UCLouvain

École polytechnique de Louvain

Optimization of production planning with resource allocation

Author: **Florian Knop**
Supervisors: **Pierre Schaus, Charles Thomas**
Reader: **Hélène Verhaeghe**
Academic year 2018–2019
Master [120] in Computer Science

Contents

Chapter 1

Introduction

Village n°1 is a belgian company employing persons with disabilities. They offer services to companies and private individuals such as industrial jobs. They are currently in the process of automating the way they schedule these jobs. The aim of this thesis is to solve their resource allocation problem automatically. First, we will introduce two models to solve this problem: a Mixed Integer Programming and a Constraint Programming model. We will then analyze and compare the performance of both models.

TODO
TODO
TODO
TODO

This thesis is organized as follows

Chapter 2 introduces the resource allocation problem of Village n°1.

Chapter 3 describes the state-of-the-art in the domains of Mixed Integer Programming and Constraint Programming.

Chapter 4 gives a formal definition of both MIP and CP models.

Chapter 5 describes the implementation of the models.

Chapter 6 presents the carried experiments and performance results of both models.

Chapter 7 TODO

Chapter 2

The Village n^{o}1 problem

This chapter presents the resource allocation problem. We first introduce the general problem and its constraints, the formal models are described in Chapter 4.

The Village n^{o}1 problem consists of allocating resources to work demands. This problem is a type of staff scheduling problem, it can be seen as a variant of the well known *Nurses Scheduling Problem* (NSP) [1]. The goal of the NSP is to assign nurses to shifts such that the entire schedule is satisfied. This type of problem often have hard constraints to state restrictions and soft constraints to state preferences.

Village n^{o}1 has internal work for their employees but also receives external labor requests. The problem is separated in multiple time periods all equal in time. A demand often occurs in multiple time slots and consists of a required number of workers, an eventual work location and additional resources like machines or vehicles. Each demand has:

– A given set of time periods.

– A required number of workers per period.

– Some skills requirements to be fullfilled by the workers. It imposes that some workers have the needed capacities to work at a given position (e.g. package lifter).

– A list of machines to perform the work.

– An eventual list of possible locations where the demand can be executed and a vehicle to drive the workers to destination. A demand can only have a location if it is an external labor request. Internal work to the company use predefined locations.

– An eventual need for a worker supervisor which will supervise the group.

Each worker has:

- Some skills and restrictions (e.g. package lifter, supervisor, etc.)
- A list of availabilities at which the worker can work.
- A list of incompatibilities with other workers (i.e. workers that can't work together).
- A list of incompatibilities with clients (i.e. workers that can't work for clients).

The goal is to assign workers, machines and locations to a list of demands over the set of all time slots. Each resource can only be assigned once per time period and need to satisfy all the constraints stated by the demand. The sub-goal is to also assign workers in such a way that they work for the longest time possible at the same position and such that the assignments between workers are balanced throughout the entire schedule.

2.1 Constraints

2.1.1 Hard Constraints

Respect worker availabilities

A worker has a set of availabilities and should not be assigned to a shift when not available.

Respect demand occurences

A demand has a set of time periods in which it occurs, no workers should be assigned to that demand if the demand is not occuring.

No worker should be assigned twice for the same period

A worker obviously can't work at two positions at the same time.

Required number of workers

A demand has a needed number of workers to be satisfied. For each time period a demand is occuring, it should have the required number of workers assigned to it.

Skill restrictions

Each position of a demand might require skills to be satisfied. To be assigned to that position, a worker must have the required skills.

Worker-worker incompatibilities

Workers might be incompatible with each other. Such workers can't be assigned together at the same time period.

Worker-client incompatibilities

A worker and a client might be incompatible with each other. If this is the case, the worker must not be assigned at a demand for such client.

The required machines must always be assigned

A demand has machine needs. Such machines should always be assigned for a demand to be satisfied.

No machines should be assigned twice for the same period

A machine is assigned for the entirety of a demand. It can be used for other demands that do not overlap in time with the first one. But it can never be assigned twice for the same time period.

