CS4402 PRACTICAL 1 REPORT

200010781

26-10-2021

Content

1 Introduction 2
2 Model 2
    2.1 Problem Analysis 2
    2.2 Basic Model 3
        2.2.1 Descriptions 3
        2.2.2 Constraints 5
        2.2.3 Solutions 7
        2.2.4 Performance 8
    2.3 Improved Model 8
        2.3.1 Descriptions 8
        2.3.2 Constraints 9
        2.3.3 Solutions 10
        2.3.4 Performance 10
3 Evaluation 11
4 Extension 11
5 Conclusion 12

1 Introduction

Aiming at getting a solution for SPCP problem, I come up with two models. The first model is a basic one, which meets all constraints and can get a solution but with bad efficiency for reaching a solution for the third example in short time. Therefore, I improved it as the second model, which is more efficient. Both models have symmetry breaking added. Besides, I set variable order in the second model and did experiments to evaluate the performance of different optimization levels in improved model.

According to the results, the improved model saves about 60% of solver solving time and reduces approximately 75% of the number of solver searching nodes. Models in different optimization levels have different performance.

Section 2 introduces the basic model, while the improved model is described in section 3. Section 4 illustrates some extension I made. Finally, section 5 shows my conclusion.

2 Model

2.1 Problem Analysis

According to The Solar Power Company Problem (SPCP) description, there are four objects in this problem: panels, buildings, workers and working day. Each of them has some attributes:

- **Panels**: Power Generated and Area
- **Building**: Power Demand and Roof Area
- **Worker**: Daily Area Installed and Daily Salary
- **Working Day**: Daily Budget

It is obvious that there are several constraints in this problem:

1. Each panel need to be assigned to only one building for one worker in one day.
2. Total area of panels on one building cannot be greater than the building's roof area.
3. Each worker has a workload limit. The daily workload of each worker shall not exceed the maximum capacity of this worker.
4. Payment for workers per day cannot surpass the daily budget.
5. Minimize the squared difference of power produced and power required for each building.

The relationships between these four objects can be represented as the diagram shown in Figure 1.

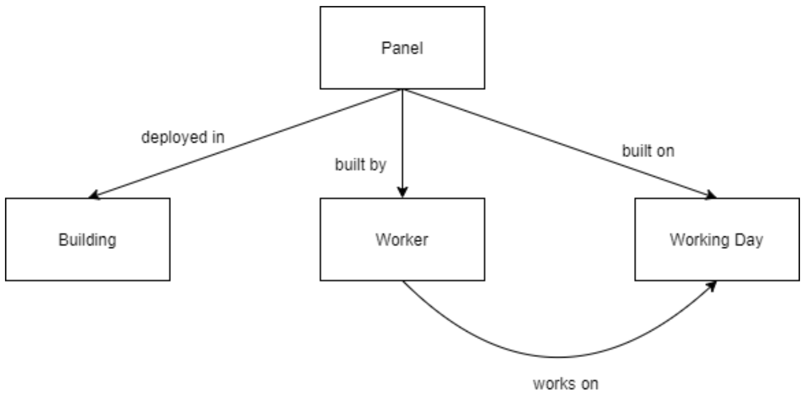

Figure 1 The relationships between objects

2.2 Basic Model

2.2.1 Descriptions

Based on previous analysis, it is clear that coming up with four matrices to represent the relationship of each two objects is a possible approach for getting a solution. Therefore, in this basic model, I use four matrices to represent the relationship between two objects and one matrix to store the square difference of each building and then program the relevant constraints to find a solution.

1. **Modelling Representation:**

Generally, there are 3 ways for modelling – explicit modelling, occurrence modelling and combined modelling of both. In this practical, I choose to use occurrence modelling. This is because each panel is assigned to only one building, consisting a one-to-one match. It is easier to model in occurrence representation.

2. **Variables and Domains:**

Four basic decision variables need to be found as shown below, and the domain of all these variables is 0 to 1 for choosing occurrence representation. Besides, in order to minimize the power shortfall, an auxiliary variable is also been added to represent the square difference of power produced and power required for each building.

