

## WORKER ALLOCATION PROBLEM

### 1. PROBLEM DESCRIPTION

This problem was introduced in [1, sec. 9.2.1]. The construction of a house is divided into a numbers of tasks, each requiring a set of workers with different skills (e.g. tiling or masonry). Hiring a worker whose skills meet the demands of a given task is subject to a cost which depends on the individual qualifications. The objective is to select a set of workers to perform all the tasks (i.e. to cover the set of tasks) such that the overall cost is minimized.

---

```
range Boolean 0..1;
int nbWorkers = ...;
range Workers 1..nbWorkers;
enum Tasks ...;
{Workers} qualified[Tasks] = ...;
int cost[Workers] = ...;

var Boolean hire[Workers];
minimize
    sum(c in Workers) cost[c]*hire[c]
subject to
    forall(j in Tasks)
        sum(c in qualified[j]) hire[c] >= 1;

{Workers} crew = {c| c in Workers: hire[c] = 1};
display crew;
```

FIGURE 1. Worker allocation problem in OPL

---

### 2. OPL MODEL

The problem model in figure 1 is taken from [1, sec. 9.2.1] and reveals itself as an Integer program. The basis is the data-type `Boolean`, an alias for the set  $\{0, 1\}$ . The main decision variable is `hire` which indicates whether a certain worker has been hired (1) for a task or not (0). The `minimize` statement sums up the costs of all workers that have been hired. In the constraints block, the covering of the set `Tasks` is achieved by asserting that at least one worker is hired to accomplish it. The last two lines serve for pretty-printing purposes, `crew` selects all those `Workers` that have indeed been hired, this set is subsequently printed using the `display` statement. For the instance data and results, refer to [1, sec. 9.2.1].

## 3. Z MODELS

We provide two models here. First, to allow a comparison to the model in [1](#) on the previous page, a close translation is performed. However, since belonging to the class of set-covering problems, a relational format is a more natural choice for modelling and so be realized in the second model. In both models, we use a generic set *Tasks*.

[*Tasks*]

We model the *Workers* as a range of Integer numbers.

$$\begin{array}{|l} nbWorkers : \mathbb{N} \\ Workers : \mathbb{F} \mathbb{N} \\ \hline Workers = 1 .. nbWorkers \end{array}$$

Being part of the instance data, we also introduce the *costs* function that associates the remuneration payable

$$| \text{cost} : Workers \rightarrow \mathbb{N}$$

to each worker.

**3.1. Direct analogue.** We begin with a schema for the problem constraints. Notice the close correspondence to figure [1](#) on the preceding page. The  $\Sigma$  operator uses the ‘bag image’ defined in the Appendix. Its effect is equivalent to the usual sum operator.

$$\begin{array}{|l} WAP \\ \hline qualified : Tasks \rightarrow (\mathbb{F} Workers) \\ hire : Workers \rightarrow \{0, 1\} \\ crew : \mathbb{F} Workers \\ \hline \forall j : Tasks \bullet \sum (hire \parallel qualified(j)) \geq 1 \\ crew = \text{dom}(hire \triangleright \{1\}) \end{array}$$

Although not an essential part of the problem formulation, we have also included how the subset *crew* translates into Z.

We continue with the optimisation part. The *objective* function exploits the fact that *cost*, *hire* are both sequences thanks to the definition of *Workers* as a range of natural numbers. We perform a sort of vector arithmetic, effectively multiplying both sequences and summing over the result. For the details, please see the Appendix.

$$\begin{array}{|l} WAP\_Optimisation\_Part \\ \hline objective : WAP \rightarrow \mathbb{N} \\ solution : WAP \\ \hline \forall w : WAP \bullet objective(w) = \text{sumseq}(cost \otimes w.hire) \\ objective(solution) = \min(objective(WAP)) \end{array}$$

As required in this problem, the *solution* exhibits a minimal value of the *objective* function.

**3.2. A relational model.** Since a worker may have several qualifications and since each task can performed by several workers, it is natural to model *qualified* as a relation. Further, we dispense with the 0/1 array *hire* and use the set *crew* directly instead.

---

*RWAP*


---

 $qualified : Workers \leftrightarrow Tasks$ 
 $crew : \mathbb{F} Workers$ 


---

 $qualified \langle crew \rangle = Tasks$ 


---

The single unquantified expression in the predicate block captures the problem constraint; relational image of the set *crew* through *qualified* must yield (cover) the set *Tasks*. This problem formulation is more succinct.

---

*RWAP\_Optimisation\_Part*


---

 $objective : RWAP \rightarrow \mathbb{N}$ 
 $solution : RWAP$ 


---

 $\forall r : RWAP \bullet objective(r) = \sum(cost \parallel r.crew \parallel)$ 
 $objective(solution) = \min(objective \langle RWAP \rangle)$ 


---

For the optimisation part, only a minor modification is required, it is noticeably similar to the previous one.

#### REFERENCES

- [1] Pascal Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, January 1999.