int}}}{\max(1, |S_{\text{int}}|)}, \quad (38)$$

where S_{frac} is the optimal fractional solution using either the LP, convex or convex2 objective and S_{int} is the optimal integer solution using the utility, convex or convex2 objective. We are mainly interested in comparing the fractional solutions to the optimal utility solution but we will also test how close the convex and convex2 fractional solutions are to their integer solutions. These results can be seen in Figure 4 and 5.

For estimating utility the LP fractional solution performs best by overestimating the integer solution by 1% for real instances and 65% for artificial instances. If the convex objective was used as utility then the corresponding fractional solutions would overestimate the integer solution by 1% and 17% respectively.

5.5 Rounding results

As the adjusted convex2 objective was clearly better and roughly as fast as the non-adjusted convex objective, we used the convex2 objective as to get fractional solutions for our rounding procedures. We then benchmarked the tick, n -best and 1-best rounding schemes. The names will be suffixed with 2 to represent the convex2 objective. These results are summarized in Tables 8 and 9.

Figures A9, A10 and A11 show the rounding progress for each real instance with n -best, 1-best and tick rounding schemes. We observe that in many instances the n -best scheme predictably maintains a higher objective value throughout the process until it plummets at the end, in most cases below the objective for the tick based scheme.

From the results we can note that the real instances are not large enough to benefit much from the n -best scheme in terms of absolute runtime. Average improvement of 0.3% is a good result as the optimal values are on average only 0.7% better as was shown in Table 5.

In the artificial instances we can see the benefit of the n -best rounding scheme as even the biggest instances take less than 2 minutes to solve. Here the average improvement is 1% when the best possible improvement would be 7.1% with the convex2 objective, see Table 6. The 1-best rounding scheme gets closer to this ceiling with average 2.7% improvement but the runtimes suffer to make it less useful as a solution polishing procedure.

5.6 Flexible shifts

Finally we used the extended model to reveal how much more room there is for improving the results if we allow repositioning shift starts and the breaks within shifts. These results can be seen in Figures 6, 7 and 8. The results match our expectation that runtimes increase and the score gets better as we allow shifts to move more freely within the constraints set by the input.

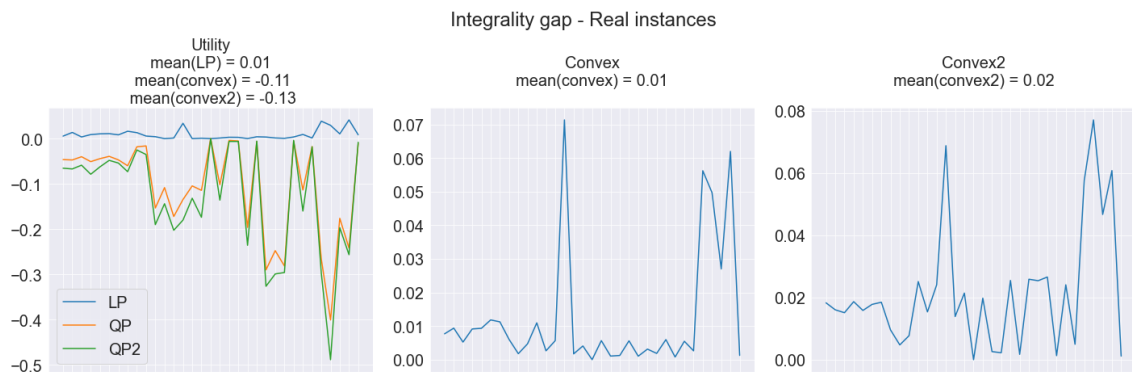


Figure 4: Fractional solutions for real instances compared to the integer solutions using the IG measure (38). Solutions evaluated by the utility (nonconvex), convex and convex2 objective in respective tables.

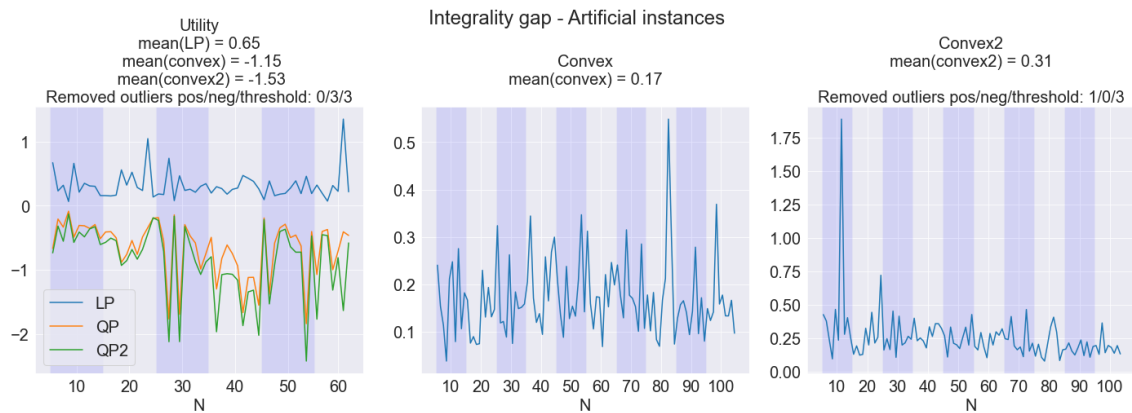


Figure 5: Fractional solutions for artificial instances compared to the integer solutions using the IG measure (38). Solutions evaluated by the utility (nonconvex), convex and convex2 objective in respective tables. Some outliers excluded in the plots for clarity but considered in the mean calculation.

	1-best2	n-best2	tick2
N	33	33	33
mean (%)	0.3%	-1.4%	0.3%
median (%)	0.1%	-0.9%	0.1%
range (%)	[-0.4%, 4.2%]	[-5.3%, 2.9%]	[-0.5%, 4.2%]
mean (s)	2.4 s	0.9 s	2.4 s
median (s)	1.4 s	0.7 s	1.4 s
range (s)	[0.1 s, 23.0 s]	[0.1 s, 6.4 s]	[0.1 s, 24.9 s]

Table 8: Rounding schemes for real instances, relative improvement (%) and runtimes (s) summarized

	1-best2	<i>n</i> -best2	tick2
N	100	100	100
mean (%)	2.7%	1.0%	0.8%
median (%)	-0.5%	-1.2%	-0.9%
range (%)	[-40.4%, 378.3%]	[-46.2%, 375.9%]	[-23.1%, 265.1%]
mean (s)	246.4 s	16.3 s	232.8 s
median (s)	77.5 s	6.9 s	70.6 s
range (s)	[0.4 s, 1257.4 s]	[0.3 s, 93.5 s]	[0.4 s, 1315.4 s]

Table 9: Rounding schemes for artificial instances, relative improvement (%) and runtimes (s) summarized

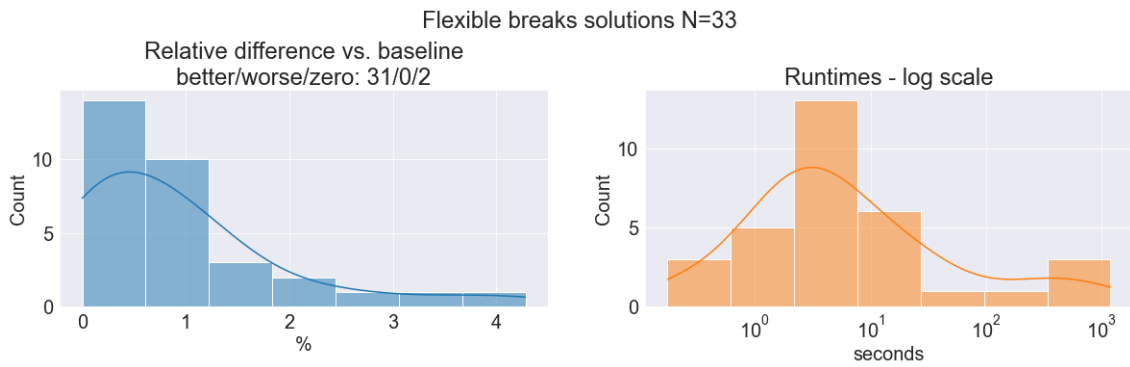


Figure 6: Relative difference and runtimes on real instances with flexible breaks.