The location assigned must be in the set of possible locations

A demand has a set of possible locations. Only one of those locations can be assigned to that demand.

No location should be assigned twice for the same period

As with machines, locations must be assigned only once per time period.

2.1.2 Soft Constraints

Client-worker preference

A client might prefer some workers over others. We use a soft constraint for this as it might not always be possible to satisfy.

Contiguous shifts

A demand consists of multiple positions over a period of time. For each position, a worker should keep working at that position for the longest time possible. We want to avoid the hassle of changing shift everytime. As this constraint is harder to solve, we express it as a soft constraint and minimize the number of violations.

Working requirements

Workers can have minimum and maximum working periods. We want to make sure that these requirements are respected as much as possible.

Chapter 3

State of the art

3.1 Mixed Integer Programming

The most common *Mixed Integer Programming* (MIP) problems are of the form:

$$\begin{aligned}
\min \quad & \boldsymbol{c}^T\boldsymbol{x} && (3.1)\\
\text{s.t} \quad & A\boldsymbol{x} = \boldsymbol{b} && (3.2)\\
& \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} && (3.3)\\
& \text{Some or all } x_i \text{ must take integer values} && (3.4)
\end{aligned}$$

(3.1) is the problem objective. $\boldsymbol{c}^T$ is the vector of coefficient, $\boldsymbol{x}$ is the vector of variables. (3.2) are the linear constraints. $\boldsymbol{b}$ is a vector of bounds while A is a matrix of coefficients for the constraints. (3.3) are the bound constraints. Each x_i can only take values between l_i and u_i. And finally, (3.4) states the integrality constraints over some or all variables.

MIP problems are usually solved using a branch-and-bound algorithm [2]. The process is as follow: we start with the MIP formulation and remove all integrality constraints to create a resulting linear-programming (LP) relaxation to the original problem. The relaxation can be solved easily compared to the original problem. The result might satisfy all integrality constraints and be a solution to the original problem. But more often than not, a variable has a fractional value. We can then solve two relaxations by imposing two additional constraints. For example, if x takes value 5.5, we add the following linear constraints: $x \leq 5.0$ and $x \geq 6.0$. This process is repeated throughout the search tree (Figure 3.1) a valid solution is found. More techniques are used to find solution more efficiently. Each solver uses its own algorithm (e.g Gurobi Optimizer [2]).

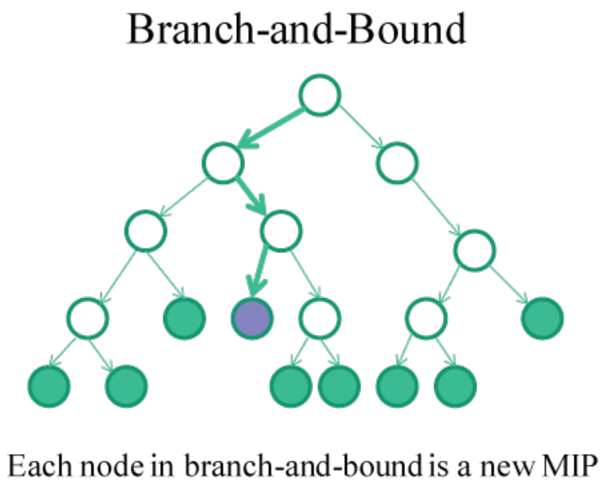

Figure 3.1: MIP Branch & Bound search tree [2]

3.1.1 Gurobi Optimizer

The *Gurobi Optimizer* [3] is a state-of-the-art commercial solver for mathematical programming. Gurobi includes multiple solvers, among those: (i) Linear Programming (LP); (ii) Mixed Integer Linear Programming (MILP), abbreviated as MIP.

The Gurobi Optimizer is used by more than 2100 companies in over 40 industries at this time. It allows describing business problems as mathematical models. It also supports a lot of programming interfaces in a variety of programming languages like C++, Java, Python, C#.