- **panelBuild:** A matrix represents that each panel should be deployed in which building

panelBuild		**Panel**			
		1	**2**	**…**	**numPanels**
Building	**1**	(0,1)	(0,1)	(0,1)	(0,1)
	2	(0,1)	(0,1)	(0,1)	(0,1)
	…	(0,1)	(0,1)	(0,1)	(0,1)
	numBuilds	(0,1)	(0,1)	(0,1)	(0,1)

➢ 1 in panelBuild means a panel (column number) belongs to a building (row number), while 0 has the opposite meaning.

- **panelWorker:** A matrix represents that each panel should be built by which worker

panelWorker		**Panel**			
		1	**2**	**…**	**numPanels**
Worker	**1**	(0,1)	(0,1)	(0,1)	(0,1)
	2	(0,1)	(0,1)	(0,1)	(0,1)
	…	(0,1)	(0,1)	(0,1)	(0,1)
	numWorkers	(0,1)	(0,1)	(0,1)	(0,1)

  - ➢ 1 in panelWorker means a worker (row number) is responsible for a panel (column number), while 0 has the opposite meaning.

- **panelDay:** A matrix represents that each panel should be built on which working day

panelDay		**Panel**			
		1	**2**	**…**	**numPanels**
Day	**1**	(0,1)	(0,1)	(0,1)	(0,1)
	2	(0,1)	(0,1)	(0,1)	(0,1)
	…	(0,1)	(0,1)	(0,1)	(0,1)
	numDays	(0,1)	(0,1)	(0,1)	(0,1)

  - ➢ 1 in panelDay means a panel (column number) need to be built on a day (row number), while 0 has the opposite meaning.

- **workerDay:** A matrix represents that each worker should work on which working day

workerDay		**Day**			
		1	**2**	**…**	**numDays**
Worker	**1**	(0,1)	(0,1)	(0,1)	(0,1)
	2	(0,1)	(0,1)	(0,1)	(0,1)
	…	(0,1)	(0,1)	(0,1)	(0,1)
	numWorkers	(0,1)	(0,1)	(0,1)	(0,1)

  - ➢ 1 in workerDay means a worker (row number) should work in a day (column number), while 0 has the opposite meaning.

- **Differences**: (Auxiliary Variable) A matrix to store the square difference of power produced and power required for each building

Building			
1	**2**	**…**	**numBuilds**
(0,…,10000)	(0,…,10000)	(0,…,10000)	(0,…,10000)

2.2.2 Constraints

1. Basic Constraints

- **Each panel need to be assigned to only one building for one worker in one day.**

  For this constraint, we must make sure sum of each columns in panelBuild, panelWorker and panelDay equals 1. In p1Model1.eprime file, we describe this constraint in the following way:

```
$ Sum of each columns equals 1
forAll panel : int (1..numPanels) .
   ( sum i : int (1..numBuilds) . panelBuild[i,panel] ) = 1 /\
   ( sum i : int (1..numWorkers) . panelWorker[i,panel] ) = 1 /\
   ( sum i : int (1..numDays) . panelDay[i,panel] ) = 1,
```

- **Total area of panels on one building cannot be greater than the building's roof area.**

  According to this constraint, for each building(row) in panelBuild, we must make sure that sum of area of panels where value is 1 is no more than the roof area of this building. Since I use occurrence representation, it is better to get the sum of multiplication of the value in panelBuild and panel area in corresponding. This is because the sum of multiplication cannot be affected by value 0. In p1Model1.eprime file, we describe this constraint in the following way:

```
$ Sum of area of panels with non-empty value in panelBuilds <= building area
forAll building: int(1..numBuilds) .
   ( sum panel : int(1..numPanels) .
      panelArea[panel] * panelBuild[building, panel] ) <= buildArea[building],
```

- **The daily workload of each worker shall not exceed the maximum capacity of this worker.**

  According to this constraint, we must make sure that all workers' workloads should not exceed their daily maximum workloads (worker capacity) in each day. Therefore, for each day(row) in panelDay and for each worker(row) in panelWorker, if value in the same column for both tables are 1, which means this worker built this panel in this day, we must make sure the sum of all the panel area that the worker is responsible for on that day is no more than the worker's daily capacity. We can get sum of multiplication in this situation as well, and the constraint is described in the following way:

```
$ Panel area must not be greater than worker capacity
forAll days : int(1..numDays) .
  forAll worker : int(1..numWorkers) .
   ( sum panel : int(1..numPanels) .
      panelDay[days, panel] * panelWorker[worker, panel] * panelArea[panel] ) <=
         workercapacity[worker],
```

- **Payment for workers per day cannot surpass the daily budget**

  I use workerDay to represent the working day of each worker and also do channeling to combine the auxiliary variable with other variables.