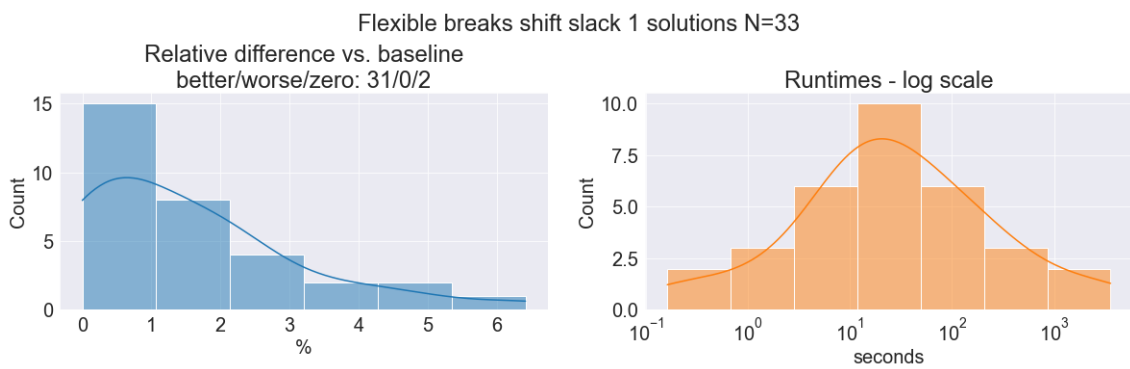


Figure 7: Relative difference and runtimes on real instances with slack 1.

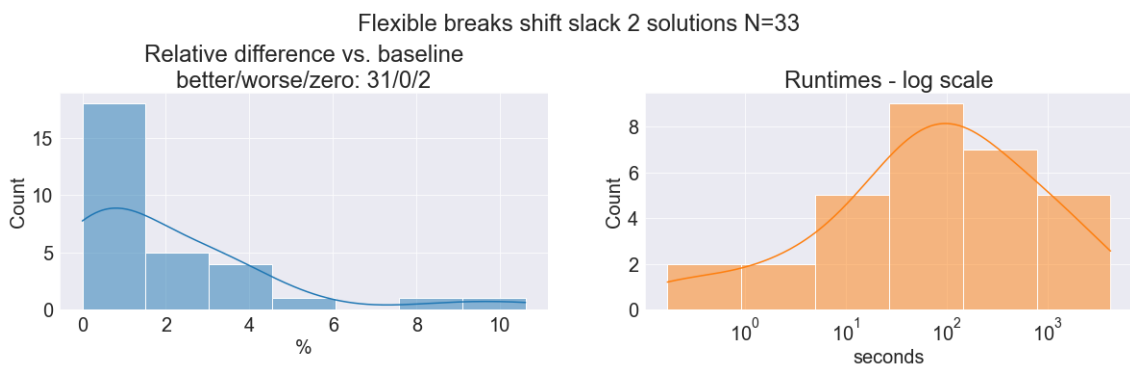


Figure 8: Relative difference and runtimes on real instances with slack 2.

6 Conclusion and future work

This thesis aimed to use mathematical optimization to aid in retail shift planning. In retail the target workforce is multi-skilled and has varied availability. The focus was on the particular case where shift and break placements have been decided in advance and then different tasks need to be assigned to these shifts by avoiding too many task switches.

We showed that this particular task assignment problem is NP-hard as it contains interval scheduling with nonidentical machines as a special case which 3-SAT can be reduced to.

By optimizing the utility function with GUROBI, which is a commercial MIP solver, we showed that the existing baseline solution can be improved by up to 4.2% (avg. 0.7%) on our real instances and by up to 785.5% (avg. 21.9%) on our artificial instances.

With our extended model we also showed that the break placement can be improved in almost all real instances. When we allowed the model to improve the shifts placements by moving them half an hour earlier or later within input limits the baseline could be improved by up to 10%. However the model becomes intractable as more flexibility is given to shift placements and thus cannot be used to solve the integrated problem as a whole.

We formulated a convex task switch objective, which rewards assigning the same task in consecutive time periods but does not exactly match the utility which we use to measure the solution quality. Using this convex objective our relaxed problem was optimized with the open source solver OSQP to get fractional solutions. These fractional solutions on real instances differed on average only 1% from the integer solution and 17% for artificial instances. Thus it may provide sufficient to use this fractional solution as-is to guide a local search for shift placements if the convex objective is used as the utility.

Next the fractional solutions were made integral by iterative rounding. This improved on the baseline solution for real instances while still maintaining a practical runtime. The optimum of the convex objective on artificial instances was on average worse than baseline and rounding the fractional solutions with this objective made it even worse. However, when the artificial instance baseline solutions were evaluated with the same convex objective as we used to optimize them there was much more room for improvement.

The core difference between the convex objective and the utility is the $\max(a, b)$ vs. $a + b$ definition of a task switch penalty from task a to b . Since the artificial instances have much more task switches this difference accumulates. For modelling purposes these both may prove sufficient to penalize task switches in which case it would be recommended to use the $a + b$ version or a scaled $0.5(a + b)$ version in the utility if there is a wish to use mathematical optimization for some part of this problem. Two further differences were how general work was additionally penalized and how continuing with the same task had an additional reward.

Since some of these differences could be at least partly moved over to static linear task rewards, we made a second adjusted version of the convex objective. This

adjusted convex objective performed much better on the artificial instances, for the faster n -best rounding scheme the median difference improved from -2.3% to -1.2% and the mean from -5.1% to 1% , while the longest runtime for the largest instances was under 2 minutes.

I would recommend that the utility should be defined as a convex function. For this problem it is likely that this would capture a quite similar real world utility. With a convex utility function all the optimization effort would go directly into improving the utility leading to even better results than our adjusted convex objective had.

As we want to get better at approximating real world practical problems we quickly end up with problems that have multiple objectives and constraints. The common approach is to generalize and simplify the problem until some structure appears which can be exploited in an algorithm. The alternative is to encapsulate this complexity in a constrained function which is then optimized with mathematical optimization.

Modern commercial solvers have gotten quite effective at optimizing arbitrary functions under constraints, particularly when the functions are convex. This can be utilized when designing heuristics as we can measure how well the heuristic performs on particular problem instances.

Mathematical optimization can then also be utilized in building these heuristics. It can provide global information on decision consequences in an otherwise greedy algorithm. The iterative rounding schemes we implemented demonstrate this idea. It combines the intuitiveness of greedy algorithms and the expressiveness of mathematical optimization.