3.2 Constraint Programming

A *Constraint Satisfaction Problem* (CSP) consists of a set of n variable, $\{x_1, \ldots, x_n\}$; a domain D_i of possible values for each variable x_i, $1 \leq i \leq n$; and a collection of m constraints $\{C_1, \ldots, C_m\}$. Each constraint C_j, $1 \leq j \leq m$, is a constraint over some set of variables called the scheme of the constraint. The size of this set is known as the arity of the constraint. A solution to a CSP is an assignment of values $a_i \in D_i$ to x_i, that satisfies all of the constraints. [4]

3.2.1 OscaR

OscaR [5] is a Scala toolkit for solving Operations Research problems. OscaR has multiple optimization techniques available: (i) Constraint Programming; (ii) Constraint Based Local Search (CBLS); (iii) Derivative Free Optimization; (iv) Visualization.

The project is mainly developped by UCLouvain and the research group of Pierre Schaus. But some companies like *N-Side* and *CETIC* allocate resources to improve it.

The library of OscaR in which this project is interested in is the Constraint Programming library. It offers a lot of existing constraints and abstractions. Some black-box searches are also implemented but we can bring our own heuristics to drive the search forward.

Chapter 4

Models for the Village n^{o}1 problem

In this chapter, we present models for both Mixed Integer Programming and Constraint Programming. We first start by presenting constraints that need to be respected. We then present formal notations used by both models. The mathematical (MIP) model is presented first for reference followed by the CP model which is an adaptation of the mathematical model.

4.1 Notations

– Set of periods:

$$T = \{0, \ldots, n \mid n \in \mathbb{N}\}$$

– Set of workers:

$$W = \{w_0, \ldots, w_n \mid n \in \mathbb{N}\}$$

  – $w^T \subseteq T$: Availabilities of a worker:

– Set of machines:

$$M = \{m_0, \ldots, m_n \mid n \in \mathbb{N}, m_n \in \mathbb{N}\}$$

  Let's also define the set of machines for a given machine value (i.e name). M_i is the set of machines that takes the value i.

$$M_i = \{m_j \mid m_j = i, \forall j \in M\}$$

– Set of vehicles:

$$V = \{v_0, \dots, v_n \mid n \in \mathbb{N}\}$$

This could also be expressed as a subset of machines:

$$V \subseteq M$$

– Set of zones:

$$Z = \{z_0, \dots, z_n \mid n \in \mathbb{N}\}$$

– Set of demands:

$$D = \{d_0, \dots, d_n \mid n \in \mathbb{N}\}$$

  – $d^w \in \mathbb{N}$: Required number of workers for this demand
  – $d^T \subseteq T$: Possible periods for a demand
  – $d^Z \subseteq Z$: Possible zones for a demand
  – $d^M \subseteq M$: List of required machines by the demand
  – $d^c \in C$: Client for that demand
  – $d^S \in S$: List of skill required by the demand (each skill need to have a different worker)
  – d^{s_0}: The first skill in d^S
  – d^{S^+}: Set of additional skills that can be satisfied by any worker in that demand
  – $d^{S_0^+}$: The first skill in d^{S^+}
  – $d^P \in \{0, \dots, d^w - 1\}$: List of positions
  – $d^O \subseteq D$: Set of overlapping demands in time for that demand. e.g. the overlapping demands for demand 1 is d_1^O

Let's also define the set of demands where the client c: $D_c = \{d \mid d^c = c\}$

– Set of clients:

$$C = \{c_0, \dots, c_n \mid n \in \mathbb{N}\}$$

– Set of skills:

$$S = \{s_0, \dots, s_n \mid n \in \mathbb{N}\}$$

Let's also define the set of workers that satisfy a skill or skill set:

$$W_s \subseteq W, s \in S$$

– Set of working requirements:

$$R = \{r_0, \ldots, r_n \mid n \in \mathbb{N}\}$$

 – r_w: The worker concerned with this requirement
 – r_{min}: The minimum number of periods the worker has to work
 – r_{max}: The maximum number of periods the worker has to work

– Set of incompatibilities between workers:

$$I_{ww} = \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid w_i, w_j \in W, w_i \neq w_j\}$$

– Set of incompatibilities between workers and clients:

$$I_{wc} = \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid w_i \in W, c_j \in C\}$$

4.2 Mixed Integer Programming Model

We first start by presenting the mathematical model, we describe the variables needed to model our problem and the constraints associated to them.