For this constraint, we can sum up the worker cost of workers who are working on the same day (column number) and make sure it is no greater than daily budget. The constraint in P1model1.eprime file is shown below:

```
$ Sum of worker cost of with non-empty value in workerDay <= budgetPerDay
forAll day : int(1..numDays) .
  ( sum worker : int(1..numWorkers) .
    workerDay[worker, day] * workercost[worker] ) <= budgetPerDay,
```

Besides, the value of workerDay depends on the values of panelWorker and panelDay. For each panel in panelWorker and panelDay, if both value in these two matrices is 1, then the value for the day and the worker in workerDay is 1.

```
$ Channeling
forAll worker : int(1..numWorkers) .
  forAll panel : int(1..numPanels) .
    forAll day : int(1..numDays) .
      ( ( panelWorker[worker, panel] != 0 ) /\ ( panelDay[day, panel] != 0 ) ) ->
        workerDay[worker, day] = 1 ,
```

- **Minimize the squared difference of power produced and power required for each building**
In order to minimize the square difference for each building, we need to find the minimum sum of all the square difference for all buildings. Therefore, I create an auxiliary variable differences to store the square difference of each buildings and minimize the sum of this matrix. This constraint is described below:

```
$ Minimize the sum of all the square difference for all buildings
$ Get the square difference of each buildings
forAll building : int(1..numBuilds) .
 differences[building] =
  ((sum panel: int(1..numPanels) .
    panelPower[panel] * panelBuild[building, panel] ) - buildPower[building])**2,
```

2. **Additional Constraints:**
As for this problem, it can be known that panels to be built on which days and workers to work on which working day will not lead differences. For example, assigning panel 1 to be built on day 1 or day 2 is the same and will not cause power or payment differences. Therefore, I add lex constraint for symmetry breaking.

```
$ Breaking symmetry
forAll d : int(1..numDays-1) .
  forAll dLater : int(d+1..numDays) .
    (panelDay[d, ..] <=lex panelDay[dLater, ..] \/
    panelDay[d, ..] >=lex panelDay[dLater, ..] ) ,
```

2.2.3 Solutions

1. **Spcp.param**

   Corresponding solution is shown in Table 1 and Table 2.

Table 1 Solution for the first instance about panel and building assignments

Building	**Panels**	**Power Generated**	**Power Demand**	**Power Shortfall**
1	2,3	32	32	0
2	1,4	28	30	2

Table 2 Solution for the first instance about workers and working day

Worker	**Capacity**	**Panels day 1**	**Area Installed day 1**	**Panels day 2**	**Area Installed day 2**	**Salary day 1**	**Salary day 2**
1	8	-	-	-	-	-	-
2	9	3,4	9	1,2	9	30	30

The total square difference for all buildings is 4, and the total payment for all worker salary for each day is 30 and 30 respectively.

2. **Spcp2.param**

   Corresponding solution is shown below in Table 3 and Table 4.

Table 3 Solution for the second instance about panel and building assignments

Building	**Panels**	**Power Generated**	**Power Demand**	**Power Shortfall**
1	4,7	20	30	10
2	2	18	25	7
3	3	13	22	9
4	1,6	23	31	8
5	5	10	17	7

Table 4 Solution for the second instance about workers and working day

Worker	**Capacity**	**Panels day 1**	**Area Installed day 1**	**Panels day 2**	**Area Installed day 2**	**Salary day 1**	**Salary day 2**
1	8	6	4	5	5	20	20
2	9	-	-	-	-	-	-
3	10	3,4,7	10	1,2	9	30	30

The total square difference for all buildings is 343, and the total payment for all worker salary for each day is 50 and 50 respectively.

3. **Spcp3.param**

   For this model, it is hard to get a solution for the third example in a short time.