6.1 Future work

The problem decomposes naturally in multiple different ways, particularly per employee, but these subproblems have dependencies to each other. It would be interesting to reformulate the problem into separate subproblems for each employee. These subproblems could then be used as a part of a master problem with column generation where the dual variables can be used to give the subproblems global information dependencies like task availabilities. These subproblems might include the shift placement decisions.

Semidefinite program relaxations, which have proved beneficial in some NP-hard combinatorial problems, could be utilized to find stronger relaxations, either for the current model or for a more complicated model which could for example incorporate shift placements.

References

- [1] F. Alizadeh. “Interior point methods in semidefinite programming with applications to combinatorial optimization”. In: *SIAM Journal on Optimization* 5.1 (1995), pp. 13–51.
- [2] E. M. Arkin and E. B. Silverberg. “Scheduling jobs with fixed start and end times”. In: *Discrete Applied Mathematics* 18.1 (1987), pp. 1–8.
- [3] K. R. Baker. “Workforce Allocation in Cyclical Scheduling Problems: A Survey”. In: *Operational Research Quarterly* 27.1 (1976), pp. 155–167.
- [4] C. Barnhart et al. “Branch-and-price: Column generation for solving huge integer programs”. In: *Operations Research* 46.3 (1998), pp. 316–329.
- [5] E. Boros and P. L. Hammer. “Pseudo-boolean optimization”. In: *Discrete Applied Mathematics* 123.1-3 (2002), pp. 155–225.
- [6] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [7] P. Brucker and L. Nordmann. “The k -track assignment problem”. In: *Computing* 52.2 (1994), pp. 97–122.
- [8] P. Brucker, R. Qu, and E. Burke. “Personnel scheduling: Models and complexity”. In: *European Journal of Operational Research* 210.3 (2011), pp. 467–473.
- [9] M. J. Brusco and L. W. Jacobs. “A simulated annealing approach to the solution of flexible labour scheduling problems”. In: *Journal of the Operational Research Society* 44.12 (1993), pp. 1191–1200.
- [10] M. C. Carlisle and E. L. Lloyd. “On the k -coloring of intervals”. In: *Discrete Applied Mathematics* 59.3 (1995), pp. 225–235.
- [11] N. Chapados, M. Joliveau, and L.-M. Rousseau. “Retail store workforce scheduling by expected operating income maximization”. In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2011, pp. 53–58.
- [12] R. Cuevas et al. “A mixed integer programming approach to multi-skilled workforce scheduling”. In: *Journal of Scheduling* 19.1 (2016), pp. 91–106.
- [13] G. B. Dantzig. “A comment on Edie’s “Traffic delays at toll booths””. In: *Journal of the Operations Research Society of America* 2.3 (1954), pp. 339–341.
- [14] G. B. Dantzig. “Reminiscences about the origins of linear programming”. In: *Mathematical Programming the State of the Art*. Springer, 1983, pp. 78–86.
- [15] G. B. Dantzig, A. Orden, P. Wolfe, et al. “The generalized simplex method for minimizing a linear form under linear inequality restraints”. In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 183–195.
- [16] J. Desrosiers and M. E. Lübbecke. “Branch-price-and-cut algorithms”. In: *Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Chichester (2011), pp. 109–131.

- [17] L. Di Gaspero et al. “The minimum shift design problem”. In: *Annals of Operations Research* 155.1 (2007), pp. 79–105.
- [18] J. Farkas. “Theorie der einfachen Ungleichungen.” In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1902.124 (1902), pp. 1–27.
- [19] M. X. Goemans and D. P. Williamson. “Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming”. In: *Journal of the ACM* 42.6 (1995), pp. 1115–1145.
- [20] M. Hojati and A. S. Patil. “An integer linear programming-based heuristic for scheduling heterogeneous, part-time service employees”. In: *European Journal of Operational Research* 209.1 (2011), pp. 37–50. ISSN: 0377-2217.
- [21] M. Holliday. *How Much of Sales or Gross Revenue Should go Toward my Small Business Payroll?* <https://www.netsuite.com/portal/resource/articles/financial-management/small-business-payroll-percentage.shtml>. Dec. 2020.
- [22] L. W. Jacobs and S. E. Bechtold. “Labor utilization effects of labor scheduling flexibility alternatives in a tour scheduling environment”. In: *Decision Sciences* 24.1 (1993), pp. 148–166.
- [23] R. Jonker and A. Volgenant. “A shortest augmenting path algorithm for dense and sparse linear assignment problems”. In: *Computing* 38.4 (1987), pp. 325–340.
- [24] Ö. Kabak et al. “Efficient shift scheduling in the retail sector through two-stage optimization”. In: *European Journal of Operational Research* 184.1 (2008), pp. 76–90.
- [25] L. V. Kantorovich. “The mathematical method of production planning and organization”. In: *Management Science* 6.4 (1939), pp. 363–422.
- [26] N. Karmarkar. “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. 1984, pp. 302–311.
- [27] V. Klee and G. J. Minty. “How good is the simplex algorithm”. In: *Inequalities* 3.3 (1972), pp. 159–175.
- [28] A. W. Kolen et al. “Interval scheduling: A survey”. In: *Naval Research Logistics* 54.5 (2007), pp. 530–543.
- [29] M. Y. Kovalyov, C. T. Ng, and T. E. Cheng. “Fixed interval scheduling: Models, applications, computational complexity and algorithms”. In: *European Journal of Operational Research* 178.2 (2007), pp. 331–342.
- [30] M. Krishnamoorthy and A. T. Ernst. “The personnel task scheduling problem”. In: *Optimization Methods and Applications*. Springer, 2001, pp. 343–368.
- [31] H. W. Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97.

- [32] N. Kyngäs, K. Nurmi, and J. Kyngäs. “Crucial components of the PEAST algorithm in solving real-world scheduling problems”. In: *Lecture Notes on Software Engineering* 1.3 (2013), p. 230.
- [33] T. Lapègue, O. Bellenguez-Morineau, and D. Prot. “A constraint-based approach for the shift design personnel task scheduling problem with equity”. In: *Computers & Operations Research* 40.10 (2013), pp. 2450–2465.
- [34] Q. Lequy, G. Desaulniers, and M. M. Solomon. “A two-stage heuristic for multi-activity and task assignment to work shifts”. In: *Computers & Industrial Engineering* 63.4 (2012), pp. 831–841.
- [35] Q. Lequy et al. “Assigning multiple activities to work shifts”. In: *Journal of Scheduling* 15.2 (2012), pp. 239–251.
- [36] L. Lovász. “Semidefinite programs and combinatorial optimization”. In: *Recent Advances in Algorithms and Combinatorics* (2003), pp. 137–194.
- [37] F. Margot. “Symmetry in integer linear programming”. In: *50 Years of Integer Programming 1958-2008* (2010), pp. 647–686.
- [38] S. K. Mirrazavi and H. Beringer. “A web-based workforce management system for Sainsburys Supermarkets Ltd”. In: *Annals of Operations Research* 155.1 (2007), pp. 437–457.
- [39] D. R. Morrison et al. “Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning”. In: *Discrete Optimization 19* (2016), pp. 79–102.
- [40] N. Musliu, A. Schaerf, and W. Slany. “Local search for shift design”. In: *European Journal of Operational Research* 153.1 (2004), pp. 51–64.
- [41] V. Nissen and M. Günther. “Automatic generation of optimised working time models in personnel planning”. In: *International Conference on Swarm Intelligence*. Springer. 2010, pp. 384–391.
- [42] K. Nurmi, N. Kyngäs, and J. Kyngäs. “Workforce Optimization: the General Task-based Shift Generation Problem.” In: *IAENG International Journal of Applied Mathematics* 49.4 (2019).
- [43] S. Pan et al. “Solving a Multi-Activity Shift Scheduling Problem with a Tabu Search Heuristic”. In: *PATAT 2016: Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling*. 2016, pp. 317–326.
- [44] R. Pastor and J. Olivella. “Selecting and adapting weekly work schedules with working time accounts: A case of a retail clothing chain”. In: *European Journal of Operational Research* 184.1 (2008), pp. 1–12.
- [45] Y. Qu and T. Curtois. “Solving the Multi-activity Shift Scheduling Problem using Variable Neighbourhood Search.” In: *International Conference on Operations Research and Enterprise Systems*. 2020, pp. 227–232.