4.2.1 Variables

To represent our problem in MIP, we will need three types of variables, one per resource.

$$w_{ijkl} = \begin{cases} 1 & \text{if worker } i \text{ is working at time } j \text{ for demand } k \text{ at position } l \\ 0 & \text{otherwise} \end{cases}$$

$$s_{jkl} = \begin{cases} 1 & \text{if no worker is assigned at time } j \text{ for demand } k \text{ at position } l \\ 0 & \text{otherwise} \end{cases}$$

$$m_{ij} = \begin{cases} 1 & \text{if machine } i \text{ is used for demand } j \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if zone } i \text{ is used for demand } j \\ 0 & \text{otherwise} \end{cases}$$

This is in fact a binary Integer Programming model as every variables is a $\{0, 1\}$ integer.

As solution can be partial, we need to introduce a way to allow the absence of worker for a given position. In MIP, we model this by having the variables s_{jkl}, s for *sentinel*. This variable is one, if and only if all the corresponding worker variables $(w_{ijkl}, \forall_i)$ are equal to zero. The goal will be to minimize the number of sentinel variables assigned to one.

4.2.2 Complete Model

$$
\begin{aligned}
\min \quad & \sum_{k\in D}\sum_{l\in d_k^P}\sum_{i\in W} min(\sum_{j\in T} w_{ijkl}, 1) && \text{(4.1a)}\\
& + \sum_{j\in T}\sum_{k\in D}\sum_{l\in d_k^P} s_{jkl} && \text{(4.1b)}\\
& + \sum_{r\in R}\Big(max(r_{min} - occ_{r_w}, 0) + max(r_{max} - occ_{r_w}, 0)\Big) && \text{(4.1c)}
\end{aligned}
$$

$$
\begin{aligned}
\text{s.t} \quad & \sum_{i\in W} w_{ijkl} + s_{jkl} = 1, && \forall k \in D, j \in d_k^T, l \in d_k^P && (4.2)\\
& \sum_{k\in D}\sum_{l\in d_k^P} w_{ijkl} \le 1, && \forall i \in W, j \in T && (4.3)\\
& t_j \notin d_k^T \implies \forall i, l \; w_{ijkl} = 0, && \forall j \in T, k \in D && (4.4)\\
& t_j \notin w_i^T \implies \forall k, l \; w_{ijkl} = 0, && \forall j \in T, i \in W && (4.5)\\
& t_j \notin d_k^T \implies \forall l \; s_{jkl} = 0, && \forall j \in T, k \in D && (4.6)\\
& t_j \notin w_i^T \implies \forall l \; s_{jkl} = 0, && \forall j \in T, i \in W && (4.7)\\
& \sum_{l\in d_k^P} w_{ajkl} + w_{bjkl} < 2, && \forall (a,b) \in I_{ww}, j \in T, k \in D && (4.8)\\
& d_k^c = c \implies \forall l \; w_{ijkl} = 0, && \forall (i,c) \in I_{wc}, j \in T, k \in D && (4.9)\\
& w_{ijkl} = 0, && \forall j \in T, k \in D, l \in d_k^P, && \\
& && i \in W \setminus W_{d_k^{s_l}} && (4.10)\\
& \sum_{l\in d_k^P} w_{ijkl} \ge 1, && \forall j \in T, k \in D, s \in d_k^{S+}, i \in W_{d_k^{S+}} && (4.11)\\
& z_i \notin d_j^Z \implies z_{ij} = 0, && \forall i \in Z, j \in D && (4.12)\\
& |d_j^Z| > 0 \implies \sum_{i\in Z} z_{ij} = 1, && \forall j \in D && (4.13)\\
& z_{ij} + z_{ik} \le 1, && \forall j \in D, k \in d_j^O, i \in Z && (4.14)\\
& m_i \notin d_j^M \implies m_{ij} = 0, && \forall i \in M, j \in D && (4.15)\\
& \sum_{i\in M_k} m_{ij} = |d_j^{M_k}|, && j \in D, k \in d_j^M && (4.16)\\
& m_{ij} + m_{ik} \le 1, && \forall j \in D, k \in d_j^O, i \in M && (4.17)\\
& w_{ijkl} \in \{0,1\}, && \forall i \in W, j \in T, k \in D, l \in d_k^P && (4.18)\\
& s_{jkl} \in \{0,1\}, && \forall j \in T, k \in D, l \in d_k^P && (4.19)\\
& m_{ij} \in \{0,1\}, && \forall i \in M, j \in D && (4.20)
\end{aligned}
$$