2.2.4 Performance

1. **Spcp.param**

   Data in spcp.param.info is shown below.

Solver Nodes	Solver Solve Time	Savile Row Total Time	Solver Solutions Found
18	0.000224	0.16	2

2. **Spcp2.param**

   Data in spcp2.param.info is shown below.

Solver Nodes	Solver Solve Time	Savile Row Total Time	Solver Solutions Found
72159	0.421984	0.264	71

3. **Spcp3.param**

   For this model, it is hard to get a solution for the third example in a short time. Therefore, the performance of this model is not good.

2.3 Improved Model

2.3.1 Descriptions

According to previous part, the performance of the basic model is not good since it cannot get a solution for the third example. Therefore, aiming at improving the performance, I improved my basic model to a more efficient one.

From Figure 1, it is clear that there exists a circle in panel, worker and day. Hence, it is possible to create a three-dimension matrix to represent their relationships. In this improved model, I combine panelWorker and panelDay matrices into a single matrix panelWorkerDay to represent the relationships of these three objects, and cut down the workerDay matrix, which is redundant since panelWorkerDay includes the relationship between workers and working days.

1. **Variables and Domains:**

   In total, there are three variables: panelBuild, panelWorkerDay, and differences. Two of them are the same as in basic model (panelBuild and differences) as shown in 2.2.1. The different one is panelWorkerDay, which is described below.

- **panelWorkerDay:** A matrix represents that each panel should be built by which worker on which day

panelWorkerDay		**Worker**			
		1	**2**	**…**	**numWorkers**
Panel	**1**	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]
	2	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]
	…	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]
	numPanels	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]	[(0,1),…,(0,1)]

- ➢ The length of [(0,1),…,(0,1)] is numDays.

- 1 in panelWorkerDay means a panel is assigned to a worker on a day, while 0 has the opposite meaning.

2.3.2 Constraints

1. Basic Constraints

- **Each panel need to be assigned to only one building for one worker in one day.**
  The constraint is programmed as below:

```
forAll panel : int (1..numPanels) .
   (sum i : int (1..numBuilds) . panelBuild[i,panel] ) = 1 /\
   (sum ( flatten( panelWorkerDay[panel,.. , ..] ))) = 1,
```

- **Total area of panels on one building cannot be greater than the building's roof area.**
  This constraint is programmed the same as the original model.

- **The daily workload of each worker shall not exceed the maximum capacity of this worker.**
  For this constraint, we must make sure for each day, sum of panel area to be done by each worker should not be greater than workers' capacity. The code in p1Model2.eprime file is shown as below.

```
$ Panel area must not be greater than worker capacity
forAll day : int(1..numDays) .
  forAll worker : int(1..numWorkers) .
    (sum panel : int(1..numPanels) .
      panelWorkerDay[panel , worker , day] * panelArea[panel]) <=
        workercapacity[worker],
```

- **Payment for workers per day cannot surpass the daily budget**
  We need to use panelWorkerDay to represent this constraint since there is no workerDay matrix here. Find the maximum value in panelWorkerDay for each worker in each day is important. This is because the maximum value means whether a worker works on this day and need salary or not. 1 for the maximum value means a worker did some works on this day, which 0 means opposite. The constraint can be programmed as below.

```
$ Maxmum value in panelWorkerDay in each day * workercost <= budgetPerDay
forAll day : int(1..numDays) .
  ( sum worker : int(1..numWorkers) .
        max( flatten( panelWorkerDay[.. , worker , day])) * workercost[worker]) <=
        budgetPerDay,
```

- **Minimize the squared difference of power produced and power required for each building**
  This constraint is programmed the same as the original model.

2. **Additional Constraints**

Symmetry breaking for this model is a little bit different from the original one.

```
$ Breaking symmetry
forAll d : int(1..numDays-1) .
  forAll dLater : int(d+1..numDays) .
    flatten( panelWorkerDay[.. , .. , d]) <=lex flatten( panelWorkerDay[.., .., dLater])
      \/
    flatten( panelWorkerDay[.. , .. , d]) >=lex flatten( panelWorkerDay[.., .., dLater]),
```

2.3.3 Solutions

1. **Spcp.param**

   Solutions found by this model is the same as the basic model.
2. **Spcp2.param**

   Solutions found by this model is the same as the basic model.
3. **Spcp3.param**

   Corresponding solution is shown in Table 5 and Table 6.