- [46] C.-G. Quimper and L.-M. Rousseau. “A large neighbourhood search approach to the multi-activity shift scheduling problem”. In: *Journal of Heuristics* 16.3 (2010), pp. 373–392.
- [47] P. Smet et al. “The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights”. In: *Omega* 46 (2014), pp. 64–73.
- [48] G. M. Thompson and M. E. Pullman. “Scheduling workforce relief breaks in advance versus in real-time”. In: *European Journal of Operational Research* 181.1 (2007), pp. 139–155.
- [49] J. Van den Bergh et al. “Personnel scheduling: A literature review”. In: *European Journal of Operational Research* 226.3 (2013), pp. 367–385. ISSN: 0377-2217.
- [50] L. Vandenberghe and S. Boyd. “Semidefinite programming”. In: *SIAM review* 38.1 (1996), pp. 49–95.
- [51] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge university press, 2011.
- [52] P. Wolfe. “The simplex method for quadratic programming”. In: *Econometrica: Journal of the Econometric Society* (1959), pp. 382–398.
- [53] S. Zolfaghari et al. “Application of a genetic algorithm to staff scheduling in retail sector”. In: *International Journal of Industrial and Systems Engineering* 5.1 (2010), pp. 20–47.

A Appendix

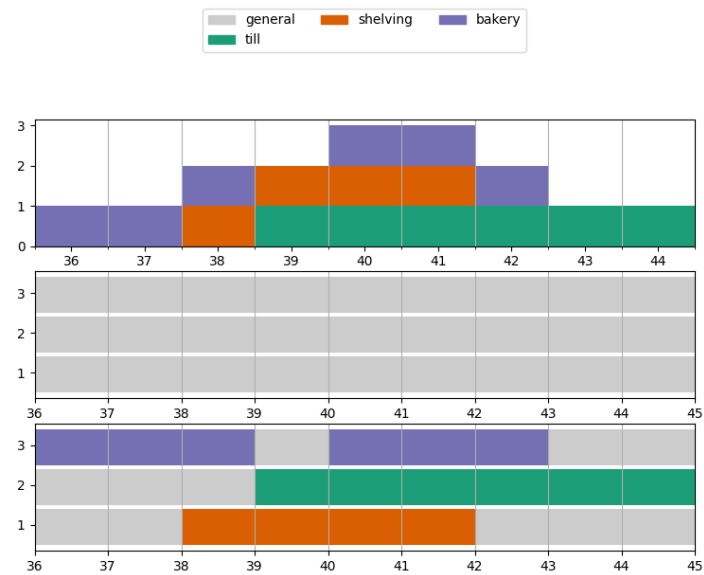


Figure A1: A variation of the problem in Figure 3 with 3 employees available.

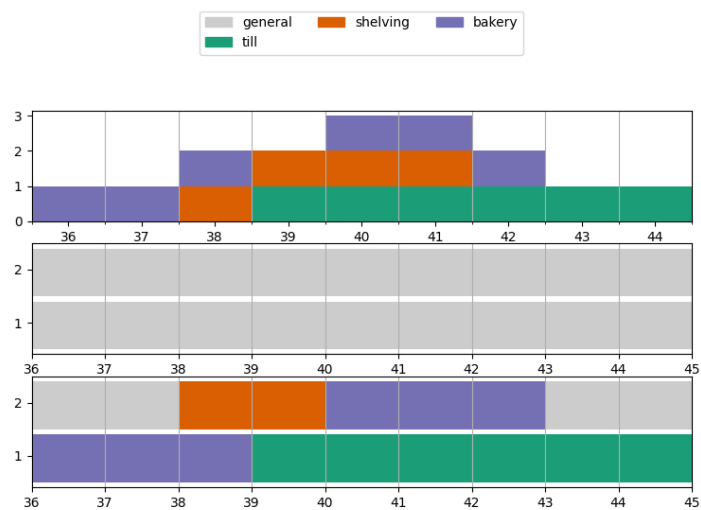


Figure A2: A variation of the problem in Figure 3 with bakery task assigned a higher priority.

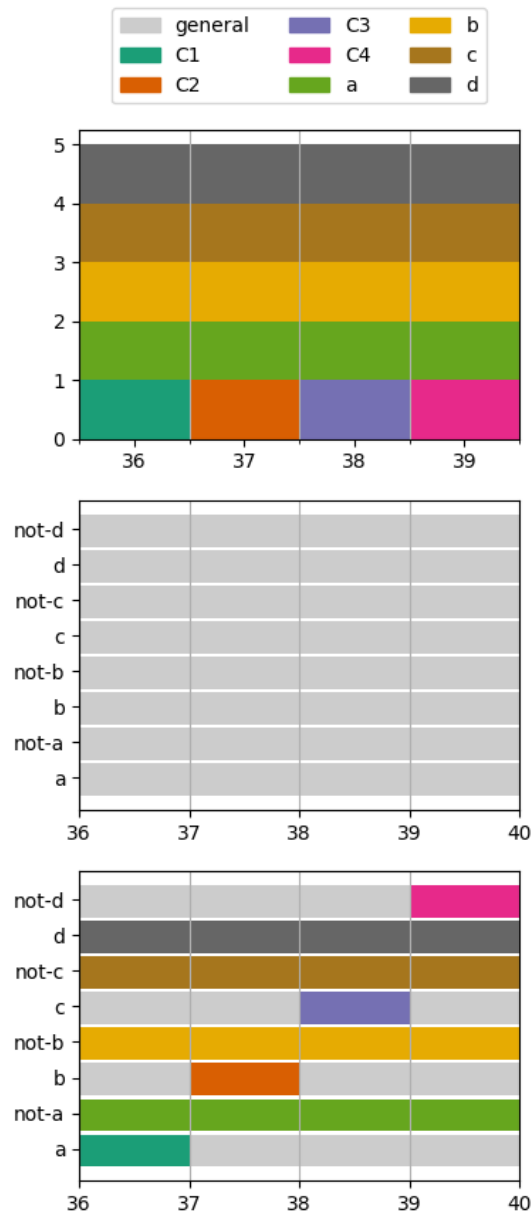


Figure A3: A example of a 3-SAT instance $(a \vee b \vee c) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee c \vee d) \wedge (a \vee \neg b \vee \neg d)$ reduced to our task assignment problem and solved. This assignment represent the solution $\{a, b, c, \neg d\}$.