$$z_{ij} \in \{0, 1\}, \qquad \forall i \in Z, j \in D \tag{4.21}$$

$$occ_i = \sum_{j \in T} \sum_{k \in D} \sum_{l \in d_j^P} w_{ijkl}, \qquad \forall i \in W \tag{4.22}$$

The objective function is stated in (4.1), it is split in multiple parts, it minimizes (i) the number of different workers for every position between periods of that demand (4.1a), $min(\sum_{j \in T} w_{ijkl}, 1)$ is one if the worker i is working for that position at that time, 0 otherwise. Hence, the sum of that value for all worker will be equal to the number of worker for that shift; (ii) the number of *sentinel* worker assigned to demands (4.1b); (iii) the number of violations of working requirements (4.1c).

Constraint (4.2) ensures that each position is filled by only one worker. The sentinel worker being a valid assignment is also part of the sum.

Constraint (4.3) ensures that no worker works for multiple demands at the same time period.

The constraints (4.4) and (4.5) ensures that no worker is working for a demand thais not occurring or when is himself not available.

The constraints (4.6) and (4.7) fullfil the same role as (4.4) and (4.5) but for sentinel variables.

Constraint (4.8) ensures that no incompatible workers work together while (4.9) ensures that no incompatible pair of worker and client work together.

Constraint (4.10) ensures that no worker work for a position in which they are not qualified to work at. Constraint (4.11) ensures that for each additional skills, at least one worker in the group has that skill.

Constraint (4.12) ensures that no zone is assigned to a demand in which this zone is not a possible assignment. (4.13) ensures that only one zone is assigned to this demand if this demand is in need of a zone. Constraint (4.14) ensures that no zone is assigned to two overlapping demands in time.

Constraint (4.15) ensures that no machine is assigned to a demand not in need of that machine. Constraint (4.16) ensures that the required number for each machine is satisfied. And again, (4.17) ensures that no machine is assigned to two overlapping demands in time.

Finally (4.18), (4.19), (4.20) and (4.21) ensure the variables only takes binary values.

4.3 Constraint Programming Model

The translation to the mathematical (MIP) model to the CP model is fairly straightforward. Binary variables are translated to integer variables, each value representing one resource (i.e. worker, zone or machines). For example, binary variables $w_{0jkl}, \ldots, w_{njkl}$ are transformed to a single variable $w_{jkl} \in \{0, \ldots, n\}$

4.3.1 Variables

First, we need to express the set of workers for each demand at each time period in which that demand occurs.

$$w_{ijk} \in W \tag{4.23}$$

(4.23) is the worker working at time i for demand j at the k^{th} position with $t_i \in T$, $d_i \in D$, $t_i \in d_j^T$ and $k \in d_j^P$. This is done by using a 3-dimensional array of variables. The first dimension being the indices of the time periods, the second dimension is the indices of the demands while the last dimension is the list of worker variables. This last dimension has the size of the number of required workers for that demand.

The same reasoning is used for zones and machines:

$$m_{ij} \in M \tag{4.24}$$
$$z_i \in Z \tag{4.25}$$

(4.24) is the j^{th} machine used for demand i while (4.25) is the zone used for demand i

Some constraints are already satisfied by the modeling of the variables, like the number of required resources (i.e. worker, location, machine) per demand.