Table 5 Solution for the third instance about panel and building assignments

Building	Panels	Power Generated	Power Demand	Power Shortfall
1	2,6,8	30	32	2
2	3,4	27	30	3
3	9,10	20	20	0
4	1,7	28	30	2
5	5	10	10	0

Table 6 Solution for the third instance about workers and working day

Worker	Capacity	Panels day 1	Area Installed day 1	Panels day 2	Area Installed day 2	Panels day 3	Area Installed day 3	Salary day 1	Salary day 2	Salary day 3
1	10	-	-	4,9	10	-	-	-	30	-
2	8	10	8	5	8	3,8	8	20	20	20
3	9	6,7	9	-	-	1,2	9	25	-	25

The total square difference for all buildings is 17, and the total payment for all worker salary for each day is 45, 50 and 45 respectively.

2.3.4 Performance

1. **Spcp.param**

   Data in spcp.param.info is shown below.

Solver Nodes	Solver Solve Time	Savile Row Total Time	Solver Solutions Found
24	0.000125	0.129	2

2. **Spcp2.param**

Data in spcp2.param.info is shown below.

Solver Nodes	Solver Solve Time	Savile Row Total Time	Solver Solutions Found
17063	0.146147	0.199	71

3. **Spcp3.param**

Data in spcp3.param.info is shown below.

Solver Nodes	Solver Solve Time	Savile Row Total Time	Solver Solutions Found
71055	0.320488	0.263	53

3 Evaluation

- **Solutions**

  From previous parts, we can see that both models get same solutions and all solutions respect constraints.
- **Performance**

  I compared the output data of both models in the second example (spcp2.param) as shown in the table below. It can be seen that improved model saves about 60% of solver solving time and reduces approximately 75% of the number of solver searching nodes, which means it is more efficient than the basic one.

Table 7 Performance of two models in the second instance

	Solver Nodes	Solver Solve Time	Savile Row Total Time	Solver Solutions Found
Basic Model	72159	0.421984	0.264	71
Improved Model	17063	0.163132	0.202	71

4 Extension

1. Set variable order in searching

   In this practical, except adding symmetry breaking in constraints, I also applied **branching on** statement of Savile Row in the improved model where exists two branching variables - panelBuild and panelWorkerDay.

2. Use various optimization levels

   There are 4 types of optimization levels are provided, and their ranks increases with the number increasing. The default level is O2, which means if there is no optimization level assigned, the solver will set the optimization level automatically into level 2. I did experiments on using different optimization levels in the third instance and compared the output results of different levels in Table 8.

   O0 is the lowest level. Compared with other levels, the solver spends more time in searching for solutions at O0. The impact of O1 and O2 seems quite similar. Although O1 has the smallest Savile Row total time, we cannot conclude that O1 has the best optimization. I suspect that Savile Row total time also can be affected by hardware and memory usage of machines. Besides, it is noteworthy that O3 reduces 4 solver nodes than others. I think the reason is because O3 eliminates some commutative common subexpression, which reduces some nodes eventually.

Table 8 Performance of improved model with different optimization levels in spcp3.param

	Solver Nodes	**Solver Solve Time**	**Savile Row Total Time**	**Solver Solutions Found**
-O0	71055	0.626396	0.245	53
-O1	71055	0.344249	0.204	53
-O2 (Default)	71055	0.320488	0.263	53
-O3	71051	0.341806	0.456	53

5 Conclusion

In this practical, I propose a basic model and an improved model with symmetry breaking added. Both models respect all constraints and have the same solutions. However, for the basic model, it can reach solutions that meet all constraints but with bad efficiency. We cannot get solutions for the third instance (spcp3.param) in this model during a short time. After making some enhancements, I propose a more efficient model which saves about 60% of solver solving time and reduces approximately 75% of the number of solver searching nodes.

For extension part, I set variable order for searching process in the improved model and did experiments on using different optimization levels to explore their performance. It is clear that O0 is the lowest optimization level and needs more solve time. O3 is the highest optimization level and can reduces some solver nodes for eliminating some common subexpression.