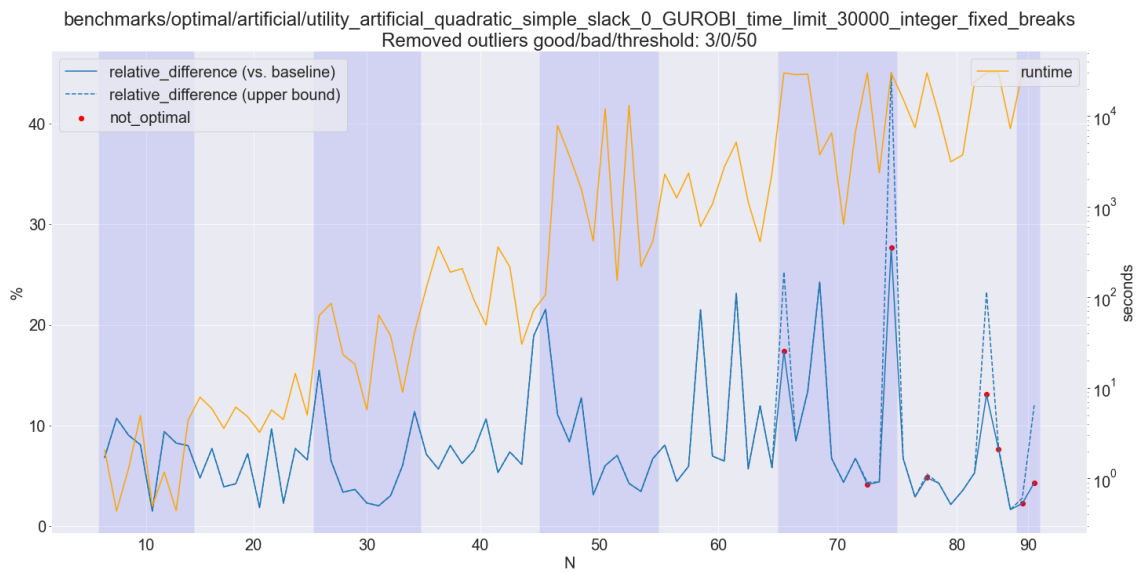


Figure A4: Optimal solutions to artificial instances compared to the baseline solution. Instances are grouped on the x-axis by increasing employee number N . For clarity three outliers have been removed which were over 50% better.

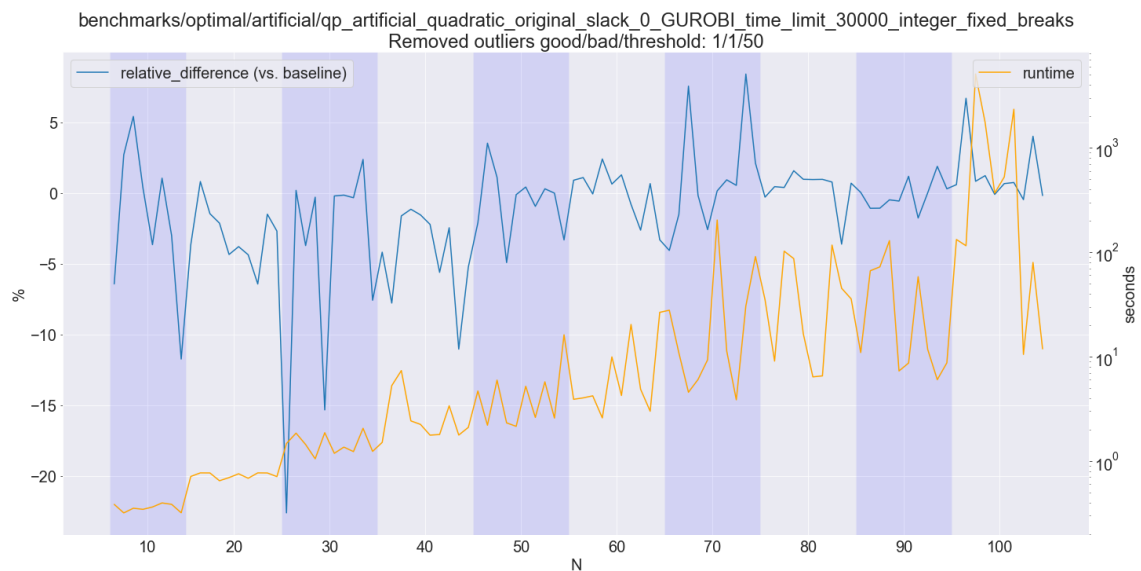


Figure A5: Optimal convex solutions to artificial instances, evaluated with the utility and compared to the baseline solution. Instances are grouped on the x-axis by increasing employee number N .

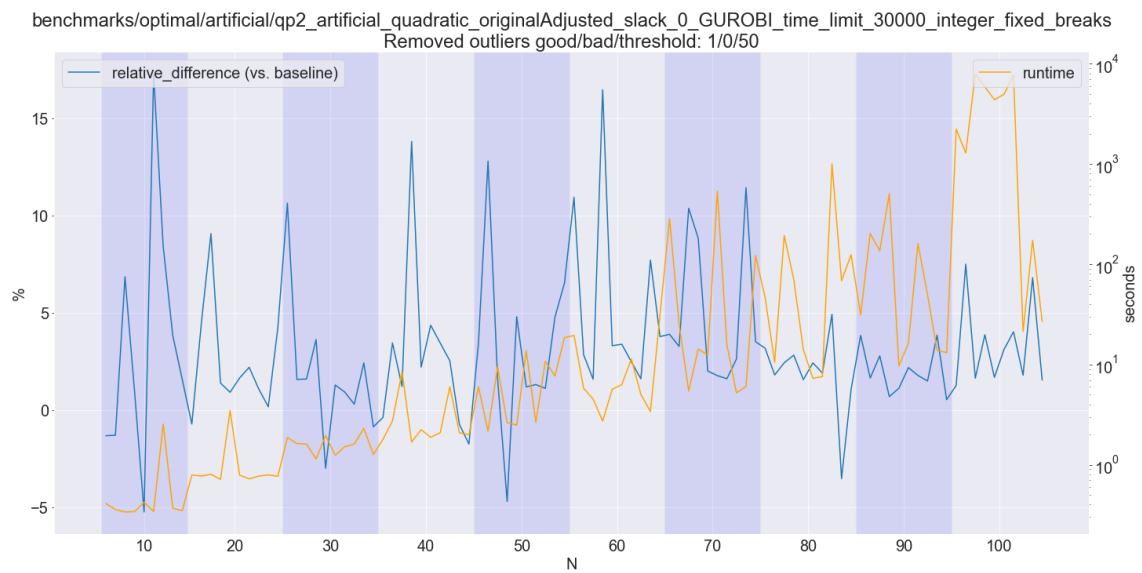


Figure A6: Optimal convex2 solutions to artificial instances, evaluated with the utility and compared to the baseline solution. Instances are grouped on the x-axis by increasing employee number N .