4.3.2 Complete Model

For this model, we define additional notations:

- N_{jk} denotes the number of different workers for shift k of demand j, these are variables added separete to the decision variables.
- $W_{jk} = \{w_{ijk} \mid i \in d_j^T\}$ denotes the set of worker variables for demand j at position k accross all time periods of that demand.
- $X_i = \{w_{ijk} \mid j \in D, k \in d_j^P\}$ denotes the set of worker variables for all the demands accross time period t_i
- $Z_i^O = \{z_j \mid j \in d_i^O\}$ denotes the set of zone variables for demands that overlap in time with d_i
- $M_i^O = \left\{m_{jk} \mid j \in d_i^O, k \in \{0, \ldots, |d_j^M| - 1\}\right\}$ denotes the set of machine variables for demands that overlap in time with d_i

– P_{ij} denotes the set of permutations of pairs of worker variables for a demand j at time i.

$$
\begin{aligned}
\min \quad & \sum_{j \in D} \sum_{k \in D_j^P} N_{jk} && && (4.26)\\
\text{s.t} \quad & \texttt{atLeastNValue}(W_{jk}, N_{jk}), && \forall j \in D, k \in d_j^P && (4.27)\\
& \texttt{allDifferent}(X_i), && \forall i \in T && (4.28)\\
& \texttt{allDifferent}(Z_i^O), && \forall i \in D && (4.29)\\
& \texttt{allDifferent}(M_i^O), && \forall i \in D && (4.30)\\
& \texttt{notEqual}(w_{ijk}, w), && \forall (w,c) \in I_{wc}, && (4.31)\\
& && i \in T, j \in D_c, k \in d_j^P && \\
& \texttt{negativeTable}(a, b, I_{ww}), && \forall (a,b) \in P_{ij} && (4.32)\\
& w_{ijk} \in W_{d_j^{s_k}} \cap \{w \mid t_i \in w^T\}, && \forall j \in D, i \in d_j^T, k \in d_j^P && (4.33)\\
& z_i \in d_i^Z, && \forall i \in D && (4.34)\\
& m_{ij} \in \{m \mid m \in W \wedge m = d_i^{M_j}\}, && \forall i \in D, j \in d_i^M && (4.35)
\end{aligned}
$$

This model is simpler than the mathematical one described Section 4.2. It needs less constraints to express the same problem. For example, (4.33) expresses multiple constraints (i.e. workers are restricted to positions with respect to their skills and to their availabilities) in only one step which is the initialization of the variable. Hence, no constraints will run during the solving process.

(4.34) expresses that the domain of zone variables are limited to the possible zones of a demand and (4.35) states that the domain of each machine variables for a demand are limited to the possible machines for that need.

(4.28) states that no workers should work for two positions at the same time. Constraint (4.31) states the incompatibilities between clients and workers and restricts workers to work for incompatible clients while (4.32) prevent incompatible workers to work together.

Constraint (4.29) and (4.30) state that no zones and machines should be used for two overlapping demands.

Finally, for the objective, constraint (4.27) states that N_{jk} will be equal to the number of different workers for the same position throughout time periods. The objective itself (4.26) is the minimization of the sum of all N_{jk}, hence minimizing the number of change between shifts.

Chapter 5

Implementation (TODO title)

In this chapter, we describe our implementation for the models presented in Chapter 4. The implementation is done in Scala using *OscaR* (3.2.1) for the Constraint Programming model and *Gurobi Optimizer* (3.1.1) for the Mixed Integer Programming model.

Chapter 6

Experiments

Chapter 7

Conclusion

Bibliography

[1] E. K. Burke, P. De Causmaecker, G. V. Berghe, and H. Van Landeghem, "The state of the art of nurse rostering," *Journal of Scheduling*, vol. 7, pp. 441–499, Nov 2004.

[2] "Gurobi - MIP Basics." `http://www.gurobi.com/resources/getting-started/mip-basics`. Accessed: 2019-03-13.

[3] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018.

[4] P. van Beek and X. Chen, "Cplan: A constraint programming approach to planning," in *AAAI/IAAI*, 1999.

[5] OscaR Team, "OscaR: Scala in OR," 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**