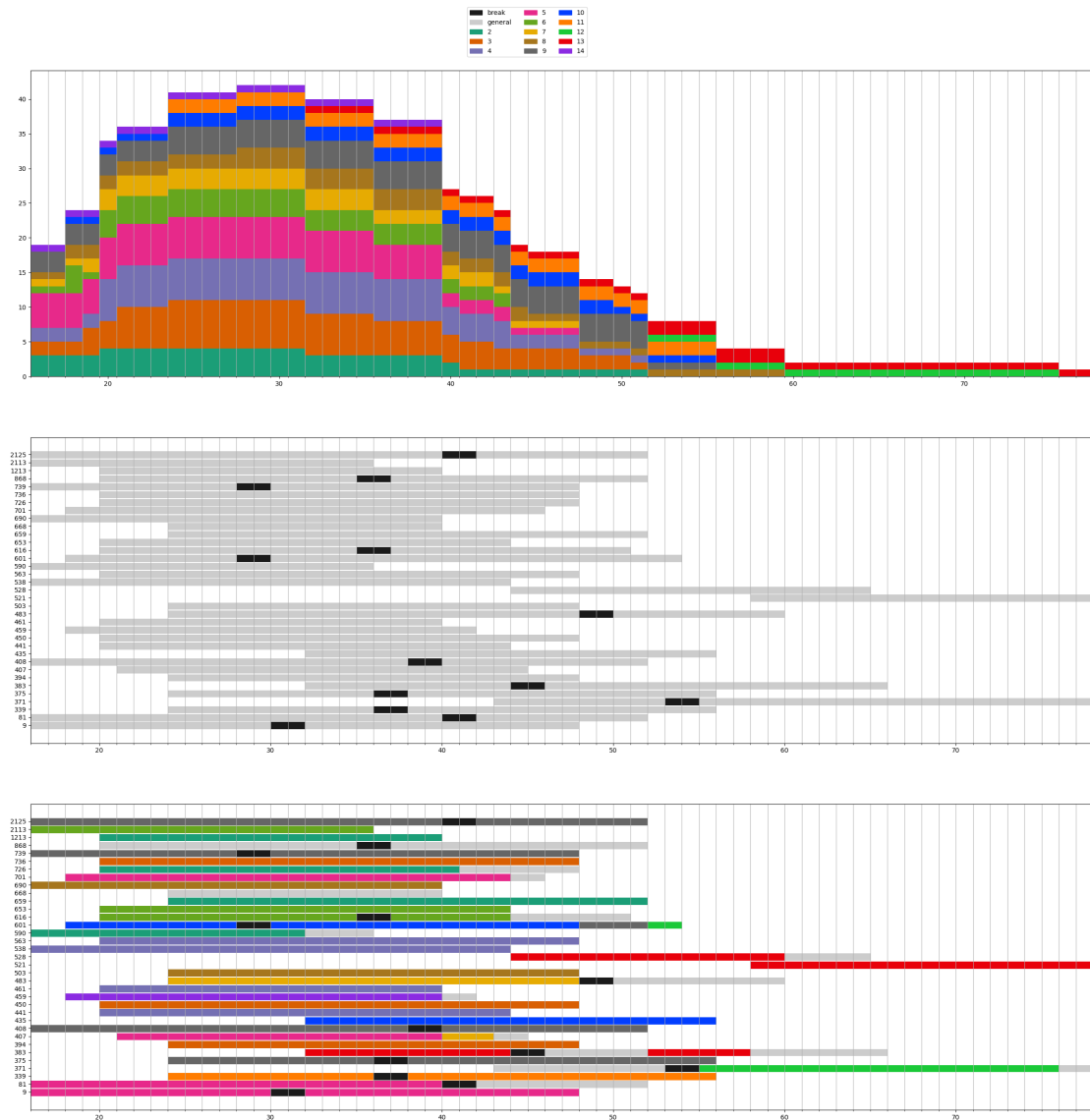


Figure A7: Solution for a real instance with 40 employees

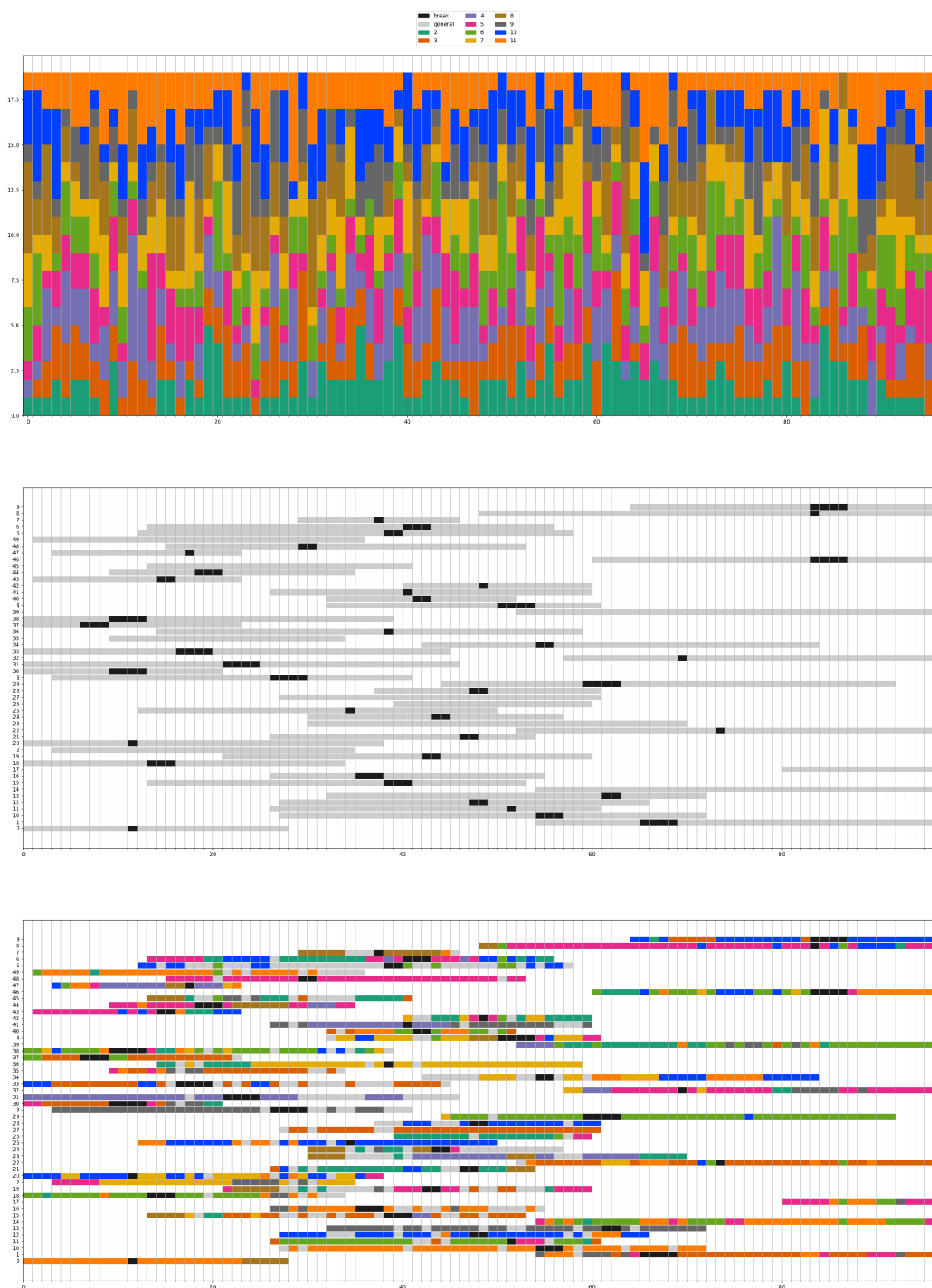


Figure A8: Solution for an artificial instance with 50 employees

	runtime	score	baseline score	relative difference	optimal
AE	0.55	5525	5301	4.23	True
AF	5.51	6745	6587	2.4	True
AD	14705	11229	10975	2.31	True
Y	1.53	6260	6128	2.15	True
K	0.84	4872	4805	1.39	True
H	3.2	13491	13386	0.78	True
O	0.63	4805	4772	0.69	True
A	2.82	9001	8941	0.67	True
AC	6.07	7818	7767	0.66	True
L	0.58	4970	4940	0.61	True
C	1.17	8334	8284	0.6	True
F	4.02	8350	8300	0.6	True
N	0.87	4183	4158	0.6	True
R	0.75	5283	5254	0.55	True
B	91.39	21767	21648	0.55	True
E	32.53	21825	21716	0.5	True
G	1.85	8578	8538	0.47	True
AB	0.23	7919	7894	0.32	True
D	2.55	8717	8691	0.3	True
AA	4.52	7449	7430	0.26	True
P	0.55	4166	4156	0.24	True
W	2.29	6235	6222	0.21	True
U	1.67	6652	6643	0.14	True
V	3.11	29654	29617	0.12	True
S	3.05	27765	27732	0.12	True
T	2.46	27331	27300	0.11	True
Z	2.63	29027	28997	0.1	True
X	1.98	6096	6090	0.1	True
M	0.72	4430	4427	0.07	True
J	2.07	8620	8618	0.02	True
Q	0.19	2452	2452	0	True
I	0.84	8464	8464	0	True
AG	0.13	1774	1774	0	True

Table A1: Optimal solutions to real instances compared to the baseline solution. Sorted by relative difference.

Name	#	(a)	(b)	(c)	Graph	Algorithm	Complexity
ISP	1	No	No	No	Interval graph maximum independent set	Greedy	$\mathcal{O}(n \log n)$
k -machine ISP	k	No	No	No	Interval graph k -coloring*	Greedy	$\mathcal{O}(n \log n)$
Weighted ISP	1	No	No	Yes	Maximum-weight independent set	DP	$\mathcal{O}(n \log n)$
k -machine weighted ISP	k	No	No	Yes	Maximum-weight k -coloring*	LP min-cost flow [10]	$\mathcal{O}(n^2 \log n)$
ISP with non-identical machines	k	Yes	No	No	Graph k -coloring	Longest path DAG [2]	$\mathcal{O}(n^{k+1})$
k -track assignment problem	k	No	Yes	No	Circular-arc graph coloring	Longest path DAG [7]	$\mathcal{O}(n^{k+1})$

(a) Machine skill/job restrictions

(b) Machine availability restrictions

(c) Task weights

* Finding the maximum (weight) subgraph where there are no cliques larger than k

Table A2: Variations of ISP summarized

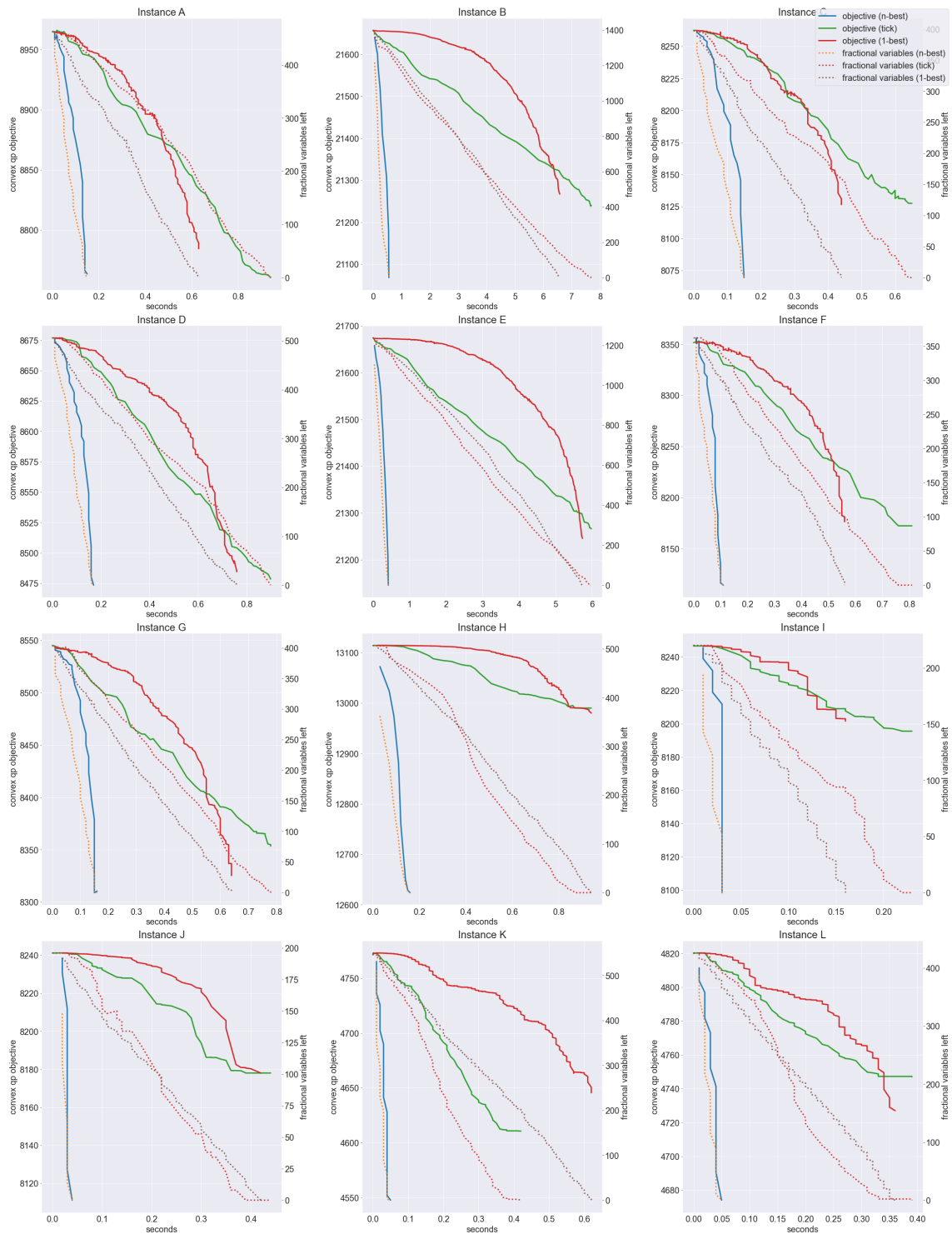


Figure A9: Rounding progress for all real instances, part1

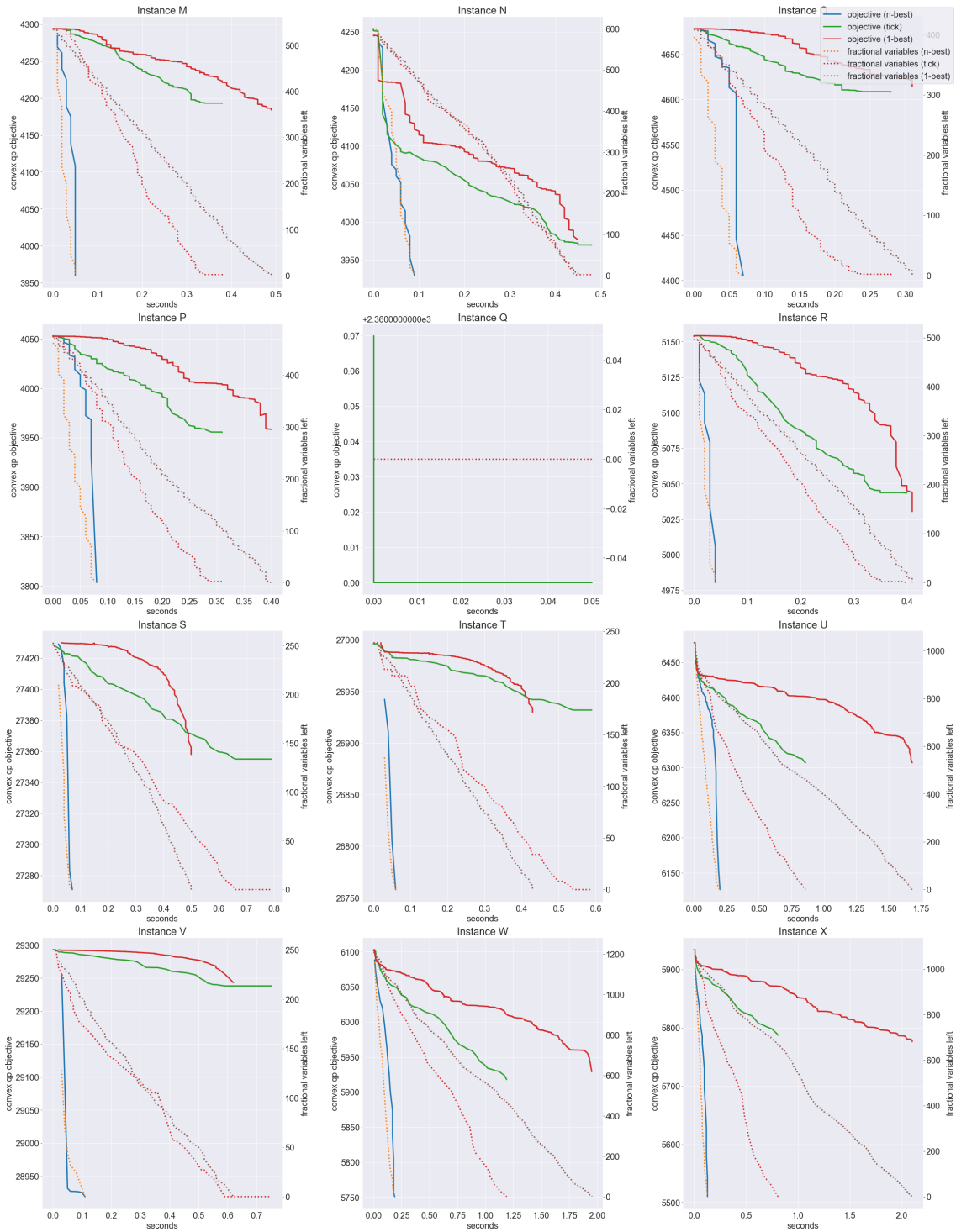


Figure A10: Rounding progress for all real instances, part2

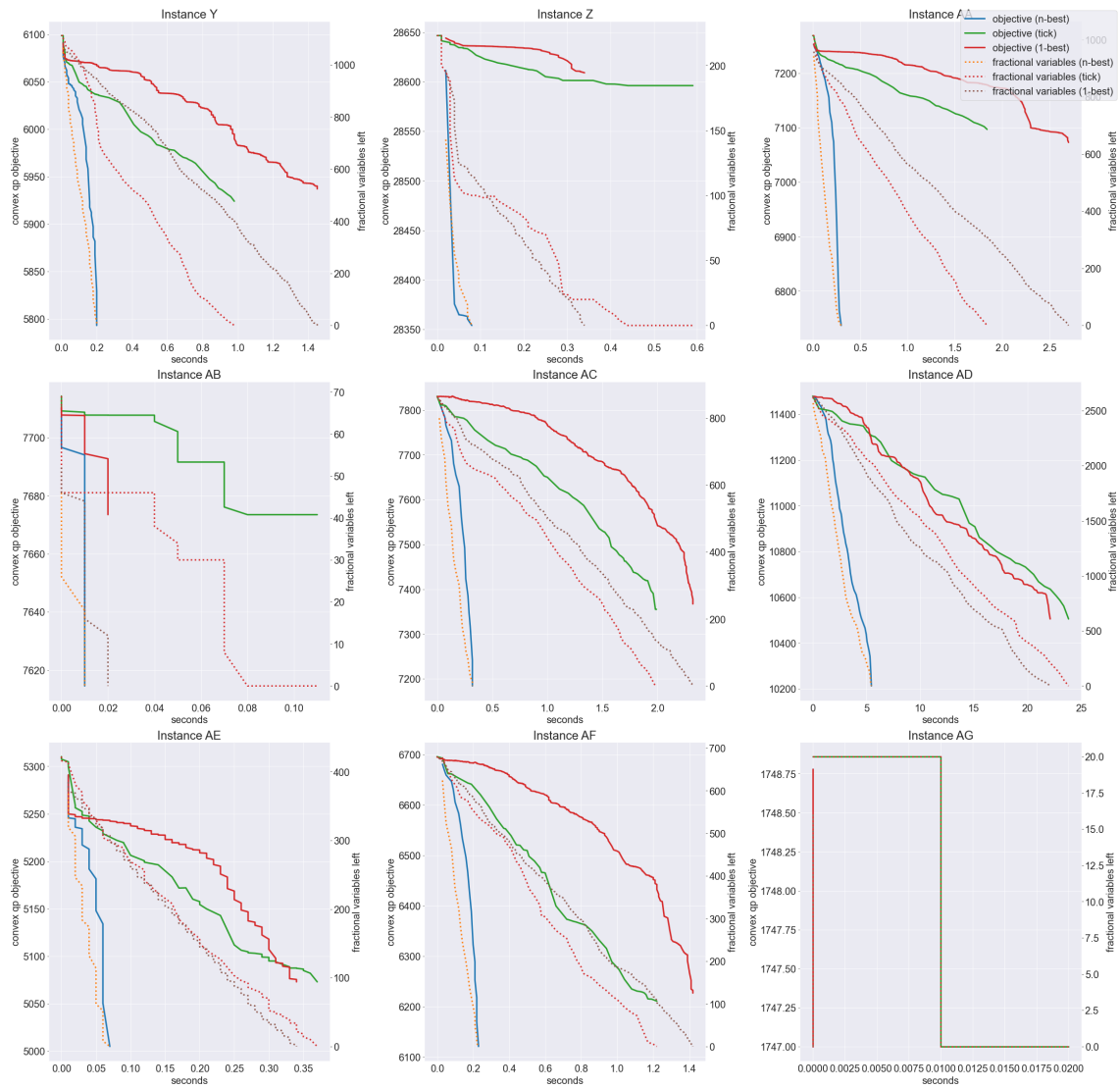


Figure A11: Rounding progress for all real instances, part3